

---

# **pydanny-event-notes Documentation**

***Release 45***

**Daniel Greenfeld**

**Jan 24, 2018**



---

## Contents

---

<b>1</b>	<b>Conferences</b>	<b>3</b>
<b>2</b>	<b>Hackathons</b>	<b>509</b>
<b>3</b>	<b>Meetups</b>	<b>513</b>
<b>4</b>	<b>Articles</b>	<b>581</b>
<b>5</b>	<b>Credits</b>	<b>583</b>
<b>6</b>	<b>Indices and tables</b>	<b>585</b>



I've been collecting my notes taken over the years into this one location. My intention is to capture all that I've learned or forgotten, share it with others, and compare that with where I am now.



### 1.1 EuroPython 2013

**venue** Florence, Italy

**dates** July 1 - July 7

---

**Note:** The first day of the conference we were asked to give a 3-hour tutorial. We put something together in 48 hours, but that means I didn't get many notes in. Nevertheless, it's been a great conference. :-)

---

#### 1.1.1 Experiences from Teaching Physics with iPython Notebook

By Anders Lehmann

- Associate Professor, AARHUS UNIVERSITY
- Denmark
  - Winner of EuroVision
  - Birthplace of Søren Kierkegaard

##### Intro

- ipython is an awesome tool for teaching
- But could do things better

##### Teaching Physics

- Physics is considered hard by students

- What is the use?
- ElectroMagnetics is abstract
- What is a field?
- How do semiconductors work?

### What's good about ipython notebook?

- Great tool
- Feels lightweight
- Like MatLab in a browser that starts counting at zero!

### Physics extension

- Adds physical quantities
- Adds physical constants
- Enables check if Units match
- Started by: `%load_ext physics`

### How do you make physics interesting?

- Use real life examples demonstrated in ipython notebook
- Be exciting and fun

### Experiment

- Introduce ipython as an online calculator
- Show that it can use units
- Extend by adding small functions
- Introduce plotting

## 1.2 DjangoCon Europe 2013 (Django Circus)

- Location: Warsaw, Poland
- URL: <http://djangocircus.com>
- Other people's notes:
  - <http://reinout.vanrees.org/weblog/tags/djangocon.html>
  - <http://foobacca.github.io/foobacca-event-notes/DjangoConEurope2013/index.html>



### 1.2.1 Opening Statements

- Kuba thanked the sponsors and community
- Russell Keith-Magee gave a wonderful eulogy of Malcolm Tredinnick, who passed away on March 17th, 2013.

### 1.2.2 Django Circus

#### Keynotes

##### First Keynote: On the Revolutions

- by Brandon Rhodes

##### Topic: Nichola Kopernik

- Polish Astronomer
- Lifted the **Earth** into the **heavens**, rather than the Earth was at the bottom of the heavens.
- Lived:

##### Near-Earth environment

- 300s BC - Aristotle - spherical Earth
- 200s BC - Erathosthenes - radius of Earth
- 100s BC - Hipparchus - distance to Moon

##### Hard to tell how things worked out

- Planets move slowly across the sky
- Retrograde motion
- Didn't make sense, so Ptolomy came up with a sophisticated way of handling this based off of observations.
- People followed this for thousands of years because it matched empirical evidence.
- It was the lack of evidence against Ptolomy that made the case against Galileo.

##### Kopernik debated Ptolomy

Kopernik read:

- Galileo
- Kepler's ellipses
- Newton theories

## Bradley in 1725 proved Kopernik

- Stellar aberration
- Speed of light
- Watched the stars move in relation to the Earth's movement

---

**Note:** “It took 1,900 years after the Greeks discovered the distance to the moon for us to determine the distance to a star.” – Brandon Rhodes

---

## Why did Kopernik debate all the physical evidence for the Earth-centric universe?

- Beauty
- Wanted better math
- Made the code pretty

---

**Note:** “Kopernik made the most awesome code refactor in history.” – Brandon Rhodes

---

Kopernik's model of the solar system made it clear that the Sun was the center of the solar system instead of yet another object.

TODO - get Brandon's code models for the Ptolomic and Kopernik solar system models.

## Copernican Refactor

Brandon's new term for any refactor that brings things to the center.

Example: Gadgets that combine automobile cigarette lighters with a USB connector to put power into mobile devices.

- Clean Architecture (<http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html>)
- Docopt (<https://pypi.python.org/pypi/docopt>)
- Django (compared to Python CGI)

## Closing

If your code is driving you crazy, think like Kopernik and turn the world upside down.

## Second Keynote: Why Django Is Awesome

- by Daniel Greenfeld aka “Pydanny”

## Background

- Principal at Cartwheel Web
- Co-author of Two Scoops of Django (<https://django.2scoops.org>)

- Fiance of Audrey Roy :)

## Why Django is Awesome

### Django Is Everywhere

NASA, PBS, Instagram, etc.

### Django Lets Us Use Python

Python is awesome!

- Python has Zen
- Python has a style guide
- And indentation and such

### Django Has Awesome APIs

Requests is known for having an elegant API. Look at sample code from using the Django test client. It's practically the same code!

- Example of models and model methods

API conventions encourage clean design:

- Make it easy to separate content from presentation
- Helps us get things done

### Django Views are Functions

Function views are very simple. Request -> Response

Even with class-based views, you have `View.as_view()` which is basically the same thing.

### Django Has Awesome Features

- The admin is what we're known for. Aka "How to sell Django 101".
- Shortest admin module possible in 3 lines.
  - Don't hack the admin to force it to bend to NoSQL. Write separate code
  - Pydanny will be sprinting on django-admin2

### Django's Full Stack Is Awesome

- Dominates hackathons because you have everything in one place
- Building companies is easy with stock Django/Python. Even if you don't understand the larger Django ecosystem, you can get amazing things done with it.

## Django Is Part of an Ecosystem

- 30,000 packages on PyPI
- 1,750 packages on <https://www.djangopackages.com/>

## Django Sets the Bar for Documentation

- In the Python world, no one says “just read the code” anymore.
- Largely because of Django

## Django’s Community is the Best

- Humble and listens to criticism
- The more you help people in the community, the more the community helps you. Case study: book had over 125 contributors. We gave free copies and asked the community to do nice things in return. Huge response: readers volunteered their time to write Django projects for churches, schools, other good deeds around the world.

## Call To Action

Be awesome. “**I want us to change the world.**” Use your knowledge of Django to help people and do good things.

## Circus: process & socket manager

- By Tarek Ziade
  - Works for Mozilla
  -

Circus is a process manager we developped at Mozilla while working on high scalability, we wanted to have a way to deal with our processes directly from python, and in a better way that’s what possible with the standard library.

Circus uses zeromq in its internals, and thus is easily extensible. We’ll present you how you why we built circus, how to use it and some core concepts that were useful in the conception of the tool. Also, we’ll demonstrate how easy it is to plug circus with a Django stack.

## Typical Django Deployment

- Nginx > Gunicon > Django + sentry + celery

Supervisord is frequently used for management of these components. Alternatives include:

- Bluepill (less mature)
- upstart (system level - root access needed)
- daemontools (low-level like upstart)
- got, monit, runit, etc.

## Missing features from supervisord

- Realtime stdout/stderr
- realtime stats
- powerful web console
- Remote access
- clustering
- event-based plugins

Since those were missing, Tarek launched his own project: Circus!

## Technical choices for Circus

- Python
- PSUTIL
- ZeroMQ
- TODO - get rest of this

## The Core: psutil

Third-party library that is easy to use and pretty elegant:

```
>>> import psutil
>>> p = psutil.Process(7384)
>>> p.name
'Address book'

>>> p.uids
user
```

## The Messenger: ZeroMQ

- async library for message passing == smart socket
- highly scalable
- transports: ITC, IPC, TCP, PGM (multicast)
- principal patterns
  - request
  - pub/sub
  - push/pull
- used by IPython
- PyZMG - zmq bind + nice I/O event loop adapted from Tornado

## Circus Architecture

TODO - Get a copy of the image from Tarek.

## Recap

- **circusd**: daemon that watches all processes
- **circusctl**: interaction shell
- **circus-top**: Like top, but only for things Circus is managing
- **circus-httpd**: Runs the web client

TODO - get the rest of the components from Tarek

## Problem

Can't interact with Django workers because they are supervised by Gunicorn, which is managed by Circus.

Answer: Circus sockets - Not just sockets, but also manage processes.

- Every process managed by Circus is forked from circusd
- circusd creates & opens sockets
- child processes can accept() connection on those sockets

## WSGI

- Chaussette: WSGI server that reuses already opened sockets
- Launched with the socket. . .
- TODO - catch up

## Benchmarks

Circus + gevent is slightly faster than Gunicorn + gevent.

## Upcoming features

- Clustering
- stderr/stdout streaming in the web dashboard

## Thanks!

- Docs: <http://circus.io>
- Code: <https://github.com/mozilla-services/circus>
- slides: <http://blog.ziade.org/slides/djangocon2012/circus.html>

## Processing payments for the paranoid

- By Andy McKay
  - @andymckay
  - Works at Mozilla
  - Presented barefoot

The Mozilla Marketplace is the app store for Firefox OS and this Django powered site takes payments from users.

Combined with issues like localisation, identity and scale - we are processing payments through Django. This talk will cover the marketplace, the architecture of the system and how we cope with all the paranoia.

## Who should be paranoid?

Everyone should be paranoid:

- Developers
- Users
- banks
- Everyone

## Mozilla Marketplace

- Powered by Django
- Don't call it an "app store"
- accepts payments
- Powered by open source project 'zamboni'

---

**Note:** Report bugs to Mozilla via their reporting system and they'll pay you.

---

## Steps for purchase

1. Set up your account with Mozilla
2. Purchase and use an app
3. Mozilla bills your carrier

All powered by Solitude: <https://github.com/mozilla/solitude>

## Vulnerabilities they had to consider

### XSS

Over come with \* Mozilla Content Security Policy \* [MDN docs](#) \* Github blog

## Phishing

- navigator.mozPay
- Trusted UI
- [MDN docs](#)

## SQL injection

- careful ORM evaluation

## Ourselves

- Many penetrations happen internally, not from technical assaults from outside.
- Wrote anonymizing code
- Inside the DB:
  - removed personally identifying information
  - encrypted other data
- Defended by depth

## Tips

- use python-requests
  - SSL certs are not handled well by Python's URLLIB
  - Requests does it well
- **wrote django-paranoia to help track security things**
  - Includes something called paranoid\_forms. Logs when people try to add or subtract keys to forms.
  - includes a special sessions component for Django. Logs when:
    - \* user agent changes
    - \* IP Address changes

---

**Note:** Just noticed that @andymckay is presenting barefoot at @djangocon @djangofact #djangocon

---

## The Imaginative Programmer

- by Zed Shaw
  - Wrote Learn Python the Hard Way.
    - \* E-Version: <http://learnpythonthehardway.org>
    - \* [Upcoming Print Version](#)
  - Talks super-fast so I can't keep up.



- True Fact: @zedshaw kicked me in the nuts at @pycon 2009.

In the world of programming nothing is more irritating than the “artist”. Programmers who are all “front-end”, business guys who are all “ideas”, and designers who can’t draw. I say real programmers don’t need to be artists. Rather, a programmer needs to have the skills necessary to take their imagination (or other people’s) and translate it into working software.

This talk will contain many of my secrets for turning ideas I have in my mind into real things. I will lean heavily on the arts, music, and writing, but ultimately this talk will be about implementing software. There will be no fluffy spirituality or hand waving in this talk, just real tricks to help you make stuff.

## Here we go

Talks a quick story about a hipster and an easel in San Francisco. He watched a guy pretend to paint in order to pick up girls.

Zed read tons of design books, but none of them make any sense. Believes that design education is fucking useless. He’s been told that since he’s “logical”, he can’t do art. However, he contends that art is logical (composition, form, etc).

Finds that guitarists are jerks about saying if you aren’t in a band then you aren’t a real musician. However if he says he builds guitars he gets respect. Until he says he’s a programmer.

Rant about writers: Because his best selling book isn’t non-technical, he’s not considered a real writer.

Rant about not submitting pull requests: He brings developers into Python and Django but isn’t considered to have contributed.

---

**Note:** I ranted yesterday about this but maybe Zed has a point.

---

## Common Threads

- Artists says he’s not an artist because he works on developing technical skills.
- Writers say he’s not a writer because he writes tech books.
- Guitarists say he’s not a musician
- Programmers say he’s not a programmer because he doesn’t submit pull requests.

## Proposal

We hack creativity and make it worthless.

## Battle Plan

Learn an “imaginative programming process”

## Four types of People who Don’t Really Exist

- Technique, No Imagination: Stereotypical Programmer

- Imagination, No Technique: Stereotypical Biz Dudes
- ???
- ???

## Zed's Imaginative Programming Process

1. Perceiving the Imaginative Idea.
2. Establish the Concept.
3. Research techniques or tools. Programmers often skip this step.
4. Refine the concept through composition.
5. Explore through prototypes
6. Make it real

## Examples

### Project Zorn

1. Idea: Zed loves mixing colors
2. Concept: Create site for teaching color similar to euler project.
3. Research: Find tools to build this
4. Composition: Make 52 exercises and put it online with interactive parts powered by Django.
5. Prototype: Using Zurb he prototyped the first page: <http://projectzorn.com>
6. Realize: Start writing exercises at Django Circus

Other ideas:

1. Painting in Poland
2. python-lost

## Closing thoughts

- Email yourself your ideas
- Don't worry about the fear aspect. Make sure you don't care if people don't like your stuff.
- Great ideas work best in solitude
- Great implementations work best in teams

## Advanced PostgreSQL in Django

- by Christophe Pettus

PostgreSQL is the most advanced open-source database on the planet, but (except for GeoDjango) few Django developers take advantage of its more advanced features. We want to change that!

We'll talk about custom types (including Admin integration), replication tricks, specialized indexes, unstructured types such as JSON, stored procedures, and other ways to get maximum functionality and performance out of your PostgreSQL data store.

## Database Agnosticism

- Django has to be able to handle all databases.
- That means it loses special features of individual database systems (PGSQL, MySQL, etc).
- Database migration is a once in a lifetime thing.

## What can PostgreSQL do for you

- PostgreSQL has a huge range of built-in types
- Most can be used natively in Django
- You can find libraries to support them.

### citext

- case insensitive text
- ignores case on comparisons for a field

### hstore

- built-in dict-like structure
- Maps to Python dict
- Can be indexed and queried for inclusion
- requires custom sql

### json

- All the cool kids use NoSQL databases
- Something faster than MongoDB to store your JSON databases
- validated going in
- Not feature rich, but it's growing

## Others

- UUID
- IPv4 and IPv6
- Interval - stores time intervals directly in the database
- See Craig Kersteins's talk later

## Where do we sign up?

Getting it work in Django

- Adapt to psycopg2
  - take string representation of the type and convert to and from the python object.
- Write a Field class for Django
  - subclass `django.forms.Field`
- Create a widget

TODO - ask Christophe for the slides so I can finish this section

## Indexes

Django models are great but it's index creation syntax is somewhat lacking. Multi-column indexes for example Partial indexes in Django are good:

```
CREATE INDEX active_orders ON cart_order(status) IF status == 1
```

## Custom Constraints

- Django's database agnosticism is a problem for things like foreign-key handling
- It's also boring

Tips:

- Push database constraints into the database whenever possible.

## Raw SQL

The three-join rule:

- If you are joining more than three tables, use raw SQL
- It turns out that PostgreSQL is really good at multiple joins
- Don't fall back into iterating over querysets to get data. Use SQL!

### Tips:

- Put RAW SQL in the model or manager. Put it in the view and you risk losing it
- Use South
- Don't be constrained by fear of migrating from one database to another. Choose and use special features

### Getting recommendations out of nothing

by Ania Warzecha

---

**Note:** Was extremely late to this talk. Which makes me sad because this was clearly a good talk.

---

### How to combine JavaScript & Django in a smart way

by Przemek Lewandowski

- <http://sunscrapers.com>
- Really nice guy, met him earlier this week.

Have you been using JavaScript more and more when building your web applications? Are you implementing REST API frequently? If so, you have probably realised that server-side generated content is no longer enough to provide cutting edge user experience.

I would like to show you how to avoid jQuery callback hell and how to gain more flexibility using MVC on the client side. I will introduce tools for managing modules in JavaScript and will teach you how to become more productive with CoffeeScript. I will share my experience of integrating Django and sophisticated JavaScript stack from two points of view: RESTful API and static files management. Let the trip begin!

### Basics

- Django
- Javascript via Coffeescript

### Javascript framework considerations

- Backbone didn't do enough
  - Lack of binding mechanism
  - no reusable views
  - Models are poor
- Backbone & Marionette helped but they were still unhappy
- Angular, very nice
- Ember, very nice

## How to use Javascript framework with Coffeescript?

- Use requireJS with extra plugins
  - Coffeescript painless integration
  - modular code
  - builder
  - uglifier

## Tools for building JS apps in Django

- django-compressor
- django-pipeline
- django-require

## Django & APIs for JS apps

- **django-piston** was too long in the tooth
- **django-tastypie** had an uncomfortable amount of boilerplate
- **django-rest-framework** was just right.

django-rest-framrwork example:

```
# serializer
class FriendSerializer(serializers.ModelSerializer):

    class Meta:
        model = Friend
        fields = ('id', 'name', )
```

TODO - add examples from our [book](#), cause we also recommend django-rest-framework.

## Static files management

- django.contrib.staticfiles
- django-storages
- django-cumulus
- django-require

## Closing thoughts

- Never stop trying new solutions

## Thread Profiling in Python

by Amjith Ramamujam

- Works for <http://newrelic.com>
- Slides: <https://speakerdeck.com/amjith/thread-profiling-in-python>

This talk will give a tour of different profiling techniques available for Python applications. We'll cover specific modules in Python for doing function profiling and line level profiling. We'll show the short comings of such mechanisms in production and discuss how to do sampled profiling of specific functions. We'll finish with statistical profilers that use thread stack interrogation.

### What's profiling?

cProfile for profiling the performance of something

Usage:

```
python -m CProfile sample.py
```

For big projects it can be big in response, so use **RunSnakeRun** (wxPython app) which gives you better data.

Google uses profiler and displays the results in their search engine. Which is why it's good to use in production. Unfortunately profiling eats up performance. So what do you do?

### Targeted Profiling

- Profile critical functions
- Do hybrid stuff

### New Relic targets

- Web frameworks
  - Django, flask, etc
  - View layer
  - template layer
  - SQL calls

### Threads for profiling

example:

```
import threading
t = threading.Thread(target=handler)
t.setDaemon(True)
t.start()
time.sleep(0.1)
```

**Pros:**

- Cross platform
- mod\_wsgi compliant

**Cons:**

- Inaccurate for CPI tasks
- can't interrupt C-extensions

### Inquire into what's going on

```
>>> import sys
>>> frames = sys._current_frame()
>>> print(frames)
{1234: <frame>}
>>> import traceback
>>> traceback.extract_stack(frame)
(...stuff here...)
```

### Collate

#### Attempt 1: Default dictionary

- Uniquely identify a function

---

**Note:** Trying to figure out how to document what he's doing here. TODO: Get Amjith's images. :P

---

### Existing Tools

- plop - Uses signals and d3.js
  - Open sourced by Drop box
  - Size of bubble shows how expensive the thread is for the system

### Summary

- No green threads is low overhead
- CPython & PyPy have high overhead

### Grand Finale

Deterministic profiler + statistical profiler is how they assemble the data. Newrelic merged the profilers so the data is much better.



## Migrating The Future

by Andrew Godwin

- Works at <http://lanyrd.com>
- Django core developer

It's been almost five years since South was first released, and in that time many things have changed - now it's time for a new migration system, built into Django itself and with new features and a solid foundation based on those five years of learning. Hear the problems in both running this kind of open source project as well as those of dealing with five different database backends, and how both you and South can learn from them.

## Kickstarter

- Wanted £2,500 pounds
- Got £17,952 pounds from 502 backers

## What's wrong with South

- 5 years of learning, lessons learned
- Poor VCS branching, two people commit to same place
- Huge migration file size is too big
- Migration sets get too large

## New modules

- `django.db.migrations`
  - Migration commands
  - autodetection
  - The public API, as it were
- `django.db.backends.schema`
  - SQL generation
  - Database abstraction

Databases supported

- PostgreSQL - yes
- MySQL - yes
- SQLite - yes
- Oracle - hopefully
- MSSQL - hopefully
- DB2 - maybe
- MongoDB - maybe

## New migration format

---

**Note:** TODO get this later, the code samples are on a black background.

---

- Shrink the size of migration

## Dependency Management

If you and another developer both add a new migration with the same name, South sorts in ASCII sort order. Which is a serious problem if you miss a dependency

- South dependencies are driven now by a specification value in the Migration module
- Auto-Merges migrations when there is no conflicting migrations
- Can **squash** all the migrations into one big migration!

## How is it going?

Working 2-3 days a week on this full-time

Working on it:

- **Schema backends:** Mostly done, ready for merge
- **Migration code:** Still going, most complex part

Upcoming

- **Field API changes:** This Field needs to be able to inform migrations what's going on

## Resources

Code: <https://github.com/andrewgodwin/django/tree/schema-alteration> Blog: <http://aerocode.org/category/django-diaries> email: [andrew@aerocode.org](mailto:andrew@aerocode.org)

Working to do this for all of us, so give him feedback!

## Having Your Pony and Committing It Too

by Jacob Burch

- Works for @revsys
- Contributed to `django.core.cache`
- Not in AUTHORS file of Django (yet)

For many years before 2012, the topic at the tip of every argument-seeking tongue at Django Conferences was “when is Django going to get on Github?” Getting the core framework on the social coding site was the first stride in breaking down the barriers to having anyone and everyone not only having a pony, but getting it into core too. Now that this important step is almost a year in, just how easy is it to take the step from end-user to core-contributor? Delightfully Easy.

So easy, that I'll be breaking every rule I know in giving a talk and actually attempting to get a feature from idea, to code, to request, to a live haggle-and-debate session with core contributors in-audience, to pull request to (hopefully!) merge all within 30 minutes. Advice from a variety previous contributors on will be provided throughout the demonstration, including tips for getting very small bugs fixed quickly to strategies for getting necessary groundswell for larger full-feature ideas.

### Not covered

- You need to know virtualenv/git
- Large overview of Django's core code
- Advice in what to get involved in

### Thoughts

- It's scary to start contributing to Django
- It seems labrythine, and it is.
- It can take a while to get core code in

Max Weber describes politics as “the slow boring of hard boards”. Open source is much the same. – Russell Keith-Magee

### Confident vs Bold

- Follow the wikipedia motto: “Be Bold” – Alex Gaynor
- You are not your code – Marty Alchin

### Before you do anything with Django

- Fork Django
- `git clone your repo`
- `./runtestspy --settings=test_sqlite`
- Do not pass GO until tests run\*

### Forms of Contribute

- Bug Fixes are a great place to start
- Minor Contributions
- Major Contributions

## Bug Fixes

1. Write a test
2. Have it break
3. Fix the code until tests pass
4. Test against regressions
5. Fix is not necessarily free from discussion

## Major Contributions

Do your homework

- Search trac
- Search django-developers
- Become familiar with the code you're proposing.

## Minor Additions

- Follow same steps as major contributions

## Risking against everything we are supposed to do

- Coding live
- Submitting to Django live

## Russ is talking

Ticket [#9595](#) in Django

---

**Note:** Video of Jacob Talking while he starts coding began here.

---

## How to make a proposal

- Don't communicate entitlement
- Don't focus only on your own needs
- Be decent
- State the problem clearly
- **Confidence:** propose a clear solution
- Show your homework
  - Previous tickets/attempts
  - Potential downsides/drawbacks

- **Humility**: Unsure of aspect? Ask!

## Code

- Make the code work
- Document your work
- Make the code follow standards
  - Stay within pep8 mostly
  - respect existing style
  - linters are your friend
  - comments are must
  - get a peer code review before submitting
- Document and **boldly** defend design decisions (wiki)

---

**Note:** Video of Jacob Talking while he starts coding ended here.

---

## Review

- Your ego is **not** on the line.
- **Humility**: No, really.

## Lack of Review

- **Patience**: Core members are people
- **Pro-active**: Send polite, friendly follow up messages often

## Review: Part 1 of n

- **Confidence**: Do not give up or get angry if changes must be mad.
- Follow up quickly
- Email tag can be frustrating
- #django-dev can help

---

**Note:** Russell reviewed the code here

---

## Fractal Architectures

by Laurens Van Houtven

An alternative take on Django's traditional layered web service architecture.

---

**Note:** Was very late.

---

### Concept

Use many tiny servers with tiny Twisted powered web servers with tiny instances of SQLite3 as the backend. Each user gets their own mini-server!

### Backend

#### Twisted

Cause he's a core dev and is asynchronous. We could use Django.

#### SQLite3

- It's in the Python stdlib
- Wrapped by Axiom, a Python library, documented at <http://lvh.com/axiombook>
- Using SQLite3 means it's the same development environment as production - because how for developers PostgreSQL is **NOT** set up the same locally as production.

### Static Assets / Long term storage

- Uses a CDN like AWS, OpenStack Swift, or something else.
- Good for handling of micro-instance failure

### Two Ways for Better Performance

- Do less stuff
- Make stuff run faster

If you off-put stuff from one server onto a database server, cache server, et al, then even in-database center latency will become an issue. His approach is to put everything on one tiny server per user and reduce latency between machines to **nothing**.

This is the concept of **Data Locality**.

## Devil's Advocacy

- Unusual design might make it hard to get more developer help.
- Weird separation.
- Data is weird, maybe not good for big data. But 99% of sites don't have actual big data.
- Transaction support doesn't work.

## Thoughts

- Crazy, fun idea
- Sometimes it's good to try the insanity and see what happens

## Getting past the Django ORM limitations with Postgres

by Craig Kersteins

- Heroku guy
- <http://craigkerstiens.com>
- <https://twitter.com/craigkerstiens>

With most frameworks the ORM attempts to treat all databases equally, this results in developers being limited in how many advantages they can take of their database. In particular Postgres has many features which developers would love to take advantage of but are not easily accessible via the Django ORM.

---

**Note:** I'm going to mention the <https://www.djangopackages.com/grids/g/postgresql-integration/> grid.

---

## Public Service Announcements

- Postgres.app

## Why PostgreSQL?

- It's the emacs of databases
  - Platform to build things on
  - Many built-in extensions
- Datatypes
- Conditional index

## Limitations with Django

- The ORM works with too many different databases
- Lowest common denominator

- Django ORM has few datatypes
- Indexes are limited compared to pure PostgreSQL

But Django isn't too bad

## What PostgreSQL does that's cool

Datatypes:

- Arrays datatype
- hstore
  - does what MongoDB does but inside of PostgreSQL!
  - Stores in JSON
  - Getting better in PostgreSQL 9.4

## Queueing

Normally doing this in a database is a bad idea. So we use Redis and other resources. PostgreSQL has pub/sub and makes a great queue. You can get it working via celery with:

```
pip install celery trunk
```

## Text Search

Instead of Lucene, Sphinx, or Solr you can use PostgreSQL for full text search.

---

**Note:** Is there a Django extension?

---

## Indexes

You should generally use a BTREE index. Depending on your use case, you may need other indexes.

## Flip to Read Only Mode

If you need to do system changes, you can make your site output only by turning on PostgreSQL's read-only mode. How cool is that?

## Connections for Django

- Django right now doesn't have persistent DB connections (not until Django 1.6 anyway)
- It has to reconnect all the time to the database, which is a performance hit.



## PostgreSQL resources

- My favorite is this [PostgreSQL book](#).

## The Web of Stuff

By Zachary Voase

- Cook
- Eat
- lift
- makes stuff
  - Was making stuff for Ford for big presentations
  - Interactive trade show experiences

As software developers, the world of hardware can seem confusing but alluring. Small computers are now cheap enough that useful products can be built for less than \$100. But the real value from the Internet of Things comes from networking. In this talk, I'll introduce you to basic hardware hacking, and show you how Web, mobile and microcontroller technologies can be brought together using Django—with surprising and playful results.

## Act I - Little Data

- Computers are getting big again because of server farms and huge number crunching efforts
- Economy of size goes in both directions. Not just big data, but little Data

As backend developers it's easy to forget that often with all the servers and code and automation we forget that at some point deep in the chain there is a human being interaction with the system. Example: Cell Phone

. epigraph:

```
The maker movement is an effort by computers to liberate themselves from their human_
↳ overlords. -- Zachary Voase
```

---

**Note:** Zach tempted the demo gods by demonstrating something that used Arduino, Github, Django, Heroku, Foursquare to do something. Alas, the demo gods were not kind. The internet was very flaky. Still, his device was really cool.

---

## Act II - Personal Development

- Tutorials are useful for beginners
- Advanced experts might need references
- In-between beginner and advanced it's hard to get good.

---

**Note:** This in-between spot was the original target of [Two Scoops of Django](#).

---

## Resources

- <https://github.com/zacharyvoase/swipecheck>
- “Getting started with Arduino”
- “The Art of Electronics”

## Bleed for Speed: Django for Rapid Prototyping

By Rob Spectre

- <http://twilio.com> director of evangelism
- <https://twitter.com/@dN0t>

Come one, come all to the DjangoCon sideshow to see feats of inhuman speed as we take Django for a spin with rapid prototyping. Tossing security, performance, and maintainability out the tent, Rob Spectre shows a 30 minutes of tips and tricks for building rapid prototypes on Django gained over dozens of hackathons. Find the fastest path from startproject to a publicly accessible endpoint. Discover the reusable apps that cut down your hack’s implementation time in half. Marvel at the testing techniques that will minimize demo-killing broken code. Step right up ladies and gents and see the framework forged in the newsroom furnace blast your entirely temporary project across hackathon deadline.

## History Lesson

In the American civil war, there were naval battles. They had boats, and they had the Battle of Mobile Bay. Rear Admiral Buegard led a fleet in with a simple battle of going between the fort on the left and the mines on the right to bash at the enemy.

The problem was that during the beginning one of the ships sailed right into the minefield and sunk. The admiral was crazy so would climb to the top of the mast and shout orders to the battle.

In his insanity he commanded the fleet to drive through the mines/torpedoes. And he said, “Damn the torpedoes drive straight ahead”. He knew that the mines/torpedoes were old and could be driven through.

### What does this have to do with Django?

Sometimes when faced with a daunting task, you have to take the bit between your teeth and plow straight ahead.

## 24 hour prototyping

- Do it outside of hackathons
- Great for tech and concept discovery
- Throw away that code!

## Why Django?

Rob’s claim: Django is the best for rapid prototyping development.

- Django was built explicitly for rapid prototyping development. Out of the newsroom and into production. Today.
- Django is flexible.
- Django has an incredible community. Have a problem?

- Jump on IRC!
- Read the docs!
- Stack Overflow!
- Read [Two Scoops of Django](#)!
- Bottom line: Gets more stuff done!

## How to speed up Django development

### Initial setup

---

**Note:** Disclaimer: Rob really promoted our [book](#). We had no idea before the conference he was going to do this.

---

- Read [Two Scoops of Django](#) chapter 2 & 3. Much of it is summarized in <https://github.com/twoscoops/django-twoscoops-project>
  - Suggests add a profile, which we have as a pull request we may review during the sprints.
- You can use Mozilla's Play-Doh as well

### Static Files

- Use the defaults as much as possible
- Nice API
- Bad: Documentation needs revision.
- Brunch (<http://brunch.io/>) is nice for compiling everything

### Deployment

- Heroku is nice, and works in Europe now!
- You can also use AWS: Learn configuration management (chef/puppet/salt stack etc).
- Salt Stack call-out: <https://github.com/esacteksab/salt-states>

### Packages to use

#### Build the RESTFUL API

Do it RIGHT away so the front end person can work

- Rob prefers django-tastypie
- These days prefer django-rest-framework now.

## Social Auth

- django-social-auth
  - Gajillions of auth services it supports
  - Takes some work to set up
- django-allauth
  - Fast setup
  - Doesn't have as many auth services it supports
- South
- celery
  - Unfair advantage for Python/Django in competitions
  - Setup is a pain
  - **Secret:** You can use the database for hackathons and not worry about setting up a real queue engine

## Testing

- Why would you test in a short time space?
- **On the contrary:** Why wouldn't you test?
- Knowing your views all return 200 tests means you don't make the same results in writing code or demoing the project

More notes:

- Django tests are fast/easy to write
- AngularJS has a great testing tool. Even if you don't use that many of it's features, check out testacular.

## Conclusion

When faced with a difficult problem, sometimes its good to plow straight ahead.

## Growing Open Source Seeds

By Kenneth Reitz

- Heroku guy
- <https://twitter.com/kennethreitz>
- <https://github.com/kennethreitz>
- Creator of python-requests

This talk will be an in-depth review of the stages that most open source projects go though, and the decisions their maintainers face. Requests will be used as an example — lessons learned and best practices will be covered.

### Once upon a time...

The Facebook SDK Python library

- Facebook rarely updated it
- Became unworkable
- People complained, got on Hacker News
- Disabled comments

Now replaced by <http://pythonforfacebook.com>

### Public Source Projects

- Company open sources code
- Doesn't maintain it: motivations are unclear
- Really sucks for users of the code

### On the other hand.. Gittip!

- Platform for sustainable development
  - Everything is open source, including internal discussions, interviews with media, etc
  - Everything is an issue
  - Major decisions are voted on github.
  - Interviewed with journalists are live-streamed
- "I'm not building Gittip, I'm building the community that's building Gittip." – Chad Whitacre

### Shared Investments

- Shared ownership, extreme transparency
- New contributors get involved by following a documented process
- Low risk. High bus-factor
- See also: Python, Django, Firefox, jQuery...

### HTTP for Humans

python-requests

- One of the most installed PyPI projects
- Key difference between gittip/django and requests: Kenneth makes all the decisions

## Dictatorship Projects

- Totalitarian BDFL owns everything
- Dictator makes all decisions
- Community feedback is encouraged, but users with feedback should have no expectation of change.

## Lessons Learned

- Be Cordial be on your way
  - Contributors
    - \* Keep all interactions with a maintainer as respectful as possible
    - \* They have likely donated a significant amount of time and energy into their project
  - Maintainers
    - \* be immensely thankful to all contributors
    - \* They are the lifeblood of your project
    - \* Ignore non-constructive feedback
    - \* Some people just take things too seriously

## Avoiding Burnout

## Sustainability

- One of the biggest challenges for open source
- Everyone has a limited amount of time in the day

## Learn to do less

- When an issue or pull request comes into the repo, two other developers usually triage it.
- This saves an immense amount of time
- I can focus my time on larger issues.

## Learn to say no

- Saying ‘No’ is really important
- Learn to do it nicely

Simple Code is Good. Complex code is bad.

“Open source makes the world a better place. Please, don’t make it complicated.” – Kenneth Reitz

## Mock your database

by Marc Tamlyn

- <https://twitter.com/mjmtamlyn>
- Pivotal figure in the giant Django CBV documentation refactor for 1.5

Databases are slow. Well, if the goal is 1 millisecond per test they are anyway. We want to avoid interacting with the database as much as possible when testing, especially if the tests aren't anything to do with the queries.

This talk will look at various ways of avoiding those pesky database queries and making tests faster!

## Your database is slow

- When you are testing, hitting the database is slow. Connections, writing to disk, getting down to SQL, etc
- Why do you care?
- We want SPEED!
- The faster your test goes, the better.

## Example

```
def test_naive(self):
    label = RecordLabel.objects.create()
    artist = Artist.objects.create()
    track = Track.objects.create()
    # etc
```

- 8 database queries
- 5.2 seconds for 1000 runs on PostgreSQL
- 3.2 seconds for 1000 runs on SQLite3 running in memory

Doesn't seem slow, but what if we are testing 40 models this way? Test factories make this worse!

## Class Based Views: Untangling the Mess

by Dr. Russell Keith-Magee

- Django Core Developer since January 2006
- DSF President since June 2010
- CTO and co-founder of TradesCloud

## Why CBVs?

Introduced in Django 1.3 in 2011. What's the history?

## History per 2005

- Django is for building websites
- Views are for displaying content
- There are lots of refactorable things to do
- Generic views could handle all of this:
  - Display template
  - Display object or list of objects
  - Handle forms
  - Every view is a function
  - Configuration via arguments

## Problems with function based generic views

- Configuration options limited by urls.py args
- No control over logic flow
- No re-use between views

<p><b>Warning:</b> Another thing against function-based generic views was that people can and do implement their own broken CBV system. Leaky states is a serious issue. Don't roll your own unless you really understand</p>
---

## Django 1.3

- They kept trying to get CBVs into Django starting with Django 1.0.

## What went wrong

- Fundamental confusion over purpose
- Confusion over implementation choices
- Ravioli code
  - Luke Plant described the effort as bad code.
  - “You don't know what's in the ravioli.”
  - Steep learning curve
- Bad documentation
  - Russell takes the blame for the problems.
  - Myself, Marc Tamlyn, and others worked to make it work.



## Purpose

Class-Based views are an object-oriented analog for function based views.

- Class based views
- Class based Generic Views

Because we are subclassing a base class, we get tons of extra options.

- automatic OPTIONS request handling
- automatic naive HEAD request handling
- automatic HTTP 405 on unknown verbs

## CB Generic Views

- Uses Class Based Views as a base
- Creates analogs of the old generic views
- Addresses limits of functional approach

## Implementation Choices

- See details of the debate at <https://code.djangoproject.com/wiki/ClassBasedViews>
- A class that is instantiated as a view
- Problems:
  - What gets instantiated?
  - How does it gets instantiated?
  - Once per process or request?
  - What's the lifespan?
  - What about state? (race conditions!!!)
  - How does it work with `urls.py`?
  - How do you configure things?
- Django's admin system is a CBV
  - Implemented using simple `__call__()`
  - Doesn't have HTTP Verb support
  - Suffers from serious state issues

<b>Warning:</b> Don't put <code>self</code> on Django admin objects or you will cause state issues.
---

Other concept design:

- Change the `urls.py` contract
  - Current: a callable

- Change to: A callable or a class
- The problem is that this would have forced them to change a lot of source code and make things under the hood much more complex.
- Decision: keep the `urls.py` contract clear

## Ravioli

- Goal: Replace FBV generics with CBV generics
- Make it easy to extend
- Unfortunately complex class hierarchy
- However...
  - Allows for maximum reuse of core logic
  - Extremely flexible for inserting new logic
  - Easy to add your own mixins

## Documentation

- Bad as originally designed
- Much better now
- Still need framework decisions needed

---

**Note:** Use django-braces to fill out the missing pieces of CBVs.

---

## Where to from here?

- Add new features?
- Did they solve the wrong problem? Modern problems:
  - Multiple forms/formsets per page
  - Conditional forms
  - Continuous scrolling, not pagination
  - AHAX support
  - PJAX
  - Multiple “actions” per page

## Call to Action

- IN discussion: Do you mean CBV or CBGV?
- Docs can still be improved. YESSS!!!!
- #18830 - FormCollection

- Experiment with APIs. Django's admin is a useful case study

## Documentation Ideas for CBVs during the sprints

---

**Note:** I'm not working on Django CBV documentation during the sprints. However, I'm open to suggesting paths to take:

---

- Tutorials in the CBVs section of the core Django documents.
- More working code examples
- Flow charts!

## Dynamic Models

by Juergen Schackmann

Django has been built on the assumption to have upfront static data models; i.e. the developer implements them completely before deployment. However, there are numerous real-world uses cases that require to have dynamic models that can be created and amended by users or some kind of user actions. Examples could be: customizable products for a shop, unique content types to represent web site content in a CMS or online surveys that are created on the fly.

There are various conceptual options to solve this problem. The most prominent ones are: a) Entity Attribute Value Models, b) Pickled Fields and Pickled Models, c) Database DDL operations at run-time. Most of which have been discussed intensively and the Django community has developed numerous apps. I will compare the various approaches and apps in terms of usability, speed, features, sweet spots and preferred use cases. This will support any future evaluation for a specific project. But it will hopefully also trigger a fruitful community discussion on the importance of this feature for Django in comparison with other applications and frameworks like Zope/Plone, Magento etc.

## How to do it

- Entity attribute values
  - Table columns become rows in another table's rows
  - Performance issues
- Serialized Dictionary Django apps
  - Dangerous because of use of Pickle
  - Problematic because of lack of searchability (this is mitigated via tools like PostgreSQL hstore or MongoDB)
- Runtime schema updates
  - Allow updates of the database schema by non-technical user action
    - \* This sounds kind of risky
  - Dynamic models problematic since it can interfere of how the database is designed.
  - Complexity is another issue. How do you keep the database from going nuts from user action?
  - Database integrity is a major issue.
  - Column updates is a really nasty issue. For example, the database has to lock the for minutes or hours.

## Prehistorical Python: Patterns past their prime

by Lennart Regebro

- Freelancer
- Django, Pyramid, and Zope guy
- One of the tech reviewers of [Two Scoops of Django!](#)
- Author of [Porting Python Python 3](#)
  - worth it if you are upgrading from Python 2 to 3!

There are many idioms and patterns that used to be a best practice but isn't anymore, thanks to changes in Python. Despite that they often show up even in new code, and some of these patterns are even explained to be Good Ideas at [stackoverflow](#) and similar.

This talk will bring out the most common of these patterns so you know what they are, and why you should avoid them.

### Defaultdict

```
# python 2.5
from collections import defaultdict
data = defaultdict()
data[key] = value

# Python 2.5-
# Exists still in Django 1.5.1
# django/db/models/sql/query.py
if key in data:
    data[key].add(value)
else:
    data[key] = set([value])
```

### Sets

- Unique values
- Unordered
- Fast lookup
- Python built-in in 2.4

### Sets before sets

```
d = {}
for each in list_of_things:
    d[each] = None

list_of_things = d.keys()
```

## Sorting

```
# new way - missed the old ways
retval = set()
for tn in template_nmes:
    retval.update(search_python(python_code, tn))
retval = sorted(retval)
```

## Conditional Expressions

```
# old way
# django 1.5.1 django/db/models/related.py
first_choice = include_blank and blank_choice = []

# new way
first_choice = blank_choice if include_blank else []
```

## Constants and Loops

```
# outside vs inside
# PyPy is 33x slower on this one!
each * 5 ** a_var
```

---

**Note:** Thought: Evaluate your constants outside the loop

---

## String Concatenation

```
# The 'fastest' way
self._leftover = b''.join([bytes, self._leftover])
```

adding is faster than using the .join() method used above. WTF?!?

### Explaining the ‘WTF?!?’

- Looping over a list of strings and adding them together is slow.
- Using .join with a list of strings is fast.
- If you add just two strings, adding them is faster.

### 1.2.3 PyWaW

PyWaW is short for the Python Warsaw Meetup Group. Which met a couple days before.

## Django 1.6 and Beyond

By Russell Keith-Magee

- Django core team developer
- President of the Django Software Foundation
- @freakboy3742
- PhD in ...

This is Russell's vision for what is happening in Django, but nothing is concrete because Django is a volunteer project.

## What's missing?

Good frameworks don't come from academia, they come from projects solving real problems.

—Jacob Kaplan-Moss

## Things likely to happen

- App Refactor
  - Application name is fixed. For example, 'coupons' in admin will retain that name.
  - What goes into an app?
  - Probably not in 1.6, maybe in 1.7.
- Schema Migration
  - South ought to be in core.
  - Andrew Godwin is working on it.
  - The plumbing is the backend, the porcelain is how users interact with it.
- Composite Primary Keys
  - Easy concept to explain, hard to implement with all the existing pieces.
- Increased decoupling
  - Pieces of Django core are getting moved out.
  - Local flavor is getting moved out.
- Admin 2.0
  - A lot of things it could do but it doesn't.
  - Many third-party skins
  - Current version not using CBVs but they could be.
- Release Schedule
  - Averaged a release every 11 months
  - **OMG** this means we have to update Two Scoops of Django faster. :P
- Singleton Cleanup
  - The settings a'la `django.conf.settings`.

- It’s a problem that really needs to be fixed.
- Considering breaking backwards compatibility.

### Long Term Predictions (low accuracy)

- Better sharing with the rest of the Python world.
  - WSGI
  - SQLAlchemy
  - NoSQL
    - \* Probably not happening because it would only allow for a subset of the Django ORM functionality
    - \* What about the ORM?
- Extinction-level events (Django is a great framework for 2005, but it’s 2013)
  - Django doesn’t handle real-time.
  - Server/client separation
    - \* Javascript frameworks are not chosen yet.
    - \* Sourcemaps are making the debugging of compiled Javascript framework
  - Mobile
    - \* Objective-C
    - \* Java
    - \* HTML5
- How do we make great ideas happen?
  - Decisions are made to those who show up.

## 1.3 Pycon 2013

---

**Note:** We arrived a few days late so my notes don’t start until Saturday morning.

---

### 1.3.1 Dynamic Code Patterns: Extending Your Applications with Plugins

Python makes loading code dynamically easy, allowing you to configure and extend your application by discovering and loading extensions at runtime. This presentation will discuss the techniques for dynamic code loading used in several well-known applications and weigh the pros and cons of each approach.

by Doug Hellmann

The applications studied include:

- Mercurial
- Sphinx
- Trac

- virtualenvwrapper
- Django
- nose
- ceilometer
- OpenStack CLI
- cliff

## Cliff

cliff is a framework for building command line programs and is where Doug did his research for his talk.

- <https://github.com/dreamhost/cliff>
- <https://pypi.python.org/pypi/cliff>
- <https://cliff.readthedocs.org/en/latest/>

see also <https://github.com/dreamhost/stevedore>

## How plugins work?

### Discovery

- File/Explicit: Mercurial
- File/Scan: Diamond/Blogofile
- Import reference / Explicit: Django, Mercurial, Pyramid, Spginx, Nova
- Import reference / Scan: 1 or 2

### Enabling

- Explicit: Django, Pyramid, SQLAlchemy, Blogofile, Mercurial, Trac, Sphinx
- Implicit: virtualenvwrapper, cliff

### Importing

- Custom: Django, Pyramid, Sphinx, Diamond, Nova, Nose, SQLAlchemy, Blogofile
- pkg\_resources: Trac, Nose, SQLAlchemy, Blogofile

### API Enforcement

- Convention is easier but Base Class / Interface is more stable
- Doug used Abstract Base Classes for cliff



## What Doug did for cliff

### Discovery / Importing

- Entry points
- Be consistent

### Enabling

- Explicit disabling
- Automatic disabling

Summary: Everything is turned on by default.

### Integration

- Fine
- Inspect
- Application owns relationship

### API enforcement

- Abstract base Classes
- Duck Typing

### Invocation

- Storage - Driver
- Notifications - Dispatcher

## 1.3.2 Porting Django apps to Python 3

Django 1.5 now supports Python 3, so now's the time to start thinking about porting your apps and sites. Come see how! I'll talk about the porting techniques that work, and present two case studies: porting a site, and porting a reusable app.

by Jacob Kaplan-Moss

- Django co-creator and BDFL
- <https://github.com/jacobian> / <https://twitter.com/jacobian>

## Do I want to use Python 3?

Python 3 has fewer warts

- `urllib` / `urllib2` replaced with `urllib.parse`
- std library cleanup
- funky syntax is killed
- `print()` is a function!
- `super()` syntax is better!
- unicode no longer sucks!

## Can I use Python 3?

A solid maybe. Missing pieces as of 3/16/2013:

- No Python Image Library (PIL / Pillow)
- No MySQL python 3 bindings aren't that good.
- Popular items on <https://www.djangopackages.com>:
  - No gunicorn as async (sync does work)
  - No django-debug-toolbar
  - No django-registration
  - No django-extensions
  - No Haystack
  - No django-tagging
  - No Sentry
  - No django-compressor
- Much easier for new projects over existing sites

## Options

1. Python 3 only
  - Brand new project
  - Fewer dependencies
2. Translated source (2to3)
3. Single codebase

## 3 only

Good for new Django sites.

## 2to3

First tool released for maintaining code, lets you translate from Python 2 to Python 3.

It's amazing but not that practical: if you release the Python 3 version of code generated by 2to3 and then someone sends you a Python 3 patch, you have to port the patch to 2, apply it, and run 2to3.

## Single (shared) source

Keep a single codebase that runs on both 2 and 3. Good for apps that need to support both.

## How to port to Python 3

- Choose an approach from the above.
- Get the test suite running (use django-discover-runner, tox):

```
[tox]
envlist = py27-django14, py33-django15

py27-django14
```

- Evaluate dependencies
- syntax changes
  - print vs print()
  - django.utils.six
- Fix unicode handling:

```
django.utils unicode
```

See Jeff Triplett's port of django-sitetree

## Documentation

- [django.me/py3](http://django.me/py3)

## Moving Forward

- Django used to be the holdup for moving Python 3 usage forward
- Q&A is at room 201 upstairs

### 1.3.3 So Easy You Can Even Do It in JavaScript: Event-Driven Architecture for Regular Programmers

In this era of rich browser applications, everybody needs to know at least enough about events to write an 'onclick' handler. But events have a reputation for being confusing. In this talk I'll explain why events can be quite easy to understand if you think about them the right way, and how to scale your understanding from trivial browser JavaScript to distributed systems in Python.

by Glyph

- Founder of Twisted
- has been doing event-driven architecture since he was 8 years old
- Not stupid enough to attempt live-coding while on stage.

---

**Note:** Code samples too dense to jot down during live-noting.

---

### Things that are hard in Javascript

- Comparing Arrays
- Adding numbers
- Defining types
- Loading Modules

### Things that are easy in Javascript

- Calling functions
- Handling events

### A Tale of Two Events

- asynchronous I/O
- Clicky buttons

### When X -> Do Y

- Event driven architecture is incredibly simple. When X, then do Y.
- When I click -> Do Say “hi”

### HTML Event-Driven Example

```
<!DOCTYPE html>
<button onclick="alert(this.innerHTML); "
    Hello, world!
</button>
```

### PyJS (Most complete Python in browser)

- <http://pyjs.org>
- Javascript compiler for Python
- Widget toolkit

```
# TODO Add imports

def hello_world(button):
    alert(button.getHTML())
b = Button("Hello, world")
b.addClickListener(hello_world)

# TODO Finish
```

```
# TODO Add imports

def alert(txt):
    lbl = Text()
    btn = Button("OK")

    vert = V.VerticalPanel()
    # TODO Finish
```

### 1.3.4 Lightning Talks

**Note:** Live-noting lightning talks is very challenging. I'll do what I can but the level of detail provided in lightning talk notes will in general not be the same as my notes on normal talks.

**Warning:** If you are presenting, never, ever, ever, ever rely on the internet.

#### Retask: Queue for humans (Kushal Das)

- Simplest setup
- Ease of use
- redis backend
- <https://pypi.python.org/pypi/retask>
- [retask.rtfid.org](http://retask.rtfid.org)

```
# producer.py
from retask.task import Task
from retask.queue import Queue
queue = Queue('example')
info1 = {'user': 'kushal', 'url': 'http://kushaldas.in'}
info2 = {'user': 'fedora planet', 'url': 'http://planet.fedoraproject.org'}
task1 = Task(info1)
task2 = Task(info2)
queue.connect()
queue.enqueue(task1)
queue.enqueue(task2)
```

```
# consumer
from retask.task import Task
from retask.queue import Queue
```

```
queue = Queue('example')
queue.connect()
while queue.length != 0:
    task = queue.dequeue()
    if task:
        print task.data
```

## How and why a Java Expert switched to Python (Ron Cox)

- Got into Java v1 ages ago
- Worked with servlets to deliver web sites
- About 2.5 years ago was working on mobile tech including Android and iOS work.
- Was tired of Java:
  - Java language wasn't productive enough.
  - Java platform was very resource intensive
- New stack:

```
Python 3.2
CherryPy
MongoDB
Mailgun
AWS
```

- Steve Holdren's comment: <http://www.artima.com/weblogs/viewpost.jsp?thread=42242>

## Coding Across America (Matt Makai)

Coding Across America is a five month journey around the United States to learn and write about technology in thirty cities.

- 30 cities in 5 months
- Talk with developers from all cities
- Especially Python developers
- <http://codingacrossamerica.com>

## Gitstreams (Justin Lily)

- Doesn't like the Github activity stream
  - Too much activity
  - Filtering isn't good enough
- gitstreams is an email digest of GitHub activity
- You choose the email frequency

### NasberryPi (Mark Ransom)

Home media server!

- Just started with RaspberryPi
- Got this working on a Pogo plug, should work fine with RaspberryPi
- What he has setup:
  - Fileserver
  - Media server
  - Web server (nginx, django) (For a personal home site, why does he use Nginx?)
  - Torrent server
  - More
- Setup is easy, just `sudo apt-get` 7 packages

### European Conferences (Mike Mueller)

#### Euro SciPy

- August 21-24 in Brussels, Belgium
- 2 days of tutorials, 2 days of conference
- <http://euroscipy.org>

#### PyCon Germania

- October 14-19
- German speaking PyCon
- <http://de.pycon.org>

### PyWeek Challenge (Richard Jones)

- Spend a week writing a video game using Python
- Learn more, create libraries, maybe even release something on Steam!

### Python Epiphanies (Stuart Williams)

- How do you pretend to type during talks so you don't make mistakes?
- Fake it until you make it!
- Use the code module from the Python stdlib

## Job Security (Chris Neugebaur)

- PyCon Australia 2013
- People code in Python because it's readable
  - “Readability counts”
  - PEP-8
- Readability sucks
  1. People can comprehend your code
  2. You can maintain your code
  3. Your code is applicable in more places
- THIS IS ALL BAD! (if you want more billable hours)
- How do you write unmaintainable code?
  - Variable naming systems
  - Metaclasses
  - Monkey-patching (roll your own stdlib)

### 1.3.5 Keynotes

#### What's makes Python awesome? (Raymond Hettiger)

- Lives in San Jose
- <http://twitter.com/raymondh>
- Core contributor of PyCon since forever
  - set(), frozenset(), sorted(), reversed(), enumerate(), any(), all()
  - collections, itertools, etc
  - etc
- I've seen previous versions of this talk. My notes at those times:
  - [http://pydanny-event-notes.readthedocs.org/en/latest/PyCodeConf2011/python\\_is\\_awesome.html](http://pydanny-event-notes.readthedocs.org/en/latest/PyCodeConf2011/python_is_awesome.html)

#### Specifics

- License
- Commercial distros (ActiveState/Enthought)
- Zen of Python
- Community
- Repository of Modules (PyPI)
- Killer apps (Django, Pandas, etc)
- Win32
- books



- *Shameless plug*: I wrote a book called Two Scoops of Django. Check it out at <http://django.2scoops.org>

## High level qualities of Python

- Ease of learning
- Rapid development cycle
- Economy of expression
- Readability and Beauty
- One way to do it
- Interactive prompt
- Batteries includes
- Protocols: WSGI, dbapi, etc

```
# search directory tree for all duplicate files
import hashlib, os, pprint

hashmap = {} # content signature -> list of filenames
for path, dirs, files in os.walk('.'):
    for filename in files:
        fullname = os.path.join(path, filename)
        with open(fullname) as f:
            d = f.read()
            h = hashlib.md5(d).hexdigest()
            filelist = hashmap.setdefault(h, [])
            filelist.append(fullname)
pprint.pprint(hashmap)
```

## Indentation

- This is how we write pseudo-code in or out of Python
- Contributes to the uncluttered feel of the language

## List comprehensions

- arguably the most loved feature of the language
- How much stuff should we put on one line?
  - Each list comprehension should represent a single English sentence

## Generators

- Easiest way to write an iterator
- Simple syntax, only adds the YIELD keyword

## Generator Expressions

- Same syntax as list comprehensions but with parenthesis instead of brackets
- Acts as a generator
- reduces memory footprints exponentially.

---

**Note:** Giant embarrassing oops by pydanny: At PyCon Philippines 2012 I demonstrated a Gajillionitem element generator expression in my shell, but used brackets instead of parenthesis.

---

## Decorators

- Expressive
- Easy on the eyes
- Works for functions, methods, and classes
- Adds powerful layer of composable tools

## Abstract Base Classes

Uniform definition of what it means to be a sequence, mapping, etc

Ability to override is *isinstance()* and *issubclass()*

- The new duck-typing “If it says it’s a duck...”

Mix-in capability

## Superstars

Unbelievably good people coming into things

### Guido Van Rossum

Forthcoming

### Van Lindburgh

Forthcoming

## 1.4 LA Migra Hack

### 1.4.1 About me

I am Daniel Greenfeld ([blog](#), [twitter](#)), one of the principals at [Cartwheel Web](#) and CTO of [MoveHero](#).

## 1.4.2 Talks

### Data Mining with Spreadsheets

by Ronald Campbell of the Journalist at the Orange County Register

#### Useful links

- <http://census.gov/cps/data/cpstablecreator.html>

The CPS table creator allows to mine data from the March analysis of Census data.

### Workforce data to consider for immigrant data analysis

Generate a report using the following criteria

- Education Attainment (sidebar)
- Nativity: Point of origin (sidebar)
- California (topbar)

You should now have data worth mining!

#### Time to interview the data!

- Paste the generated data into your spreadsheet of choice.
- Select the highest row with numbers and enter something like into the first empty cell to the right: `=D1/B1`
- Copy/paste the **formula only** to the empty cells corresponding to the other rows.
- Format the results to show percentages

### Adjusting the Data

- Under 'Data Options' change the year to 2007 (sidebar)
- Formula for comparing 2012 with 2007:
  - `new - old / old`

#### What does this show?

- Incoming workforce with high school and post-high school education dropped
- **Surprise:** Incoming workforce with bachelor's degree or higher is increasing.
  - Might be because more immigrants coming in have bachelor's degrees.
  - Might be because more immigrants are getting bachelor's degrees.
  - No way to figure out what this means from the data available. We need a journalist to investigate!

## Foreign Born Workers

- Choose 2012 (sidebar)
- Table definitions are **state** and **nativity** (sidebar)
- Refresh

## Visualizing the Results

I used Many Eyes (<http://bit.ly/many-eyes>) to quickly visualize the data. See

## Google Fusion Tables Bootcamp

<http://research.google.com/tables>

Presented by Rebecca (of Google)

## Examples

- NYC snowplow map
- Texas Tribune County map
- SF BAY Area Bike accident map
- Connection examples

## Cool Tools

- Chrome Extension to find tables
- New capability to turn shared Google Spreadsheets data into visualizations automagically
- Open Refine
  - Used to be Google Refine, a closed source product
  - <https://github.com/OpenRefine> #lamigrahack

## Open Refine

Makes turning data into machine readable format much, much easier. Can we have this as a hosted service somewhere?

- Used to be Google Refine, a closed source product
- <https://github.com/OpenRefine> #lamigrahack
- Installation: <https://github.com/OpenRefine/OpenRefine/wiki/Installation-Instructions>

## Google Data bootcamp Advanced Track

Presented by Rebecca (of Google)

<http://support.google.com/fusiontables>

## Questions to be answered

### 1. How can I show 2 values for a given polygon?

Fusion tables is about column/row data and merges, so you just use a merge column to show the data you want per polygon.

### 2. What are best practices for fusion tables?

Not yet covered

### 3. How do you use fusion tables with secure data?

Not yet covered

### 4. How do you import data files directly?

- Store in AppEngine
- Store in CloudSQL

### 5. How do you hide columns completely from viewers?

- It's a setting control

## Google Fusion Charts Templates

Looks like they implemented their own template language

```

```

## 1.5 Lean Startup 2012 Simulcast

I am Daniel Greenfeld ([blog](#), [twitter](#)), one of the principals and CTO at [Cartwheel Web](#), a software consulting shop that builds startups. I have two startups of my own:

- [Pet Cheatsheets](#)  
Build custom 1-page reference sheets for your pets in minutes.
- [MoveHero](#)  
Get free, anonymous moving estimates.

### 1.5.1 Other live-notes / live-blogs

As is frequently happens, I'm not the only one documenting the event live. Check out the hard work and excellent writing of others!

- <https://docs.google.com/document/d/1scTPA9HyYdhMISy1vm6XI8viEUOaKRzboDjTsI2W3fw/preview?sle=true#>
- [http://www.bizwatchsearchanalytics.com/reporting/?p=907&option=com\\_wordpress&Itemid=1](http://www.bizwatchsearchanalytics.com/reporting/?p=907&option=com_wordpress&Itemid=1)
- [http://www.shoestring.com.au/2012/12/the-lean-startup-conference-live-blog-from-4-am/#.UL0\\_H2zg4Ds.twitter](http://www.shoestring.com.au/2012/12/the-lean-startup-conference-live-blog-from-4-am/#.UL0_H2zg4Ds.twitter)

### 1.5.2 Eric Ries

- If you are building a startup you are trying to replace the big companies you dislike. The big companies started the same way you did, as a way to break the current system.
- We want to build the next big companies by mastering the disciplines of entrepreneurship.

### 1.5.3 Todd Park - USA CTO

- [https://twitter.com/todd\\_park](https://twitter.com/todd_park)
- Previous - CTO of Health and Human Services
- Current: - U.S. Chief Technology Officer and Assistant to the President. Tech entrepreneur-in-residence at the White House
- Worked a medical startup in Boston that went public in 2006
- Created two more health based companies in Boston

#### Work

- Often runs startup-like efforts called 'entrepreneurs in residence' inside the USA, which allows radical new approaches for the federal government
- The FDA has been working with this program to help the health of the nation.

#### Open Data Initiatives Program

- URL: <http://www.data.gov>
- Ronald Reagan is the godfather of the 'Open Data Initiatives Program'!
- GPS grew out of this system which has provided billions of dollars of business
- The government is the holder of immense archives of useful data

---

**Note:** Sunlight Labs (<http://sunlightlabs.com/>) is a group that works to translate the often **not-machine** readable data into formats that can be read and used by machines (and hence entrepreneurs).

---

### Todd is working on...

- See RFPEZ to get through government RFPs faster (find on GitHub)
- Getting your health records from the US government without pain.
- US Government runs 24,000 websites. How are they all done? How much replication happens?
- US government spends \$80 billion on IT per year. Needs to be cheaper! Use open source and better IT companies!
  - <https://github.com/presidential-innovation-fellows>
  - <https://github.com/presidential-innovation-fellows/rfpez>

---

**Note:** This is why OpenStack (<http://www.openstack.org>) was started at NASA by Chris Kemp. It was to reduce cost of single server setup from tens of thousands of dollars (mostly labor costs for meetings to discuss setting up single machines) to the same cloud costs paid for Amazon AWS. I can tell you as an ex-NASA employee that server provisioning was overly expensive as of 2010.

---

### 1.5.4 Diane Tavenner - Summit Public Schools

- <http://www.summitps.org/>
- Chief Executive Officer and Co-Founder of SPS

### Issues

- PROS: Lots of successes! Tons of High School graduates!
- CONS: Only 50% of their graduates finished their college degrees
- Something is wrong

### MVP concept

- What if instead of teachers directing classrooms, students went down their own path
- While this is not a new concept, they decided to map out the requirements extremely clearly to students and parents.
- They created an on-line testing system so that students could update their status by passing tests so they could see the results instantly.
- Added tons of testing and metrics. Rather than wait for years for results, they needed to know right away so they could fix it.
- Refused to use vanity metrics to promote the schools. They needed to know honest, real data on actual results - and kept even the bad news.
- Encouraged teens to provide feedback through mechanisms that teens like to use.

### What they discovered

- Lectures were not effective.
- Teachers were much more effective dealing with individual student issues, rather than just broadcasting knowledge.

### 1.5.5 Tendai Charasika

- <http://www.greaterlouisville.com/EnterpriseCORP/>
- Works with the Kentucky Innovation Network

### Get Out of the Building

- Get out and talk to users
- Get Uncomfortable
- Learn quickly and upfront if people actually want/need your idea implemented
- If you don't ask you miss out on what they really want.

### 10 pragmatic ways to get out of the building

1. **Don't Ask Your Uncle.** In other words, don't ask people you know will say nice things.
2. **Set up a booth, do a public demo**
3. **Interview potential customers.**
4. **Put your office where your customers are**
5. **Throw a party**
6. **Talk to experts in the field**
7. **Find the decision maker** (everyone else is just chaff)
8. **Listen to what customers are demanding**
9. **Pre-order, landing pages, analytics** (show demand for the product)
10. **Ask for the introduction**

---

**Note:** Idea: Market your tech startup by sitting in a coffee shop and showing people.

---

### 1.5.6 TWO PEOPLE - Eric Ries and Tereza Nemessanyi

- Tereza Nemessanyi (<https://twitter.com/TerezaN>)
- Talking about using general accounting practices.
- Stay away from vanity metrics, except for what goes into a pitch deck.
- Investors use vanity metrics to make investments in your project, but using them for concrete business decision making is dangerous.



- Issue: Investors often use your original vanity metrics when determining how well your project is doing

### 1.5.7 Beth Comstock interviewed by Eric Ries

- Beth is the Chief Marketing Officer of General Electric (GE). [https://en.wikipedia.org/wiki/Beth\\_Comstock](https://en.wikipedia.org/wiki/Beth_Comstock)
- GE believes that entrepreneurs are everywhere
- GE has to keep reinventing itself: **You don't last for over 130 years by staying static.**
- **Lessons learned:** Partner with outside firms to help bring outside ideas into the company
- Recently: GE got into energy storage (batteries) via startup/entrepreneurs and it is now a multi-billion dollar part of their business
- **Lesson learned:** Really focus on MVP before trying to make it perfect for market. This is critical before ramping up to large production efforts - otherwise you have no idea what the problems really are.
- **Statement:** You can fall in love with your technology or you can fall in love with what your customers think about your technology.
- They want help and will pay for it! See <http://www.gequest.com>

### 1.5.8 Jessica Scorpio

- Founder of <http://www.getaround.com/>, which lets you rent cars from other people who live near you.
- <https://twitter.com/jessicascorpio>

Were not sure if it would work, so began prototyping.

#### Efforts

- Worked with students out of Moffet field, near San Francisco, to see if it would work.
- Built an iPhone app right away to get them a working prototype.
- Competed in Tech Crunch Disrupt to get publicity and won.
- They have a custom product called CarKit to let it wire into your car.
- Worked in litigation because part of this means granting easier access to your car. What if someone else is driving it and wrecks it? By getting some laws passed in California they cleared up the rules for making this service work.

### 1.5.9 Daniel Kim

- Founder of Litmotors (<http://litmotors.com/>)
- Builds self-balancing contained motorcycles.
- <https://twitter.com/litmotors>

## Thoughts

- Building a car is hard.
- Building a car and mass producing a car is crazy hard.
- If you are creating a car company, you should know how to build a car, not just be a car executive.
- Trying to build the Model-T of the 21st century. Getting it right means positive income for 90-100 years.
- Different approach from segway
  - Spent a lot of money doing research if there was a need for a small, sustainable vehicle market.
  - Did building of product after doing market research
- Engineering:
  - Built by hand, rather via expensive machinery.
  - Didn't worry too much about meeting prototype deadlines
- Feedback
  - Did a small production round to demonstrate that people would buy it. This impressed investors
  - Got lots of feedback from users and drivers

### 1.5.10 Lane Halley

- Carbon 5
- <http://www.lanehalley.com/>
- thinknow

## Process for building products

- Sketch out your ideas as a team
  - Lowest response fidelity
  - Cross functional pairing is important
- When designers and developers work together, they need to understand each other's tools.
- Lean startup is great for design
  - Quick
  - Visual
  - Collaborative
  - innovative
- Use workflow sketches to determine the flow of a product
  - Don't worry if it's ugly, use paper
  - Don't use fancy tools
  - If you use fancy tools, you risk locking up your product in whoever controls the fancy tools.
- Wireframes

- balsamiq is great
- So is paper

### 1.5.11 Ron Williams

Kind-of-lean startup talk

- Founder of Knodes
  - <http://knod.es/>
  - <https://twitter.com/Knodes>
  - If they can figure out the right people into your funnel to being a user, your user becomes better than you about marketing your product.
- **build/measure/learn** for everything... or else
  - Build: If you don't build it you don't know if it can be done
  - Measure: Find out how it's used, by people or whatever
  - **Learn** from what you observe.
- Telling your team to **BE** lean is like a crash diet
  - **Don't say:** Hey I just read this awesome book and we're going to start doing these 15 things differently.
  - Changing habits is **HARD**.
- **Beeing** lean isn't your goal
  - The real goal is to have fun creating a product your customers love.
  - GitHub is a **GREAT** example.

### 1.5.12 Andres Glusman

- Works at Meetup.com as Head of Insights & Strategy
- <https://twitter.com/glusman>

RSVPs are going up? Here is why:

#### Myth: People give a damn about lean methodologies

- No one wants to switch gears
- No one wants to buy a process
- Instead of convincing, just start doing it.
- Avoid Malkovich Bias
  - The tendency that everyone uses technology the same way that you do.
  - Example: iPhone/iPad users often don't realize that the Android market is larger than the iPhone/iPad market.

### Myth: People want to test things

- People actually like to build things
- Because of this issue, try to test easy things.
- As you improve your system thanks to easy test results, testing becomes more exciting
- Failure:
  - Don't try to avoid failure, embrace it.
  - Learn from each mistake via metrics and tests and improve ever since.
- Go after the things that will cause us to fail as fast and often as we can.

**Reality:** People want to build and test things.

### Myth: You can test your way into a great experience

- Testing your way to an experience often means you create a complete and total mess
- Sometimes you have to restart from scratch and see how it goes.
- See <http://www.meetup.com/create/> to see what they've managed to get working

## 1.5.13 Panel - Getting engineers to embrace Lean

- **Moderator:** Even Henshaw
- Melissa Sedano (<http://www.bloomboard.com> - <https://twitter.com/Bloomboard>)
- Sam McAfee (<http://www.change.org> / <http://www.change.org/users/sammcafee>)

How to get developers/engineers to switch from Agile to Lean.

- Get engineers to embrace smaller prototypes
- Get your engineers to embrace metrics
- Throw away the code when you are done with the MVP

<b>Warning:</b> Read the 'Danger: MVPs often not disposable' section below.
---

### Danger: MVPs often not disposable

Throw away code after the MVP is done? That only works for established companies.

Anyone who thinks you can throw away MVP code hasn't talked to anyone at Twitter, GitHub, or 95% of other companies. They still run off the original MVP code. The only companies who can get away with throwing away MVP code are pre-existing companies with multi-million dollar budgets who use MVP efforts in tiny segments of their system architecture.

### 1.5.14 TWO SPEAKERS - Nikhil Arora and Alejandro Velez

- <http://www.backtotheroots.com/>
- Started selling at farmers markets
- Used a timer to gauge how long each person hung at their booth
- Got explosive growth
- Switched from selling mushrooms to selling mushroom growing kits all over the country
- Have a fish? Grow mushrooms! <http://www.kickstarter.com/projects/2142509221/home-aquaponics-kit-self-cleaning-fish-tank-that-g?ref=card>

### 1.5.15 Stephanie Yeager

@ [http://twitter.com/steph\\_hay](http://twitter.com/steph_hay)

Using words that help people find you and choose you

- You want people to choose you.
- But words describing superlative are overused. Everyone is 'the best'.
- Try using 'Lean content' to describe your product to someone who isn't you.
- Look for the **ah-ha** body language
  - See the questions they respond to you with before the **ah-ha** moment
- Use the mom test. If you feel uncomfortable explaining it to your mom, then you need to find a better way.
- **Growth goal:** Get found
  - Test your messages in AdWords. Test for clicks, not conversions.
  - Embrace the unsexy words in organic searches
  - Look for Entry Points and Top Content in GA

### 1.5.16 Steve Blank

[https://en.wikipedia.org/wiki/Steve\\_Blank](https://en.wikipedia.org/wiki/Steve_Blank)

#### Teaching Entrepreneurship

**What we used to believe:** Entrepreneurship can't be taught.

**What we know now:** Entrepreneurship can be taught to anyone who volunteers to try.

#### Learn Entrepreneurship

**What we used to believe:** Learn Entrepreneurship requires a lot of education

**What we know now:** Learning Entrepreneurship some theory and a lot of practice

**Warning:** Learning entrepreneurship from an educator is risky. Their experiences may not translate to today's conditions.

## Teach the Entrepreneurial API

1. Teach how to create a business model canvas
2. Teach understanding of Customer Development
3. Teach how to implement the plan using Agile Engineering

### 1.5.17 George Bilbrey

“Enterprise in the lean startup”

- Part of Return Path: <http://www.returnpath.com/>
  - new product: Anti-phishing system
- Built with small team inside of their large 400 person company
- Read all the lean books

---

**Note:** Read <http://www.amazon.com/The-State-Philosophy-Theodore-Andrew/dp/1480290556/?ie=UTF8&tag=cn-001-20&linkCode=ur2>

---

## Lessons learned

- Determine who the buyers really are.
- Bring in a Salesperson earlier in the process, however, the salesperson must like experimentation.
- Prepare to pivot: That means you have to be ready to admit you got it wrong
- Start small and organize for experimentation.

### 1.5.18 Ivory Madison

- <https://twitter.com/IvoryMadison>
- CEO and founder of <http://redroom.com>

“Bonfire of the Vanity Metrics”

- Vanity blinds you to a lack of actually important data
- Mark Twain: “Facts are stubborn, statistics are more pliable.”

## Don’t use these metrics

- Page views
- New members
- Total members
- conversion rate
- Percent growth
- Twitter followers

- Facebook friends or likes

### Characteristics of actionable metrics

- Measure success at your core business
- Show direct relations to revenue

### Your Four: Most important Metrics

- Measure revenue
- Measure Sales Volume
- Measure Customer Retention
- Measure Relevant Growth

### Find the big picture in???

---

**Note:** They switched back to the speaker after 2 seconds. :P

---

## 1.5.19 Ash Maura

“Getting the ultimate metrics dashboard”

1. Establish a standard measure of progress
2. Dave McClure’s Pirate Metrics (look them up)
3. As you gain users, it becomes harder to measure progress.

## 1.5.20 Leah Busque

- Founder and CEO of Task Rabbit

“If you had only \$1, where should you spend it?”

- Really understand your customer so you can target your acquisition techniques
- Be holistic:
  - test everything
  - not just channels
  - not just funnels
- Geo-targeting is critically important.
  - What works in one place will not work somewhere else
  - Test and measure the results

### 1.5.21 Big Panel

- Scott Cook (Intuit boss)
- Carol Howe
- Joe Hernandez
- Barath Kadaba (VP of engineering)

#### Question: What is the goal you have for your venture?

- You want to stay small and insignificant? (0%)
- You want to be giant and well known? (100%)

#### Making it happen

- Scott:
  - leaders need to change and lead this change into the business
  - Change things to create success after new success
  - Large companies typically get stuck and become stifling
  - Companies lean on politics and slide desk to stop changes:
  - Leaders need to stop deciding on opinion, but to work on actionable metrics

#### Components of making it happen

##### 1. Leader has to set the grand challenge

Barath Kadaba

- In 2008 he was told to change the lives of India. All the lives
- Given budget for just 3 people to do it.
- First effort:
  - Decided to focus on the lives of Indian farmers.
    - \* 150m+ of them
    - \* Contribute 25% of India's GDP
    - \* Most live in poverty
  - Decided to solve the narrow problem:
    - \* **Problem:** To whom can they sell their produce to get the best price?
    - \* **Solution:** Send farmer's SMS text messages with the latest known data
    - \* **Quick Implementation:** Faked it with hand-texted SMS messages to farmers.
    - \* HUGE success
- They got 20+ projects done this way
  - Team fought management death threats to stay alive



- Only survived because they were so small
- Yet increased the income of millions of farmers by 20%

## 2. Leader has to implement organization settings to make it possible to change

“Lawyers often are the barrier to success, they need to be instead considering how to make success more possible”

Joe Hernandez

1. Change **Mindset**, which will change **Behavior**.

How do you shift a group from saying **no** (leaders, lawyers) to saying **yes**?

2. **Democratize** Action

Create a clear set of guidelines in non-legalese that makes it easy for people to understand when they can move forward.

---

**Note:** How is a set of guidelines ‘**democratization**’?!? I think he needs a dictionary. :P

---

3. Becomes the power of success

Enable easily understandable rewards so you can demonstrate success. Payment can be financial or simple numbers.

## 3. Leader has to model pulling insights from both successes and failures

Carol Howe

- In 2009 created a start-to-finish app for Intuit that lets you take pictures of your tax documents and it files for you.
- But this wasn’t how it started:
  - Started with a photo capture app that would upload to your computer and that would file to the government
  - But when they created the app prototype, testers made it clear they wanted to just finish it on their phone
  - Stepped back and looked at the feedback from prototype users and listened carefully
    - \* Mobile fans raved in long discussions
    - \* Web fans said one word answers like, “nice” and “neat”
  - Started with launch in California and took lessons from there

## 4. Leader has to live by the same rules and disciplines as everyone else.

Scott Cook

- Test your beliefs the same way you make others test theirs.
- If you don’t test your beliefs, then you’ll drive into places based on opinion, not science.
- By testing your hypothesis, you don’t just get better results, you often have more fun.

### 1.5.22 Drew Houston

- Founder of dropbox.com

Q&A from questions given from the audience:

- **Question:** What do you look at in regards to metrics?
  - **Answer:** We look primarily for: “How many active users do we have?”
- **Question:** What tools do you use for gathering metrics?
  - **Answer:** The simplest tools possible to gather metrics
  - **Answer:** Store them in google docs and other simple tools
- **Question:** How do you find people?
  - **Answer:** Personal network
  - **Answer:** Connect with the developer/business communities
- **Question:** What are your goals and how are you accomplishing it?
  - **Answer:** Build something that makes me happy
  - **Answer:** Build something that makes others happy
  - **Answer:** Have fun making it work
  - **Answer:** Figure out how many users you need to get in order to do the startup full time.

### 1.5.23 Charles Hudson

- <https://twitter.com/chudson>
- <http://www.charleshudson.net/>

“Being a VC does not protect you from making boneheaded mistakes as founder.”

#### How they got started

- Saw that none of the games for Android were any good
- Decided to become the ‘android guys’
- Platform decisions matter
  - Tech
  - Distribution Channels
  - Can they go between systems?

#### Had to pivot

- Couldn’t monetize just on Android
- Tried to leverage switching to Kindle Fire and iOS
- Story isn’t done yet

### 1.5.24 Dave Binetti

- co-founder of Votizen
- <http://davidbinetti.com/>

#### When you do you pivot?

- You need to have a vision to make a decision based off of hard metrics.
  - Often people make a pivot based not on hard metrics but emotion
  - Pivoting doesn't mean changing your vision, it means changing your path

### 1.5.25 Mark Abramson

Did Lean Startup Machine and won it. Ran 10 experiments and pivoted 5 times during the conference

1. Experiment - Tax paying
  - Discovered that restaurants over 25 employees have to pay an extra tax
  - They all send it accountants and pay serious money.
  - No pain. Not worth doing.
2. Experiment - Happy Hour Marketing
  - No one has problems here. Not worth doing.
3. Experiment - Getting people into food places
  - Fierce competition everywhere. Not worth doing.
4. Missed
5. Experiment - Wine club for restaurants
  - 6 bi-monthly events in 2013 for the serious wino with exclusive chefs
  - \$1500 for 2 people for an annual membership
  - **People will pay for this service!**
    - They've made \$4500 in days
    - They could have sold out if not presenting

### 1.5.26 Marc Andreessen

- <http://a16z.com/>
- <https://twitter.com/pmarca>

Interviewed by Eric Ries. Notable quotes:

- Pivoted twice when it was still called, "We fucked up"
- When you get a ton of customer service requests it means you are succeeded.
- You have to move quickly in order to capture the market. You can't wait. Just have to move.
- Until your effort makes a product market fit, it's not a real company.

## We learned a lot of lessons from the dot-com crash

- Worried that people who lived through the crash are suffering psychological damage from the event.
- Many of the ideas of the time were valid, but were just too early.
- You can take the ideas of the time and with a twist, apply them to great success
- The bubble itself was only 18 months. From 3rd quarter of 1998 to the 1st quarter of 2000.

## Problems he hears in pitches

- Not every startup can be lean. Sometimes you need to just be audacious. Think Apple, Intel, or anything Elon Musk does these days.
  - You can't put a rocket into space on a lean program.
  - Don't let the lean startup methodology destroy other ways of doing things.
- Lean startup methodology used to avoid sales marketing strategy seriously.
  - Sales and marketing doesn't happen magically.
  - No matter how good your product might be, people won't come to it without sales and marketing.
  - Critical examples: Google, Salesforce, Facebook, and Twitter have thousands of sales rep.
  - Business applications do not sell themselves
  - Don't rely on anything going viral
- Entrepreneurs give up too quickly
  - Are you going to do the heavy lifting over a long period of time?
  - keep at it!
- Failure fetish
  - By taking the stigma out of pivoting, entrepreneurs have an excuse to not try hard enough. Don't be gleeful about failure.
  - People who claim to be "serial entrepreneurs" without success are giving themselves a fancy title for being a failure.
  - Preserve the good of failing, learn from it and succeed next time.
- History is weird
  - Winners are portrayed perfectly, losers are portrayed like idiots
  - He contends that winners often are just lucky enough to start at the right time.

## Tips

- Do not go public until your company has built a fortress. If you don't have all the positions filled, brand established, predictability in the market, then you are at great risk.
- Going public today is an extreme sport. It's very dangerous.
- Lean Startup is like the *Theory of Relativity for Business*.
  - We now have a process and science for getting things done.
  - **BUT** you still need old fashioned sales and marketing.

## 1.6 PyCon Poland 2012

Dates: September 13-16 Venue:

Hotel Przedwiośnie "Early Spring"  
Małocice Kapitulne 178  
26-001 Masłów k / Kielce  
center coordinates: 50.9024 N, 20.78021 E.

---

**Note:** Even though I don't speak Polish, I tried to capture some of the Polish talks.

---

### 1.6.1 Talks

#### Fractal Architectures

- by Laurens Van Houtven
  - <https://twitter.com/lvh>
  - Twisted Developer
  - PSF Member
  - Lives in Krakow

#### Talk Description

Traditional service architecture wisdom generally tells us to build services like this:

- Load balancer in front
- Web servers, preferably stateless
- Database (with a caching layer)

That works great for a wide variety of use cases. The point of this talk isn't to deprecate that design, but to discuss a radically different one.

The design I will present in this talk is one consisting of recurring, identical components. It localizes state to individual application servers and persists it to durable stores later. It aims to be easier to scale horizontally: that is, enabling you to increase throughput by simply adding more machines to the homogenous cluster.

I will talk about its benefits, such as performance and how it fits in well with many cloud providers' services, but also its downsides, such as the inherent complexities of distributed systems. These qualities are analyzed to come to a conclusion about which kinds of project this design is suitable or not suitable for.

In this talk I will discuss both the abstract concepts and the practical implementation that I have built using Twisted and Axiom (a simple object database on top of SQLite 3), which is currently running in production. Although I will touch on the practical implementation, the talk should still be useful for anyone wanting to implement a similar idea using different tools.

#### Standard Architectures

Check out Twelve Factor App.

- Level 1: Servers Database Cache
- Level 2: Application Servers
- Level 3: Load Balancing

The problem for you is that scaling all of these levels gets server and code expensive. You have to add in distributed data, messaging queues, and extra servers. Or pay companies like Heroku and dotCloud and Redhat a lot of money.

### Consider Instead...

- Sharding architecture
- Problems:
  - Expensive
  - Only for things on a Facebook scale.
  - Most people don't need this sort of thing.
  - Forces restrictions on code patterns.
- Advantages
  - Constraints on code means you have the freedom to do what you want within those constraints.
  - Lower latency
  - Great for when one user is only interacting with data that just affects themselves. For example:
    - \* Perfect for things like a webmail client. Most of the real behavior of the system is interacting with the client, not doing SMTP.

### Breaking the rules

Special cases aren't special enough to break the rules. However, practicality beats purity

—Tim Peters, Zen of Python

Sometimes it's good to farm things out rather than forcing it into your stack. For example, instead of doing the SMTP yourself, let Rackspace (Mailgun) or Amazon (SES) do it for you.

### The Diablo III example

Auction house could benefit from this architecture.

- Store the data in tiny places per user per general geolocation.
- Would work perfectly using SQLite3 per user if you add in Axiom
- Alternative databases:
  - PostgreSQL
  - Redis
  - MySQL (not recommended)
- Try to use byte-differential storage. Unfortunately, the only professional option for this method is Dropbox.

## Axiom

Links:

- <http://divmod.readthedocs.org/en/latest/products/axiom/index.html>
- <https://launchpad.net/divmod.org>
- <https://github.com/rcarmo/divmod.org/tree/master/Axiom>
- <http://www.devshed.com/c/a/Python/SSH-with-Twisted/3/>

Installation caveat: Axiom requires Epsilon in setup.py egg-info, so you need to manually install it first

Info:

- Runs on top of SQLite3
- Object database that works with one class per one table.
- Strongly typed
- Great for doing queues
- Does filestore
- Axiom powerups can have more than just static data, you can add behaviors

## Manhole

- Twisted project
- TODO: find details as to why he mentioned this

## Contention of the Talk

Either make things run faster or make things do less work.

- Query latency between servers (database, caching, http, etc)
- Caching really doesn't work for game servers and processing

**Talk Contention:** If you put it all on a bunch of small servers that can just do their limited collection of tasks, then you get to avoid latency issues between components.

## Poking holes in his own design

- Some of his data doesn't fit into small shards. So things like Encyclopedic data or 'world data' won't work. So where do you put this data?
- Size of data becomes an issue. Small shards hold less data
- Data updates with 10 million user stores means you have to update 10 million datastores
  - You need to keep most of your queries local per shard.
  - This forces tight coupling because a shard needs to really focus on shard data
- Querying across stores is hard. :-(
  - Data analytics is harder

- Big data requires special tools like Hadoop, Apache HBASE, Hive, etc
  - \* Odds are you don't actually need Hadoop. Unless you have terabytes of data you don't need these tools
- Transactions are a challenge.
  - \* Get the RDBMS to do it
  - \* You could do it in Python, but that isn't ideal
- No existing tools and frameworks designed explicit for sharding
  - Tools he mentions are general purpose that he uses for this sort of activity
  - Nothing like Django to composite everything together
  - No PaaS (Heroku, dotCloud, OpenShift) to do the system engineering for you
- No load balancing exists that handles this behavior. Which means depending on your setup you're still playing with load balancing.

## Testing

How do you do it?

- Careful focus on functional
- Careful focus on unit tests with mocks
- If you must, use Paxos algorithm to manage the transaction tests

## Forms in python - problems and my proposal of solving them

- By Szymon Pyzalski
  - STX Next Python Experts
  - <https://github.com/zefciu/Forms-in-python>

## Talk Description

My lecture would consist of two parts. First I would like to discuss what can a developer expect from a form library. Secondly I will show a design of one that would address all these problems.

## Introduction

The basis of reviews:

## Why are they important?

Forms are ubiquitous across all Python frameworks

- Python is a strongly typed language so we have to handle input properly
- Closest to the user
  - What they see most



- This is where they tend to see our mistakes
- Our first line of defense against security against CSRF and other attack methods.
- Boilerplate and repetition removal

## Scope of Features

All form libraries need to have the following components:

- User input handling
  - Type coercion
  - Validation
- Widget generation
- Data schema reflection
  - Critical boilerplate reduction
  - Try not to define both data and form schema

## Challenges

- Flexible but not full of feature creep
  - Easy to grow too big
  - but you can't make the project unmanageable
- Allow reflection but don't bind user's hands
  - If you can't modify the reflection then the form library quickly becomes useless on real projects
- Portable but allows developers to use specific features
  - If coupled too tightly then it's hard to move to other projects
  - If coupled too loosely then API can suffer.

## FormEncode

- By Ian Book
- Minimalist: only validation, coercion, html-filling
- Was recommended by Pylons book
- **Problem:** No schema reflection

## Django Forms

- Second attempt
- Plays best in the Django framework
- **Problem:** Hard to create new widgets

```
from django.forms import ModelForm, Textarea

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ('name', 'title', 'birth_date')
        widgets = {
            'name': Textarea(attrs={'cols': 80, 'rows': 20}),
        }
```

## Sprox

- Combines FormEncode and ToscaWidgets
- Extendable and easy to create new widgets
- **Problem:** unpleasant API

```
from sprox.formbase import AddRecordForm
from formencode import Schema
from formencode.validators import FieldsMatch
from tw.forms import PasswordField, TextField

form_validator = Schema(chained_validators=(FieldsMatch('password',
                                                         'verify_password',
                                                         messages={'invalidNoMatch':
                                                         'Passwords do not match'})),))

class RegistrationForm(AddRecordForm):
    __model__ = User
    __require_fields__ = ['password', 'user_name', 'email_address']
    __omit_fields__ = ['_password']
    __field_order__ = ['user_name', 'email_address', 'display_name', 'password',
    ↪ 'verify_password']
    __base_validator__ = form_validator
    email_address = TextField
    display_name = TextField
    verify_password = PasswordField('verify_password')

registration_form = RegistrationForm(DBSession)
```

## FormAlchemy

- Built on idea of schema reflection
- Generates forms and tables
- Type coercion

```
fs = FieldSet(User)
fs.append(Field('repeat_password').label('Repeat password'))

def password_match(value, field):
    if value != field.parent.password.value:
        raise ValidationError('Passwords do not match')
```

## Formish and Deform

- deform is a fork of formish
- don't do reflection
- Strong separation between schema and form
- Schema can be used for other data-parsing formats

```
class Schema(colander.Schema):
    password = colander.SchemaNode(
        colander.String(),
        validator=colander.Length(min=5),
        widget=deform.widget.CheckedPasswordWidget(size=20),
        description='Type your password and confirm it')
schema = Schema()
form = deform.Form(schema, buttons=('submit',))
```

## Anthrax

<https://github.com/zefciu/Anthrax>

---

**Note:** The name comes from *classic literature*, where Galahad visits Castle Anthrax and has his purity threatened.

---

His own forms library. Pre-alpha but it looks interesting.

- Highly modular. If you create a dependency, create a module
- 4 layers
  - fields
  - widgets
  - views
  - templates
- building blocks
  - forms: A collection of subcontainers and fields
  - Field: Knows how to validate and coerce a particular data type
  - Widget: a suggestion about presentation
  - Validator: Works on a form or container, ad-hoc or generic
  - Front-end: A complete system to render the form in forms like HTML, Dojo flavored HTML, Angular flavored HTML, XML, etc
  - View: Front end dependent object
  - Template: Let you define the output in a flexible way
- Building block relations
  - A form has fields. It can be rendered into a front end
  - A field has a list of widgets that are called depending on the format requested

- A front-end handles some widgets by assigning views to render them.

```
class RegisterForm(Form):
    __validators__ = [('equals', 'password', 'repeat_password')]
    __reflect__ = ('eplasty', User)
    __frontend__ = 'dojo'
    login = {'label': 'Login'}
    hash = salt = None
    password = TextField(widgets=[PasswordInput], label='Hasło')
    repeat_password = TextField(widgets=[PasswordInput], label='Powtórz hasło')
    ok = HttpSubmit()
```

My thoughts on it:

- I like the separation of layers.
- Like the way widgets are a list attached to a field, not just as a single widget per field
- I don't like the `__<SOMETHING>__` syntax. He likes them so we'll agree to disagree. ;-)

## Continuous integration - czyli jak spędzić weekend z dziewczyną zamiast z szefem

by Łukasz Langa

---

**Note:** Alas, I got convinced to try doing this late. And I don't speak Polish.

---

### Description

Większość z nas woli programować zamiast debugować. Tym bardziej mało kto lubi szukać błędów na serwerze produkcyjnym w sobotni wieczór. Jak tego uniknąć? Nie wpuszczaj błędów na produkcję. Podczas prezentacji pokażę jak przy użyciu takich projektów jak nose, jenkins, pyflakes, fabric tego dokonać.

This is a talk on continuous integration and best practices.

### Pyflakes and PEP-8

- Use tools to validate the quality of your code
- Develop good habits

### Coverage.py, nose, and other tools

- coverage.py lets you know how much is tested
- nose discovers tests.

### Automatic Installation

- Create a reproducible installation procedure that is executed via tools
- Don't do it manually

Useful tools include:

- Fabric
- Pip
- Virtualenv

## Set up your own QA servers

- Set up your own servers takes a lot of work and effort.
- OpenStack is nice because:
  - It does a lot of the lifting for you
  - Open source so you can use it for free and contribute back
- My Polish is bad so I wonder if I missed him suggesting paid PaaS like Heroku, dotCloud, et al

## What I missed about Python (and how JS taught me to love Python even more)

- by Audrey Roy
  - <https://twitter.com/audreyr>
  - <http://audreymroy.com>
  - PSF Member
  - My fiancée!

## Description

What happens when you take a Python developer and immerse her completely in JavaScript for a few weeks? This talk tells the story of my journey through JavaScript, from deep-diving in and looking for Python analogues in JS to achieving a greater understanding and appreciation of Python's design through comparative language study.

---

**Note:** Spent the talk taking pictures. Waiting for some notes taken by others that I'll be including here.

---

## Background

PyLadies, OpenComparison, PyCon Phillipines

## Total immersion in javascript

Several weeks of intense JSing

"JS is a terribly misunderstood language"

**Pre-Immersion:** python is better js is inevitable

Is JS good parts is good enough to just work with it?

**Is it worth it?** use ajax more? use full blown python soa and backbone.js? integrate js relatime features?

**JS spectrum:** avoid at all cost <—> Happiliy use 100% JS

## Findings

**Python is elegant!** good parts included (in js not so readily)

**JS ecosystem is thriving** works where python ecosys does not very ambitious

**Funcions in python** intedentation begets clarity of scope docstrings, named parameters minimizing anonymous funs

**Funcitions in JS** the infamous ‘var’ anonymous function. . . not no default params worse documentation tools

**JS is more functional, and thus better. Right?** closures as solution for scope leakage closures as classes closures as modules

(Audrey had an error on her slide, as if to emphasise cumbersomnes of JS closure hacks).

**Functional programing in python** iterators, generators list comprh. batteries included (itertools, functools, operator)

Lambdas – anonymous funcs are by design constrained.

You can nest functions in python has much more sense.

**What python cant do?** close over a non-global var in outer scope (python 3)

**Python classes are elegant!** scope is obvious class is namespace docstring

**Klass in JS:** at least two different hacks not one obvious way prototypal inheritance is complex different is not always better reality: prototypal is annoying

**Modules:** long history in python simple, defined by files

**Modules in JS:** just a script tag not part of the language, some libraries provide shim

**Packaging** no one true way in python, confusing

**Packaging JS:** many alternatives: npm, ender, jam, bower not build into the lang

**Code reuse** two obvious necessities importing libraries organizing code into directories

**Design patterns** classes provided by language decorators iterators and gens modules

python minimizes boilerplate js brings DP with libraries, many times

**Standard library** python has great stdlib, with some parts dated; opinionated js has none; jQuery, Node, no strong leadership

**Stdlib - datetime** js date.js died, although it was touted as best python datetime has limitations, but it’s there some good js libs exist, but lack recognition

## JS and polyfills

hack away what is not defined

## JS beats Python (reality wins)

js working in browser cross platform mobile dev tools huge innovation

## Summary

Python has good parts emphasised but has catching up to do

*You should try diving deep into JS.*

## Diversity helps community!

## Composability through multiple inheritance

Original Polish Title: *Kompozycja poprzez wielokrotne dziedziczenie*

note:: Talk is in English, but title and description were in Polish

- by Łukasz Langa
  - <https://twitter.com/llanga>
  - <https://github.com/ambv>

## Description

Jednym z momentów zwrotnych w historii było wprowadzenie produkcji półfabrykatów. Poprzez tworzenie prostych komponentów, które integruje się później w złożone produkty, producenci są w stanie budować szybciej i taniej, osiągając lepszą jakość. W tej opowieści programisty o sercu inżyniera opisuję, jak używam mechanizmu wielokrotnego dziedziczenia dostępnego w Pythonie, by realizować tę rzeczywistość przemysłową w kodzie źródłowym. Przykłady bazują głównie na Django, jego ORM, formularzach i klasowych widokach. Jednakże zasady, które opisuję, są ogólne. W trakcie wykładu wspominam o sposobach implementacji komponowalnych modeli abstrakcyjnych, a także mix-inów do formularzy i widoków. Tłumaczę, jak to podejście tworzy czytelne i zarządzalne idiomy, wraz z ich plusami i minusami. Zakończę podsumowując moje doświadczenia z próbą stworzenia biblioteki ściśle reużywalnych komponentów.

This is a talk on composition

## Act I: Exposition

- You can use legos to build small things and yet also to build big things.
- Lego blocks do have the composability feature
- To make components work, you need to have a framework that embodies compositionality.
- UNIX pipes are a good example:
- Composition isn't a science, it's an art
- Programming done well is art. Programming done badly is trash
- Jamie Zawinski: You can have a string that describes things accurately, or you can have a string that describes things accurately with flair

---

**Note:** According to Lucasz, the owner of Lego stole the invention from someone else and patented it, and made a fortune. The actual inventor died of grief.

---

## Act II: Rising Action

- If you use old-style style classes, you're going to have a bad time.

## MRO

```
>>> class A(object):
...     pass
...
>>> A.mro()
[<class '__main__.A'>, <type 'object'>]
```

## Thoughts on the diamond problem

```
>>> class A(object): pass
>>> class B(object): pass
>>> class AB(A,B): pass
>>> class BA(B,A): pass
>>> class C(D, AB): pass
>>> class D(A): pass
```

- Python has a definition of how to resolve the diamond problem in multiple inheritance.
  - Python has cooperative inheritance
  - In our example, you have to carefully watch how things are constructed

## Super was designed to solve this problem

But it failed. It's only useful in limited cases and can fool you.

```
class D(A):
    def __init__(self):
        super(D, self).__init__(arg_a='d')
```

- Don't omit `super(c, self).__init__()` even if your base class is *object*
- Don't assume you know what arguments you are going to get
- Don't assume you know what arguments you should pass to *super*

Warning: If you mix `ClassName.__init__()` and `super` your are going to have a bad time.

## Django ORM as a diamond pattern case study

- Problems: If you have a diamond pattern in Django it causes duplicate fields
- breaks the Liskov substitution pattern
- Example <https://github.com/ambv/django/blob/master/src/lck/django/common/models.py>
  - TASK: Check out what happens when you add `TimeTrackable` and something else from this file. You will apparently get duplicate fields.



### Act III: Example

- Use base classes in Django models is a good way to have easily maintained code. Examples:
  - EditorTrackable is a Model base that handles not just who can edit data, but also handles cascading deletes elegantly.
  - TimeTrackable is a model that tracks when something was created/deleted. Includes the following:
    - \* Created
    - \* Modified
    - \* cache\_version is an field that tracks which cached version is being displayed
- By composing his models on many projects via Abstract Models, he can test each reused abstract model extensively and repeatedly.

### Monkeypatching Django

- <https://github.com/ambv/kitdjango/blob/master/src/lck/django/common/monkeys.py>

### Using Python to Generate Art and Sound

- by Audrey Roy
  - <https://twitter.com/audreyr>
  - <http://audreymroy.com>
  - PSF Member
  - My fiancée!

---

**Note:** Lots of code samples with detailed explanations. Can't keep up with my notes but it's awesome.

---

### Description

I've used Python to draw rainbows of different shapes and colors, Gaussian clouds, and landscapes in perspective. I've also used Python to create sound effects for games. This talk explores my experiments with the various Python imaging and sound tools. First, I walk the audience through implementing basic audio building blocks with the Python stdlib's wave, math, and array modules. Then, I improve upon the code with NumPy and SciPy. Finally, I demonstrate how audio synthesis can be very similar to generative graphic art, using similar techniques to create building blocks for basic illustration.

### background of the Talk

A few years back she was painting landscapes and got tired of repetitive techniques so she decided to write a program to do it for her

## Introduction

- Overwhelming variety of Python libraries for audio/graphics
- Understanding the fundamentals first
- Helps you understand your options

## Overview

- Simple sound with the Python stdlib
- Numpy and Scipy
- Plotting sound arrays with Matplotlib
- Creative sound generation techniques
- Using the same tricks on graphics instead

## STDLIB!

- Very east to get started
- Other libraries can and are tricky to install

Parts she'll be using

- Wave Module
  - Use it to open and write .vave files
  - Introduced in stdlib 1.6 and hasn't changed much since
- Array Module
  - Using it to store data over time
- Math module
  - Using math.sin(x) to calculate 440 Hz audo audio samples

## wave, array, math

Generates a 440 Hz sine wave

```
import array
from math import sin, pi
import wave

SAMPLE_RATE = 44100
DURATION = 3

# TODO finish tons more code
```

Simplifying via a function

```
import array
from math import sin, pi
import wave

SAMPLE_RATE = 44100

def note() # TODO finish coding this out
```

### Can this be simplified further?

- Yes via NumPy arrays!
  - perfect for sound operations

```
# numpy.linspace(start, stop, num):
>>> linspace(0, 1, 10)
array() # TODO get this value

#sumpy.sin(x)
```

Now we show the simplified example:

```
from numpy import linspace, int16, sin
from scipy.io.wavfile import write # Using this because it's less code to use than
↳ the Wave module

def note(freq, duration, amp=10000, rate=44100):
    # TODO add code stuff here
    pass
```

### Is this music?

Not yet. You need chords for music!

### Chords for music

- Simply add 2 notes of different frequencies together
- She looked up Piano key frequencies on wikipedia

```
# chord function
def chord():
    # TODO get a sample of this code
    pass
```

### Using matplotlib to visualize the chord

She showed very nice code to plot out audio files.

## Concatenate notes into sequences

She showed using numpy's *concatenate()* function to add up arrays of sound samples.

## Weaving it all together

File structure

- notes.py
  - contains piano keys
  - contains imports of all the notes components
- Used numpy's *uniform()* function to create nice distributions of frequencies and durations
- Constrained the frequencies so they are humanly playable
- Explained use of *random.choice* over *numpy.choice*. Chose it because numpy's version is in beta.

## Results

- Colorful rainbow of sounds that sounds relatively pleasant to the ear

## Adding Gaussian Distribution

- Using an algorithm to make things more centralized.
- Which blurred things so instead of a rainbow of sounds it sounded like puffy clouds. :-)

## Introducing Pycairo

- Python API for cairo
- HTML Canvas uses cairo as well
- Showed how to use Gaussian algorithm to build clusters of dots

## Blocks and Puffs

- Show same technique as used in audio to create puffs of clouds
- Added blue background.
- Alpha and radial gradient background
- Adjust X and Y axis of gaussian to stretch the clouds into a more cloud-like shape

## Not just puffs

Can also use these processes on colors.

- Use uniform distribution for picking colors randomly
- Explore constraining to a subset of colors

- Used this technique and more to generate real paintings

### Summary: Think functionally

- Parametrize everything
- Use numpy array functions as much as you can
- Can combine wave, array, math from the Python stdlib for audio synthesis
- Sound and art composition are extremely similar
- Experiment with Gaussian distributions

### In Memorandum

- John Hunter, founder of Numpy, passed away recently
- <http://numfocus.org/johnhunter>

### Bonus Slide

- Tones + filters = sound effects
  - Play with looping, itertools
- Image sequences + Reportlab = flipbook PDFs
  - Use strokes and not fills
- Save image + sound sequences as videos
- Image composition can respond to audio input

### PyCon PL 5 lat

aka Five Years of Python

- by Filip Kłębczyk

---

**Note:** I'm translating this entire talk from Polish. I don't speak Polish except how to say, "Thanks" and now "Questions?".

---

### Introduction

- We've been doing PyCon PL for 5 years
- How many people have been to PyCon before? (about half)
- How many people have come to their first PyCon? (about half)

## How did we get started?

- The inspiration comes from various other user group conferences.
- We wanted an event for Python
- We also give credit to PLUG (Polish Linux User Group)
- Piotr Kasprzyk, Ph.D, gets credit and applause for his untiring work in the Polish Python community. Many attendees are previous students of his and he invited Audrey Roy to PyCon PL 2012.

## PyCon PL 2008

- October 18-19 in Rybnik
- Rybnik looks like a lovely city.
- The administration of the event has been a growing process.
- We had a big old-fashioned monitor on the presenter's desk.
- Lennart Regebro was the first foreign guest to PyCon PL!
  - Zope
  - Plone
- Lots of beer was drunk!

## PyCon PL 2009

- October 16-18 in Ustron
- So much fun last year we had to expand it!
- The day before the conference there was a big snow storm
  - It was much harder to get there
- Wesley Chun was the guest speaker
  - Google App Engine
- This year we didn't have a big monitor on the presenter desk.

## PyCon PL 2010

- October 8-10 in Ustron
- A little bit earlier so we could miss the snow storm
- The weather was beautiful!
- So nice that people wore short sleeves outside!
- Armin Ronacher was our guest speaker
  - Jinja2
  - Flask
- This guy played piano a lot

## PyCon PL 2011

---

**Note:** When asked who had been to all PyCons before this one people raised their hand. So I raised my hand too.

---

- September 22-25 in Machocice Kapitulne
- Took place at this hotel we are at now (Hotel Przedwiośnie)
- Guest speakers:
  - Brandon Craig Rhodes
  - Lennart Regebro
- Yummy BBQ outside with lightning talks!
  - Note: looks like a very impressive setup
  - You had to do it in your heavy coat
  - It was fun!

## PyCon PL 2012

---

**Note:** This weekend's conference!

---

- September 13-16 in Machocice Kapitulne
- Took place at this hotel we are at now
- Guest speakers:
  - Audrey Roy
- Have fun and do Python!

## Highly scalable services in Python

Original title: *Wysoko skalowalne serwisy pythonowe w OnetPoczcie*

- by Igor Waligóra
  - From Warsaw
  - Works on high integrity systems at Onet
- About Onet
  - Over 70 developers use Python full time
  - <https://twitter.com/onetpl>
  - news portal and email provider
  - One of the oldest sites in Poland

---

**Note:** I'm translating this entire talk from Polish. I don't speak Polish except how to say, "Thanks" and "Questions?".

---

## Talk Description (In Polish)

W prezentacji pokażemy nasz model rozproszenia usług pocztowych, jak zrobiliśmy to przy użyciu Pythona. Pokażemy metody realizacji stabilnych i skalowalnych systemów wykorzystujących niskopoziomowe biblioteki oraz model ich zwielokrotnienia i integracji. Uchylimy rąbek tajemnicy działania jednego z największych systemów pocztowych w polskim Internecie.

## System Architecture

Some sort of email system

- 200 servers
- 20 database servers
- 1 Petabyte of data
- 4.7 million users
- 80 thousand requests a minute
- Spam

## Server types

- SMTP
  - Postfix > MDA > Storage
- POP3 / IMAP
  - Dovecot > storage
- Webmail
  - PHP / JavaScript > Python Server > Storage

## Persistence Server

- check RFC 822

## Their Python server

- Tornado
- JSON-RPC

## libOP - API

- libAUTH
- libDB
- libOCACHE
- libANTYSPAM



- libStorage

## Making Python faster

Write in C, make bindings in Cython, import into Python

```
// op.c
int mparser_fetch(const struct mparser_server *mparser, etc){
    [...]
}
```

Workflow:

- Take C code that does what they need.
- Implement as Cython
  - <http://cython.org>
- Call the Cython modules from Python
- Put all the dependencies for the C library, Cython components, and Python into setup.py files so they can easily deploy

## System Summary

- C
- Cython
- Tornado
- Python

## Benefits

- Challenging but successful implementation
- Good performance
- optimized to handle any load
  - 20x speed over standard Python

## Critical tools

- PyPI
- Virtualenv

```
$ dpkg -i libop_1.1.0_amd64.deb
$ mkvirtualenv mparser
(mparser) $ source mparser/bin/activate
$ pip install -r requirements.txt
```

## Results

- really good performance
- 99.8% uptime
- Able to handle 500 thousand spam hits a minute

## Summary

- Build good systems
- C libraries are the way to go
- Use Python to build your stuff, but leverage in the C libraries
- Processes
  - Scrum
  - DevOps

## Blame it on Ceasar: A rant on calendaring

by Lennart Regebro

- <https://twitter.com/regebro>
- <http://python3porting.com/>
- <http://regebro.wordpress.com>
- Runs a consulting shop and does really good work.

## Talk Description

Timekeeping on all levels is surprisingly difficult. This talk explains why it's sort hard, and which parts Python can help you with.

- What is calendaring, really, and why is it so complex?
- What's in a year?
- Dissecting the Julian/Gregorian calendar
- Mesopotamian mathematics
- Time zones
- Recurring events is less fun than you think.

## Introduction

The problem with calendaring is that it is based off of multiple cycles that don't work well with each other

**Rome:** They had a 10 month calendar that made winter a dead period in the calendar. Eventually they added January and February

## Lunar Calendars

- The year is twelve lunar months long.
- The year is out of sync with the seasons
- Example: The Islamic Calendar

## Lunisolar calendars

- The year is 12 or 13 months long
- The year is kept in sync with season by leap months
- examples
  - Hebrew
  - Buddhist

## Solar Calendars

The villain of the story is Caesar

- The year follows the solstices/seasons
- The moon is ignored completely
- Examples:
  - French republican
  - Julian Calendar

Thanks to the success of the Roman Empire Europe takes a weird Solar calendar, and thanks to the success of Europe, the world takes it on too.

## How do you implement the calendar yourself?

You don't. You use libraries.

- Python has datetime, date, and calendar
- JavaScript is momentjs.com, which is the current best option for JavaScript
- Java has issues thanks to an early design decision mistake.
  - <http://date4j.net/>
- The US calendar shows Sunday as the first day of the week, which is confusing because it puts the first day on the weekend.

## Timezone woes

- There are not 24 timezones, there are standard times per country
- standard times change
- If you want to accurately describe times from the past, you need a database of timezone changes.

## Abbreviation Evil

- CST
  - Australia CST
  - China Standard Time
  - Chungua Standard Time
  - US CST

Timezones are based on politics, not science.

## Daylight Savings Time

- Changing the hour can cause problems with computers. Going over midnight twice breaks things.
- JavaScript handles this well
- Python handles it well

## Pytz discussion

He gave examples of how this module does a lot of the lifting for you on timezones and daylight saving time:

```
pytz 30 - 15 dateutil
```

## Advantage pytz

- Works well
- Except for POSIX

## Current standard specification

- TODO: Find out specified RFCs

## Libraries

- <http://pypi.python.org/pypi/tzlocal> (Download and test it out!)
- <http://pypi.python.org/pypi/icalendar/>
- <http://pypi.python.org/pypi/DateUtils>

## Datepickers

- Based on JavaScript if you are doing the web
- <http://arshaw.com/fullcalendar/>
- <https://github.com/collective/jquery.recurrenceinput.js>

## How to bootstrap a startup using Django

- by Jannis Leidal
  - <https://twitter.com/jezdez>
  - <https://github.com/jezdez>
  - Django core dev
  - DSF and PSF member
  - Co-maintainer of pip/virtualenv
  - works on PyPI
  - Lead engineer at <http://gidsy.com>

### Talk Description

Based on the experiences building Gidsy.com this talk will give you valuable insights as to how your infrastructure will evolve and how to set up the basic components (load balancer, web servers, DB, caching, celery, CDN, ...) of your site.

### What is Gidsy?

Gidsy is a place where anyone can explore, book and offer things to do.

### Why did they choose Django?

- Big community
- Network
- Language
- Many problems already solved
- The Admin

### Why Django is a good choice?

- Proven technology by similar use cases
- Stable APIs in a well-defined release process
- Good documentation with focus on prose
- Huge community of 3rd party components

### Implementing search

**Haystack:** <http://www.haystacksearch.org/>

- Needed customizable search abstraction
- Indexing, filtering, faceting, “More like this”

- Spatial search and sorting

## Implementing API

**Tastypie:** <http://tastypieapi.org/>

- Highly customizable Web API library
- Hooks for auth, throttling, caching, etc
- Backbone.js compatible
- Not for 3rd parties, just to serve out system content. So no need yet for OAuth or other protocols

## Task Queues

**Celery:** <http://celeryproject.org/>

If you have a user triggered process that will take a long time, use an asynchronous task queue to manage the task

- Async code execution
- Generate thumbnails, search index updates, caching, etc
- Collect stats without blocking

## Caching

### Memcached

- Periodic cache refreshing for high traffic sites
- Fragment caching with dates and cache version
- Cache warming during deployment
- All their code is based off the Django cache module. They don't use any libraries here because their data is too complex to rely on a caching framework.

## Workflow

“Always check in a module cleaner then when you checked it out.” – Uncle Bob

- Main branch is always deployable
- Development happens in feature branches
- Code reviews via pull requests
- Shared responsibility

## Testing

- Separation of fast and slow tests
- Full test suite via private Travis CI project
- Fast tests locally with django-discover-runner

- <http://www.djangopackages.com/packages/p/django-discover-runner/>

## Releasing

- Virtualenv/pip
- localshop as local PyPI host
- django-configurations for Django Settings
  - <http://www.djangopackages.com/packages/p/django-configurations/>
- Upstart for processes management

## Scaling up

- TODO - get slide that I missed

## Deploy System

- Builds are virtualenvs
- Atomic and orchestrated releases
- Lots of notifications
- Mix of Chef and Fabric
- Trying to open source it
- Using New Relic and Hipchat to track things
- Operations made as easy as possible via knife and fabric

## Operations

- Log everything you can for debugging
- If you deploy often you need immediate feedback
- Use services if you can:
  - Mixpanel
  - NewRelic
  - Librato
  - Papertrail
  - Pageduty
- Show the metrics on a screen in the office
- django-app-metrics to get a log trail from the system
  - <http://www.djangopackages.com/packages/p/django-app-metrics/>

## Things they learned

- Only scale when you need to, but be prepared
- Be pragmatic, use the best tool to do the job
- Automate as much as you can
- Continuous integration and deployment
- Make routine tasks as easy as possible
- Use services
- Display metrics

## Asynchronous and event-driven PyOpenCL programming

- by Tomasz Rybak
  - [tomasz.rybak@post.pl](mailto:tomasz.rybak@post.pl)
  - Debian Maintainer of PyOpenCL and PyCUDOA
  - Currently working at CodiLime
  - Worked at University of Geneva

## Description

OpenCL is the library, API, and programming language intended to help with performing computations on different computing devices like ordinary CPUs, graphical cards (GPU), specialized chips or FPGAs. OpenCL provides different profiles offering various capabilities (e.g. kernel compilation during runtime, executing native binary code, embedded function libraries) to allow to support different device types. Programming GPUs in Python is easy thanks to PyOpenCL (and PyCUDA). Not everything offered by OpenCL can be used in Python though, because OpenCL is defined assuming usage of the C language. Some functionalities, like calling function in response to event, require providing pointer to C function; fortunately such requirements show themselves only in the most sophisticated use cases. PyOpenCL helps with achieving high performance through asynchronous event-driven programming by allowing us to use many queues and many devices and by mixing synchronous and asynchronous calls. We can create quite sophisticated computation workflow and OpenCL will take try to use the available hardware, e.g. by concurrently call code and transfer data at the same time. New OpenCL versions allow for splitting one physical device into many logical ones (fission) which can be used to reserve some computing capabilities for usage in time-sensitive manner. We can also attach many devices to once shared context which allows to write code performing different tasks and computations in parallel. Some of the features offered by PyOpenCL are similar to those present in Python. Performing asynchronous computations on GPUArray and retrieving results later is similar to Python's Futures. So far it is impossible to retrieve Futures from GPUArray (to integrate GPU and CPU computing) but this seems to be the case of missing functionality, not incompatibility preventing it from happening. I want to show that creating programs performing quite sophisticated computations might be easy thanks to Python and PyOpenCL. I would also like to start discussion about current PyOpenCL limitations and to get feedback from PyOpenCL users.

## Increasing hardware parallelism

- More's law, increasing transistor density



- Power wall
- Chip's frequency doesn't increase anymore
- We get more cores instead
- No more automatic performance improvements
- Different programming models
- OpenCL has emerged as a standard intended to help with programming over this obstacle.

Summary: Use OpenCL to access the power of graphics cards as math processors

## OpenCL

- Standard maintained by Khronos
- Similar to OpenGL
  - Extensions
  - Different models for different devices
- Compile and binary kernels run on cores separate from CPU
- Based on C
- Includes events and asynchronous execution
- Information
  - <http://pypi.python.org/pypi/pyopencl>
  - <http://mathematician.de/software/pyopencl>
  - <http://documentation.de/pyopencl/>

## Basic OpenCL programming model

- Execution units hierarchy
  - Hosts
  - Platforms
  - Computing devices
  - Computing units
  - Processing elements
- Memory hierarchy
  - Global memory
  - Constant memory
  - local memory
  - Private memory
- Relaxed consistency of memory access
- Cache

## Execution run-time hierarchy

- Context
- Queue
- Work-group
  - A bunch of threads go into a work group
  - Which means you can have 100 threads run in a group, or 1000.
- Work-item

## Execution Models

- Task parallelism
  - One thread running computations
  - Possibility of running many threads at the same time
  - Require out-of-order queue or many queues
- Computation parallelism
  - Many

TODO - Get the parts I missed

## PyOpenCL

- ... and PyCUDA
- Python wrapper for OpenCL
- Not only wrapper
  - Pythonic
  - Object oriented
- Stable but still work in progress
  - extensions
  - high level programming

## OpenCL programming workflow

1. Compile kernels
2. Prepare data
3. Transfer data to device
4. Run computations
5. After finishing computations get results from device
6. Free resources

## Event based programming done in Python

- Instruct OpenCL to run computations
- Don't wait for data
- Computation will get to you when it's done

```
event = pyopencl.enqueue_copy(queue, a, agpu)
event.wait()

event = program.increase(queue, a.shape, None, a_gpu)

# later code
queue0 = pyopencl.CommandQueue(context)
queue1 = pyopencl.CommandQueue(context)
event = pyopencl.enqueue_copy(queue)
```

## Event-related objects

- Not all PyOpenCL functions and methods accept list of event to wait for
- We can wait for these events manually
- Or we can create a marker or barrier to force the end of a queue

## Fission

- Splitting one physical device into many logical ones.
- Can be used to reserve some computational power
- Solution similar to CPU virtualization
- No problems with device-to-device memory transfers

## Where PyOpenCL helps

Provides:

- Array
- Random number generators
- Single pass element-wise expressions
- Reduction
- Parallel scan

Designed so you aren't writing C code from scratch all the time to make your computations work fast in the graphics cards.

## Extensions

All extensions require pointers to the C so it's tricky to make them work

## OpenGL

Can share data between OpenCL and OpenGL

## Future of PyOpenCL

- Intention to share code between PyOpenCL and PyCUDA
- Increase number 3rd party libraries
- Some of those could be added to PyOpenCL
- Resolving existing problems
  - Adding extensions should be easier
  - Supporting additional libraries

## Suggestions

Check out <http://copperhead.github.com> as a way to wrap PyCUDA for easier coding.

## An Extreme Talk about the Zen of Python

---

**Note:** I would have taken notes but as I was presenting it would have caused a fatal recursion error. Fortunately, three people (Audrey Roy, Łukasz Langa, and Zbigniew Siciarz) submitted live-notes of this talk.

---

## An Extreme Talk about the Zen of Python

---

**Note:** live-noted by Audrey Roy (audreyr) and submitted as a pull request

---

## Daniel Greenfeld

- Family on both sides from Poland
- Principal at Cartwheel Web, PSF & DSF member
- Fiancee is @audreyr!
- Co-lead on OpenComparison which powers djangopackages.com and more

## Zen of Python

- By Tim Peters, author of Timsort which is used everywhere
- Extreme examples of each follow

## super()

- Built-in method
- Walkthrough of circle-ellipse problem
- Can create ambiguity
- Hard to remember syntax for super()
- Circle.\_\_init\_\_(self, outer) is more explicit and simpler
- Explicit is better than implicit

## Django CBVs

- Quiz: What is the ancestor chain for django.views.generic.edit.UpdateView?
  - Answer: There are 8 things. Hard to remember what each ancestor does.
- In super(ActionUpdateView, self).form\_valid(form), which form\_valid() is being called?
- If not careful, super() can cause MRO problems
- Possible mitigations:
  - Hope maintainers aren't angry
  - Convert to functional view
  - Explore better patterns

## Django

- Special cases aren't special enough to break the rules, although practicality beats purity
- WSGI is fixed
- Config & installation - working on it
- CBVs - working on it
- Not MVC. Model-Template-View. Web not necessarily appropriate for MVC.

## Web2py

- Where did the response object come from?
- 3 koans broken:
  - Explicit is better than implicit
  - In the name of ambiguity, refuse the temptation to guess
  - Namespaces are good...
- In their case, it's a design decision. "Practicality beats purity"
  - Easier for beginners
  - Easy to learn their namespace pattern
  - For experienced devs, saves boilerplate

- Web2py's easy installation process is where they shine

## OpenComparison

- Example showing a general exception
- <http://bitly.com/???>
- By printing (e), you don't get the error type or stack trace
- Fixed code with a custom exception that gets raised and prints additional info

## Decorators

- Awesome to use
- Easy to explain what they do
- He did a walkthrough of a sample decorator without arguments. Then one that accepts an argument.
  - 3 nested functions
  - Hard to read
- He corrected himself because he didn't use `@functools.wraps`, which is the better way to define decorators. More complexity.
- Hard to explain implementation
- If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea.

## Getting it done vs. technical debt

- Tests and docs take time. Skipping them risks:
  - Multiple coding standards
  - Deploying broken code
  - Problems upgrading dependencies
  - Forgetting install/deploy
- Must document:
  - Installation/deployment
  - Coding standards
  - How to run tests
  - Config
- Easy test patterns:
  - Always make sure test harness can run
  - Use tests instead of shell/repl
  - After 1st deadline, reject incoming code that drops coverage
  - Use coverage.py

## Namespaces

- Powerful, useful, precise
- Dangerous to use *import \**

```
>>> from re import *
>>> from os import *

>>> re.error == os.error
False
```

## Breaking built-ins

Continued from above:

```
>>> compare_builtins(re)
```

- Breaks compile() built-in

```
>>> compare_builtins(os)
```

- Breaks open() built-in
- Bad shortcut pattern to teach beginners. Technical debt.

## Summary

- Our community is built off of the Zen of Python
- Thank you: Richard Jones, Raymond Hettiger, Matt Harrison, Kenneth Love, PyCon Poland, others

## Extreme Talk about Zen of Python

---

**Note:** live-noted by Łukasz Langa (ambv) and submitted as a pull request

---

“What mistakes I did and how I correct them.”

The speaker: Daniel Greenfeld, both his parents’ ancestors were from Poland. Learned Python at NASA.

Tim Peters is the author of “Zen of Python”, also known for Timsort.

### The Opening:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.

- Sparse is better than dense.
- Readability counts.

### Example 1:

The `super()` method is doing things automatically and can create ambiguity. It doesn't adhere to the Zen of Python by being implicit.

Moreover:

- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

### Example 2:

The ancestor chain of `django.views.generic.edit.UpdateView` is very long (8 ancestors or so):

```
>>> pprint.pprint(UpdateView.mro())
[<class 'django.views.generic.edit.UpdateView'>,
 <class 'django.views.generic.detail.SingleObjectTemplateResponseMixin'>,
 <class 'django.views.generic.base.TemplateResponseMixin'>,
 <class 'django.views.generic.edit.BaseUpdateView'>,
 <class 'django.views.generic.edit.ModelFormMixin'>,
 <class 'django.views.generic.edit.FormMixin'>,
 <class 'django.views.generic.detail.SingleObjectMixin'>,
 <class 'django.views.generic.edit.ProcessFormView'>,
 <class 'django.views.generic.base.View'>,
 <type 'object'>]
```

Readability counts and this is not readable:

- it is very hard to actually remember what each mixin does
- they can have non-obvious side effects

### Possible mitigations for this view

- leave it as it is
- use concrete parent class methods instead of `super()` (bad idea)
- rebuild it to use functional views
- increase awareness of the design, simplify it, document it in detail

### Controversies

- Special cases aren't special enough to break the rules.
- Although practicality beats purity.



## Django

Django is pretty good about following the Zen of Python.

- WSGI
  - fixed
- Class-based views are too complicated (versus complex)
  - author works on document them better and simplify where they're too complicated
- Not MVC compliant
  - not a concern because what matters is separation of data and presentation

## web2py

Web2py argues practicality in some very specific places, will always be contentious.

- “Explicit is better than implicit.” - has implicit imports
  - On the other hand this implicitness makes it easier for beginners.
  - The namespace pattern is easy to learn.
  - Imports are boilerplate.
- “In the face of ambiguity, refuse the temptation to guess.”

## Exception handling

- Errors should never pass silently.
- Unless explicitly silenced.

Story: Django Packages. Once a day iterates across all packages. Updates the metadata from multiple sources. Sometimes the APIs go down or change. Sometimes objects get deleted. Sometimes network connectivity fails.

The first approach to a solution of these problems was to catch a bare `Exception` and print it out. Problems:

- the code is nearly silent: printing the exception causes the stacktrace not to appear
- `print` as a logger

## More controversy

- In the face of ambiguity, refuse the temptation to guess.
- There should be one– and preferably only one –obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.

Decorators are easy to explain for the user, not so much for the implementer. Especially if they should accept arguments. And don't forget about `functools.wraps`. Etc. etc.

Using decorators is like Zen. Writing decorators is not.

## Decorator Template

```
def decorator(function_to_decorate):
    def wrapper(*args, **kwargs):
        # do something before invoation
        result = func_to_decorate(*args, **kwargs)

        # do something after
        return result
    # update wrapper.__doc__ and .func_name
    # or functools.wraps
    return wrapper
```

```
# class as a decorator
class decorator_class(object):
    def __init__(self, function):
        self.function = function
    def __call__(self, *arg, **kwargs):
        result = self.function(*arg, **kwargs):
        # do stuff to result
        return result

@decorator_class
def hello():
    return 'hello'
```

On one hand:

- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

On the other:

- Practicality beats purity.

## Final section

Some things can take time like tests or documentation. You can skip them risking multiple coding standards, deploying broken code or problems upgrading dependencies. So if you have to skip documentation, at least write down:

- installation/deployment procedures
- coding standards

Easy test patterns for developers racing to meet deadlines:

- always make sure your test harness actually runs even if you don't have tests yet
- try using tests instead of shell/REPL
- after the first deadline, reject any incoming code that drops coverage
- use `coverage.py`

## Namespaces

- Extremely powerful

- Useful
- Precise

Beware: `from ... import *` makes development faster but it can be dangerous:

```
import re
import os

# clashing names!
assert re.sys == os.sys
assert re.error != os.error

# clashing builtins!
assert re.compile != compile
assert os.open != open
```

So don't do `from re import *`, etc. Especially, `import *` is not for beginners. If you do know Python and know about `__all__`, etc. you might use it if you're careful.

## Summary

```
>>> import this
```

## An extreme talk about the Zen of Python

---

**Note:** live-noted by Zbigniew Siciarz (zsiciarz) and submitted as a pull request

- by Daniel Greenfeld
- 

## Description

In the Python community we are taught from the outset of learning the language that the Zen of Python serves as a guide for how we should construct our codebases and projects. Rather than go into the zen-like meanings of each statement, this talk will explore how individual koans are implemented via detailed displays of sophisticated code examples.

## Introduction

No easy code examples, Daniel's gonna show some twisted, complicated code.

Daniel's grandparents come from Dynów, Poland. (Applause)

He was a Python programmer at NASA and later started consulting work.

Met @audreyr at PyCon 2010.

Runs some opencomparison sites - djangopackages etc.

## The Zen of Python

Written by Tim Peters, author of timsort algorithm - a really smart guy according to Daniel.

## The Opening

Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts.

Demonstrated using `super()`.

Geometrical figures, `Ring` derives from `Circle`. Obvious what `super` will do. But what if it wasn't so simple?

`super` does things automatically and can lead to ambiguity.

In the face of ambiguity, refuse the temptation to guess.

Absolutely inheriting `__init__` from base class: explicit, simpler, more readable.

## Explicit > Implicit

```
:: Circle.__init__ > super()
```

## Django CBVs

Composition, inheritance, polymorphism and other funny words.

What's the ancestor chain to `UpdateView`? Answer: 8 ancestors. Impossible to memorize what each of them does.

`form_valid()`, but which one?

OMG! OMG! OMG! Even more mixins. Let's print the MRO. A screenful of `<class '... '>` follows.

Filter the list on classes which have `form_valid()` method -> only 5 classes (*I was lucky*).

**MRO is simple, but simple is relative.**

## Moving on

## Django controversy

- WSGI (fixed)
- configuration and setup (working on it)
- CBVs (working on it)
- not MVC-compliant (and this is fine)

MTV != MVC

Is MVC applicable on the web?

The Zen of Python doesn't mention MVC.

## Separation of presentation from content

Django is fine. CBVs are not along the lines of Zen.

### Controversy: web2py

Implicit > explicit.

Follows it's own design patterns.

Where are the imports? No imports necessary. WAT?

---

**Note:** Nobody expected Spanish Inquisition!

---

web2py breaks 3 koans of the Zen. Implicit behaviors, ambiguity, namespaces.

*Although practicality beats purity.*

Easy to install, easy to learn, less boilerplate. Web2py <3 Windows.

Similar to Django in terms of contention and trackage.

## Exceptions in exception handling

### Django Packages

Updates metadata from PyPI, Github, Bitbucket. PyPI unstable, APIs change, projects get deleted etc.

First: concatenating some string with error messages from exception handlers.

Traceback wanted. Type of the error, message, location.

Code is silent - for no good reason apart from laziness.

**Solution:** added logging in `__init__` in a custom `Exception` subclass.

Code is not silent anymore. Errors are noisy.

### Cleaner code

Even more controversy. (*Unless you're Dutch*).

## Decorators

Decorators are easy to explain!

Wrapper function running code before/after the decorated function.

Getting harder to explain... closures etc.

Now let's talk about decorators with arguments. *general laughter*

Danny is evil, uses confusing names: `multiplier`, `multiple`...

Whew.

Don't forget `functools.wraps`. The decorator code in the slides is growing like a tumor.

*It's not easy to (explain how to) write decorators.*

But decorators are awesome! Using them is like Zen, writing is not.

## The last section

### Getting it done vs technical dept

Tests & docs take time. Do we have to do them? Maybe not. But it brings a lot of risks.

### Must-have docs

- installation/deploy
- coding standards
- how to run tests
- version information

### Test patterns

Test harness must at least run even without tests.

Use tests, not shell/repl.

Use coverage, reject code that drops coverage.

Don't use doctests.

### Namespaces

Powerful, useful, precise.

`import *` makes development faster. **IMPORT ALL THE THINGS!**

Confusing imports, same names in `os` and `re`. Subtle trouble!

Replaces things from in `builtins` (`os.open` breaks `open`)

### The `open()` story

`os.open` needs different arguments than `open`. You're screwed if you confuse these calls.

### Contention

`import *` is for people who know what to do.

Remember `__all__`.

## Summary

The Zen of Python (repeated)

## One more thing

Capoeira moves!

## Schedule

- <http://pl.pycon.org/2012/en/agenda>
- CouchDB and Python (unfortunately was doing client work so worked through this)

## Invited Speakers

- Audrey M. Roy
- Jannis Leidel
- Laurens Van Houtven
- Kai Diefenbach
- Stefan Kögl

## Sponsors

- Python Academy: <http://www.python-academy.com/>
- Allegro
- Onet
- Megiteam
- OSworld.pl: <http://www.osworld.pl/>
- OSnews.pl: <http://www.osnews.pl/>
- Polish Python User Group: <http://pl.python.org/>
- Bioinformatyk <http://www.bioinformatyk.eu/>
- Linux Magazine Poland: <http://www.linuxmagazine.pl/>
- Wingware Python IDE
- Github: <https://github.com>
- Enthought
- Coders Mill

## 1.7 Pycon Philippines 2012

Dates: June 30 & July 1 Venue: University of Philippines, Diliman

### 1.7.1 Keynote

I was honored with the opportunity to keynote the event. My slides are available at:

### 1.7.2 Talks

#### Basic Python

- by Allan Palo Barazone
- <http://twitter.com/titopao>
- Python since 2007, but since early 2000s
- Affiliated of Wikimedia Philippines, Inc
- Major equipment issues including the microphone.
- Live code writing is never a good idea. :P

#### About Python

- Started by Guido van Rossum a.k.a. Benevolent Dictator for Life
- Named after **Monty Python and the Flying Circus**
- Logo of Python is the snake
- Dynamically typed

#### Variants of Python

- CPython
- Jython
- IronPython
- PyPy
- Stackless Python

#### Prerequisites

- Python 2.7
- editor [scintilla.org/SciTE.html](http://scintilla.org/SciTE.html)

#### Hello Python

```
print("Hello Pycon!")
```



## Assigning Variables

```
PI = 3.1415
answer2life = 42
_secret_recipe = 0
x, y = 4, 20
```

## Dynamic Typing

```
a = 10
a = 'python rocks'
a = True
b = None
a = b
```

## Numeric Data Types

```
>>> print range(5)
[0, 1, 2, 3, 4]
>>> a = 9
>>> b = 2.0
>>> c = 0x999
```

## Operations

```
>>> 2 ** 10
1024
>>> abs(-1) # absolute
1
>>> 5 % 2 # Modulus
1
>>> hex(12)
'0xc'
>>> oct(100)
'0144'
>>> pow(16, 0.6)
5.278031643091577
```

## Booleans

```
>>> True
True
>>> true
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

```
and
or
is
is not
```

## String Operators

```
>>> len('Hello')
5
>>> 'hello'.upper()
HELLO
>>> s = 'Hello PyCon'
>>> s[:5]
'Hello'
```

## Sequences

```
>>> l = [1, 2, 3, 4]
>>> t = (1, 2, 3, 4)
>>> l.append(5)
>>> l
[1, 2, 3, 4, 5]
```

## Indentation

- Code blocks are defined by indentation
- The standard is 4 spaces.

## Python tricks you can't live without

- by Audrey Roy
- <http://twitter.com/audreyr>
- <http://audreymroy.com>
- My fiancée!

---

**Note:** Audrey used my laptop to present and I was manning the <http://twitter.com/pyconph> feed. Otherwise this would be full of stuff. What I'm going to do is take notes of the video and add them later.

---

## Django Quickstart

- By Marconi (@marconimjr)
- on Facebook he is 'Alexander Pierce'
- Wore a github shirt and gave shout out to the pony

- Built off of Audrey's talk. :-)

## What is Django?

- MTV framework
  - Template = View
  - View = Controller

## Demo app

- [quickstart.marconijr.com](http://quickstart.marconijr.com)

## Set up develop environment

- virtualenv + virtualenvwrapper

```
# .profile on OSX or .bashrc
export WORKON_HOME=~/.Envs
source /usr/local/bin/virtualenvwrapper/sh
```

## Creating your virtual environment

```
$ mkvirtualenv pyconph
$ workon pyconph
```

## Installing Django

```
$ pip install Django
```

## Create Django project

```
$ django-admin.py startproject quickstart
$ cd quickstart
$ python manage.py runserver
...
Development server is running at http://127.0.0.1:8000
```

## Directory structure

```
quickstart
|-manage.py
|-quickstart
|  |-__init__.py
|  |-settings.py
```

```
| -urls.py  
| -wsgi.py
```

## settings.py

```
DATABASE = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': 'dev.db',  
    }  
}  
  
PROJECT_ROOT = os.xxxxx
```

## Add templates

```
mkdir templates
```

```
quickstart  
| -manage.py  
| -quickstart  
| -__init__.py  
| -settings.py  
| -urls.py  
| -wsgi.py  
| -templates
```

## Game Programming with Python

- by Mr Sony Valdez
- <http://twitter.com/mrvaldez>
- Fantastic speaker
- Need to introduce him to Richard Jones
- OMG he used notepad to present and pulled it off!

## Quotes

- “Are you afraid of math? Too bad! In this tutorial you will learn how to math! Scary, isn’t it?”
- If you’re not familiar with cartoons I feel very sad for you.

## Ph Game Development Industry

- <http://www.igda.org>

## What you need to know to program games

- Programming language
  - Traditional
    - \* C
    - \* C++
  - New
    - \* Flash
    - \* Java
- Math
- game design patterns
- C++
- Flash
- Lua
- Python
- Javascript

## Math

### Coordinate System

- x,y system

### Collision Detection

- Collision detection is the algorithm used to see if two sprites intersect
- If two rectangles overlap then there is a collision. An event is triggered.

### Game Design Patterns

- **Game Loop** is simply an infinite loop in which input, updates, and draws occur. Each iteration is what is called a 'Frame'
- **Game object** represents an object in the game

### Game Platforms

- Programmers use whatever is available
- half-life, Warcraft 3, Multimedia Fusion, GameMaker, and pygame

## pygame

- Based on Python
- Object oriented

## Let's make a game

- Python
- Pygame
- Shump

## Functional Programming in Python

- by Malcolm Tredinnick
- malcolmt
- Started in Python in 1.4

## Intro

- Python is more than just OO, it's also functional
- Almost every language we use is imperative
- Python can be functional

## The cheatsheet for this talk

- **map()**
- **filter()**
- **functools** module
- **itertools** module
- list comprehensions
- generators

## Functional programming is...

- ... difficult to define precisely
- Good! (most worthwhile systems are)

## Maybe this?

- transform data structures, don't manipulate state

## Most useful?

- concentrate on function return values, not side-effects

## Python Functions

- Functions are first class objects
- You can pass them around as arguments to other functions
- You can construct them dynamically

```
def my_function():
    print "hello"

def show_string(func, arg):
    print func() + arg

>>> show_string(my_function, " Goodbye!")
hello Goodbye!
```

## Useful language features

- **lambda** expressions
- **functools** module
  - TODO: check out `partial()` capability in **functools**
- **itertools** module

## Pull out the even numbers

```
def evens(seq):
    results = []
    for item in seq:
        if item % 2 == 0:
            results.append(item)
    return results

# List comprehension way
def evens(seq):
    results = [x for x in seq if x % 2 == 0]
    return results

# pull out the even numbers
def is_even(value):
    return value % 2 == 0

def evens(seq):
    return filter(is_even, seq)

# using partials
from functools import partial
```

```
def is_even(value):  
    return value % 2 == 0  
  
evens = partial( filter, is_even)
```

```
>>> evens([1, 2, 3, 4, 5])  
[2, 4]
```

## Types of transforms

- Apply a function to every element
  - map()
  - list comprehensions [x for x in ...]
  - generators (x for x in ...)
- Select elements and filtering
  - filter()
  - itertools.dropwhile()
  - itertools.takewhile()
- combining elements (folding)
  - functools.reduce()
  - manual loops
    - \* sometimes a good idea
- unfolding
  - Manual loops
  - Recursion (sometimes)

## Ansible - code deployment made simple

### Simple Deployment and Configuration

- by Rodney Quillo
- <http://capsunlock.net>
- <https://github.com/cocoy>

### What is Ansible

- You should make it easy to get servers running
- Ansible aims to solve deployment issues
- <https://ansible.github.com>
- 3-in-1: Just like filipino coffee



## Dependencies

- Jinja2 (in case you want fancy templates of configuration files)
- PyYAML (for settings configuration)
- parameiko

## Question

- Why isn't ansible pip installable?
- How does this compare to Salt Stack?
- Why YAML for configuration?

## I didn't know it's Python: Python Advocacy

- by Allan Palo Barazone
- <http://twitter.com/titopao>
- Python since 2007, but since early 2000s
- Affiliated of Wikimedia Philippines, Inc
- Good advocacy talk

## How you attract tourists

- ... use paid television ads?
- ... use print ads?
- ... use social media?
- ... and still be consistent

## Mount a campaign!

- Slogan: It's more fun in the Philippines!
- Logo and slogan alone do not make a good campaign

## People need to know:

- What's in it for me?

## Why Python?

- The usual 'advocacy' stuff.
- Very, very gentle learning curve
  - Even complete newbies can understand

## Prototyping/Gluing

- Easily done with Python
  - Low-level modules in another language (like C or C++)
  - Then Python joins ‘em all together
- Python’s simplicity allows easy rewrites of prototypes
  - competitive edge over C/C++/Java

## Python is...SIMPLE

- Faster learning curve
  - Easier to pick up than ‘traditional’ Filipino CS 101 languages
  - Focus on thinking, not syntax
- More readable than other languages
  - Probably influenced by GvR’s math background
  - Similar to pseudo code
  - Hard to obfuscate

## Works with JAVA and .Net!!!

- Reality: IT industry in PH is Java/.Net centric
- problem:
  - We’ve already invested in Java/.Net technologies - will we have to rewrite the stuff?
  - NO!

## Ten things you didn’t know that use Python

- Google!
- YouTube
  - <http://highscalability.com/youtube-architecture>
- Blender
- reddit
- Disqus
- Dropbox
- Facebook
- Instagram
- Pinterest
- Pywikipediabot

## Maps of Imaginary Lands

- by Malcolm Tredinnick
- malcolmt
- <https://github.com/malcolmt/imaginary-maps-in-django>

## The Goal

- Build an imaginary land

## The Problem

- Not trivial
- Some preparation required
- May be new
  - This should **not** be a problem, but... :-)

## The Solution

- I have provided running code
- Github URL at end of the slides.

## What is success?

- Get up to speed on Django and GeoDjango
- Run (and read) my code
- Do something better!

## The Secret Tip

- All maps are mashups

## The Stack

- PostGUS
- OpenLayer.js
- Mapnik
- Tilecache
- GeoDjango

## OpenLayer.js

- Client side, Javascript framework
- For doing maps layering

## Mapnik

- Server side way to combine data sources
- Different details and different zoom levels
- Input from raster or vector formats

## Tilecache

- Caching tile
- Use this or mod\_tile or tilestache or other
- Avoid recomputing common data

## GeoDjango

- Use views to provide subset of data
- Easy default output in formats understood by OpenLayers

## Imaginary Maps

- Need to replace base image
- GeoAdmin needs to be customized for the imaginary maps
- Mapnik WMS server running locally

## Sample Model Code

```
class Track(models.Model):
    name = models.CharField(unique=True, max_length=50)
    path = models.LineStringField(geography=True)
    objects = models.GeoManager()

    def __unicode__(self):
        return self.name
```

## Closing Keynote: Design your open source project

- by Bryan Veloso
- <https://twitter.com/bryanveloso>
- <https://github.com/bryanveloso>

---

**Note:** Beautiful and inspirational!

---

---

**Note:** Bryan got schooled on that cats have 3 whiskers per side and the Octocat only has 2 whiskers.

---

## Choice Quotes

- I know what you really want: beer!
- Open source is about collaboration
- GitHub is your new portfolio! Hi @pydanny!
- If you test, use <http://travis-ci.org>
- Sphinx pretty much powers all of Python's documentation
- Trade development knowledge for design knowledge
- <https://github.com/edu>

## About Bryan

- Github employer
- <https://github.com/jdriscoll/django-imagekit>

# 1.8 DjangoCon Europe 2012

## 1.8.1 Keynotes

### Jacob Kaplan-Moss

```
{{ keynote }}
```

- `jacob**@jacobian**.org`
- Keynotes are hard.
- Researched to find different types of keynotes
  - Announcements
    - \* Steve Jobs
  - Technical Tour-de-force
    - \* David Beasley style
  - State of...
    - \* West Wing style
    - \* Make it dramatic
  - Celebration

- (Constructive) Criticism
  - \* Cal Henderson’s 2008 DjangoCon talk
- Inspiration
  - \* Neil Gaiman
  - \* Adam Savage

### Se he’s going to do the following talks:

- Technical
- State of...
- Celebration
- Criticism
- Inspirational

### Technical Talk

---

**Note:** @jacobian pointed me out and embarrassed me so I didn’t finish copying out his sphinx example.

---

Sphinx is awesome:

```
.. code-block:: html+django
```

Useful Sphinx stuff for authentication-protected static files:

- **static:** <http://lukearno.com/projects/static>
- **barrel:** <http://lukearno.com/projects/barrel>

These sorts of integrated components are an incredible indicator of some of the awesomeness of Python

### State of Django Talk

- Django is doing very well
- Each release has more and more incredible stuff added.
- “Always feels like we are not moving as fast as we should, but when you look at what’s been accomplished it’s amazing. Especially since forward compatibility has been pretty well maintained.”
- Django 1.5 should give us
  - Python 3 support
  - Composite keys

## Celebration

- This year has seen a ridiculous amount of adoption for Django.
- There is no longer an industry where Django does not exist.
- Django is now considered boring compared to things like node.js. Exciting is good when you are trying something new, but “exciting” and “production” should never be combined in the same sentence.
  - See this article [http://blog.pinboard.in/2010/01/technical\\_underpinnings/](http://blog.pinboard.in/2010/01/technical_underpinnings/)
- Notable tech acquisition for Instagram.

## Criticism

- HTML5 issues
  - Bruno’s floppyforms handles the form elements for you: see <http://django-floppyforms.rtfld.org/>
- Real-time
  - State of the art: parallel MVC stacks.
  - Mentioned Meteor framework as state of the art.
  - You are fooling yourself if you don’t realize that Meteor style systems are not the future of web development.
  - Are we doomed to callback hell?

```
        });  
    });  
});  
});  
});
```

Can we not do client/server apps purely in Django?

```
url('people/1/', person_detail)  
  
def person_detail(request, pk):  
    ctx['person'] = get_object_or_404(Person, pk=pk)  
    return (request, template, ctx)  
  
{% person.first_name %}
```

## Inspirational Talk

Jacob’s uncle, a lawyer asks: “But if you give it away, how will you make money?”

- By giving it away, people have made a ton of money with Django
- Jacob is doing quite well

## Summary

- Huge community brought in by our boring, stable system.

- Now we can get really crazy with Django
- Make good art

### Karen Tracey - Django and the Community

- [http://twitter.com/km\\_tracey](http://twitter.com/km_tracey)
- Been programming since 1987
- Django Core dev for a while.
- Crossword puzzle constructor since 2001
- Cat rescuer since 2009

---

**Note:** I've been programming since 1980. See <http://mytechne.com/user/pydanny/programming-languages/>. I win on the age game. So there.

---

### Why?

- Django's community is one of it's greatest strengths

### Her Django story

- 2006 she found Django
- Django open-sourced a year earlier
- Django 0.96

### Puzzle Database

- Aid in constructing puzzles, accessible from construction tool
- Amassed over ~5 years
- ~5,000 puzzles, ~100,000 unique entries, ~500,000 clues

### Project: Web front-end for database

- Primary goal: better ability to see data in crossword puzzle tool
- Secondary goal: learn Python

### Survey of Python web frameworks in 2006

- Pylons
- Turbogears
- Django



## Snag: Weird database

She wanted to use her old tables instead of Django generated tables. Then she wasn't sure about the code patterns she was exploring.

```
class Entry(models.Model):  
    entry  
    # TODO find the rest of this content
```

She contacted django-users on mailing list and talked to Malcolm and Adrian. Submitted a patch and got it accepted fast.

## Sadness

- Probably never happens today that a person contributes so quickly.
- Everything needs tests before submissions are accepted.
- Balance stability with wow-factor

## Back to the mailing list post

- She was hesitant to sign her name
- Open source has bad with regard to treatment of women
- Confident of technical ability
  - ... but conscious she didn't know much about web programming
- Would she get more respect if she didn't reveal her gender

## Plea: encourage women

- Women actively discouraged from participating in open source communities
- Please don't join in bad behavior
- Speak out against it when you see it

## Why did she become so active?

- Learn more about Django
- Improve communication skills
- Help people
- Puzzles!
- Long range-goal: get a job

### What did the Django community gain from Karen's involvement?

- Lots of triage/bugfixes prior to 1.0
- Some features/bugfixes since 1.0
- Helped many people learn Django

### What did she gain out of Django?

- Become core committer in 2008
- Asked to write a book in 2009 (got published in 2010)
- Got a great job in 2010

### Get involved!

- Community events, big or small
- Mailing lists
- IRC
- Stack overflow
- Ticket triage
- Bug fixes
- Feature development
- Patch review
- Blogs

### Jessica McKellar

- <http://twitter.com/jessicamckellar>
- <http://jesstess.com>
- Kernal Engineer
- PSF Board Director
- Co-lead of Boston Python

### Theme talk

- Make me make good choices
  - How to make a proper internationalized site
  - Education and best practices by default for novice web developers

## Accessibility

- Visual
- Motor
- Auditory
- Cognitive

## Visual Accessibility

- 7-10% of Caucasian men have some form of color blindness
- 2.6% of the global population is visually impaired
- <http://www.w3.org/WAI/>
- <http://www.section508.gov>

---

**Note:** From 2000 to 2010 I was heavily involved in Section 508 implementations for various US government funded projects including <http://science.nasa.gov/>

---

## Accessibility Guidelines

- Alt-text on images
- Accessible intra- and inter-page navigation
- Audio and video accessibility: captions, transcriptions
- Indicate important info by other things besides just color
- TODO: Finish

See [django/django/commit/29a80354ab5e5b85fa37863f70b1cf95646dc699](https://github.com/django/django/commit/29a80354ab5e5b85fa37863f70b1cf95646dc699) being fixed!

<https://github.com/django/django/commit/29a80354ab5e5b85fa37863f70b1cf95646dc699>

## Django Accessibility

- How can Django help people avoid, detect, and address accessibility issues?
- Set a good example: audit ourselves!
  - Websites
  - Conferences
- Accessibility tutorial?
- Accessibility checklist?
- Warnings on easily correctable issues?

## Security

Django handles the following:

- XSS
- CSRF
  - But instruction on how to do it with POSTS could be better
- SQL Injection
  - Warnings on RAW SQL could be better
  - ORM is EXTREMELY secure
- Clickjacking
  - Super easy to enable, but not set by default
  - Documentation on this is kind of buried
- Cookies
  - Important things are possibly set the wrong way

## django-secure

- Great little app.
- But if there are that many stupid little things that need to be checked, maybe the defaults should be changed?

## How about a security tutorial?

- Teach people from the start what they should be doing
- Include a security checklist

## Internationalization

Done:

- Localization
- Translation
- Timezones
- Django natively supports Unicode data everywhere
- How about:
  - Internationalization tutorial?
  - Internationalization checklist?

Existing Security pages exists, but needs work: <https://docs.djangoproject.com/en/1.4/topics/security/>

## django.contrib.auth.models.User

- First name and last name is very specific to certain Western European nations.
- Work is being done to make the User model properly extendable

## Gender Issues

Code samples at [djangoproject.com](http://djangoproject.com) are gender specific:

```
class Foo(models.Model):
    GENDER_CHOICES = (
        ('M', 'Male'),
        ('F', 'Female'),
    )
    gender = models.CharField(max_length=1, choices=GENDER_CHOICES)
```

Our examples should not get locked into examples from which people could feel excluded by because of personal life choices.

## 1.8.2 10 Steps to better postgresql performance

- Christophe Pettus
- PostgreSQL guy
- Done PostgreSQL for over 10 years
- Django for 4 years
- Not going to explain why things work great, just will provide good options. Ask him later for details
- <http://thebuild.com/presentations/not-your-job.pdf>

---

**Note:** Christophe talks super fast and I can't keep up

---

## PostgreSQL features

- Robust, feature-rich, fully ACID compliant database
- Very high performance, can handle hundreds of terabytes
- Default database with Django

## PostgreSQL negatives

- Configuration is hard
- Installation is hard on anything but Linux
- Not NoSQL

## Configuration

### Logging

- Be generous with logging; it's very low-impact on the system
- Locations for logs
  - syslog
  - standard format to files
  - Just paste the following:

```
log_destination = 'csvlog'
log_directory = 'pg_log'
TODO - get rest from Christophe
```

### Shared\_buffers

TODO - get this

### work\_mem

- Start low: 32-64MB
- Look for 'temporary file' lines in logs
- set to 2-3x the largest temp file you see
- Can cause a **huge** speed-up if set properly
- Be careful: it can use that amount of memory per query

### maintenance\_work\_mem

- Set to 10% of system memory, up to 1GB

### effective\_cache\_size

- Set to the amount of file system cache available
- If you don't know it, set it to 50% of the available memory

### Checkpointing

- A complete flush of dirty buffers to disk
- Potentially a lot of I/O
- Done when the first of two thresholds are hit:
  - A particular...

---

**Note:** Didn't get any of this part of things.

---

### Easy performance boosts

- Don't run anything else on your PostgreSQL server
- If PostgreSQL is in a VM, remember all of the other VMs on the same host
- Disable the Linux OOM killer

### Stupid Database Tricks

- Don't put your sessions in the database
- Avoid constantly-updated accumulator records.
- Don't put the task queues in the database
- Don't use the database as a filesystem
- Don't use frequently-locked singleton records
- Don't use very long-running transactions
- Mixing transactional and data warehouse queries on the same database

### One schema trick

- If one model has a constantly-updated section and a rarely-updated section
  - last-seen on site field
  - cut out that field into a new model

### SQL Pathologies

- Gigantic IN clauses (a typical Django anti-pattern) are problematic
- Unanchored text queries like '%this%' run slow

### Indexing

- A good index
  - Has high selectivity on commonly-used data
  - Returns a small number of records
  - Is determined by analysis, not guessing
- Use `pg_stat_user_tables` - shows sequential scans
- Use `pg_stat_index_blah`

## Vacuuming

- autovacuum slowing the system down?
  - increase `autovacuum_vacuum_cost_limit` in small increments
- Or if the load is periodic
  - Do manual VACUUMing instead at low-low times
  - You **must** VACUUM on a regular basis
- Analyze your vacuum
  - Collect statistics on the data to help the planner choose a good plan
  - Done automatically as part of autovacuum

## On-going maintenance

keeping it running

### monitoring

- Keep track of disk space and system load
- memory and I/O utilization is very handy
- 1 minute bnts
- `check_posgres.pl` at [bucardo.org](http://bucardo.org)

## Backups

### pg\_dump

- Easiest backup tool for PostgreSQL
- Low impact on a running database
- Makes a copy of the database
- becomes impractical for large databases

## Streaming replication

- Best solution for large databases
- Easy to set up
- Maintains an exact logical copy of the database on a different host
- Does not guard against application-level failures, however
- Can be used for read-only queries
- if you are getting query cancellations then bump up a config
- Is all-or-nothing



- If you need partial replication, you need to use Slony or Bucardo
  - ..warning:: partial replication is a full-time effort

## WAL Archiving

- Maintains a set of base backups and WAL segments on a remote server
- Can be used for point-in-time recovery in case of an application (or DBA) failure
- Slightly more complex to set up

## Encodings

- Character encoding is fixed in a database when created
- The defaults are not what you want
- Use UTF-8 encoding

## Migrations

- All modifications to a table take an exclusive lock on that table while the modification is being done.
- If you add a column with a default value, the table will be rewritten
- Migrating a big table
  - Create the column as NOT NULL
  - Add constraint later once field is populated

---

**Note:** I've done this a lot.

---

## Vacuum FREEZE

- Once in a while PostgreSQL needs to scan every table
- This can be a very big surprise
- Run VACUUM manually periodically

## Hardware

- Get lots of ECC RAM
- CPU is not as vital as RAM
- Use a RAID

## AWS Survival Guide

- Biggest instance you can afford
- EBS for the data and transaction
- **Set up streaming replication**

### 1.8.3 Round pegs for square holes - using mongoDB with Django

- Audrey Roy and Daniel Greefeld
- Cartweel people
- Using mongoDB with Django
- Taken by @chrisglass

---

**Note:** Hard to keep up here

---

## MongoDB

- Mongo is NoSQL, stores stuff in BSON, uses javascript (V8), bindings for pretty much anything available.
- Collections are like tables, Documents are like rows.
- Queries return a list of dictionaries.

## Many options

### pymongo

- Plenty of connectors available, pymongo being the “official” one, that most others wrap.
- Schemaless, very fast, supported directly by 10Gen.
- You lose modelforms, some admin

## MongoEngine

- Mongoengine is another option. It is a more Django looking piece of code, Integrates better with all the django bells and whistles. VERY FAST development (basically Another import instead of django.model) A con is that it's very close to the normal way of having schemas, which is counter intuitive in a schemaless DB, and you lose the django admin layer.

## MongoKit

- MongoKit: Makes queries a little less Django-like, more like MongoDB, therefore easier to go “schemaless”. A little slower, but admittedly no benchmarks <not sure it matters anyway compared to the DB-server roundtrip>

## Django-nonrel

- Django-nonrel with a mongodb backend: “a patch to django”, it’s actually a fork of django. You can use django as you would normally, still lagging behind The rest of django, multi-db is confusing

## Conclusions and further thoughts

- Django doesn’t really feels like a good match for NoSQL, and better suited for relational DBs
- Mongo is a greate DB, but the is some work to be done to simplify usage of Mongo, “lack of ai simple bridge”.
- If you have a schema definition anyway (models), why should you not use postgres and reap all the good stuff people wrote?
- Treat instrospection like MongoDB queries? To investigate
- Schemaless databases bring great advantages on the other hand - it is should be worth a few compromises.

## 1.8.4 Square pegs and round holes - Django and MongoDB

BY DANIEL GREENFELD AND AUDREY ROY

---

**Note:** Obviously not taken by Daniel as he’s talking. This version by Marc Tamlyn (@mjtamlyn).

---

- Danny might cartwheel...
- Work at cartwheel web - a django consulting shop. Met at Pycon 2010. Engaged!

## What is MongoDB?

- NoSQL
- Fast, indexable...
- Schema-less
- C++, Uses BSON (extended JSON), JS internals, Bindings in EVERYTHING. There’s a big community.

## Analogies

- Collections ~ Table
- Document ~ Row
- A QS looks like a list of dictionaries.

```
collection = []
document = {
    '_object_id': ObjectId('sasdfasdfsa'),
    'name': 'PyDanny'
}
collection = [document,]
```

## Connectors

- pymongo (low level)
- mongoengine/mongokit (Document ORM)
- Django non-rel

## PyMongo

- Official binding.
- powers everything else
- low level, but nice enough api.

```
connection = pymongo.Connection()
db = connection.db

for review in db.reviews.find({'rating': 3}):
    review['title']
```

- FAST!
- Schema crazy! (each row has its own schema)
- Supported directly by 10gen who make Mongo. Their recommended solution.

## Cons

- Very low level.
- Lose all of the things from Django you want.
- Syntax not so clear.

## Mongoengine

By @harrymarr!

Looks a lot like the Django ORM.

```
class Review(mongoengine.Document):
    name = mongoengine.CharField()
    ...
```

- Queries like the Django ORM.
- Super easy to develop with.
- Light schema, unenforced by db.
- django-mongonaught for admin-like functionality
- Supports *some* inter-document connections

## Cons

- Too structured?
- Validation messages sometimes unclear
- Lose on things like introspection (though that's what mongonaut is for)

## MongoKit

- Similar pattern to monogengine

```
class Review(Document):  
    structure = {  
        'title': unicode,  
        'body': unicode  
    }  
  
...
```

- Queries like mongo rather than Django. Much easier to make it schemaless.
- Pretty quick.
- Types are a mix of python & mongo.
- Losing the introspection again. No schema to inspect!

## Django non-rel + monogdbengine

- Adds NoSQL to the ORM. A Fork of django.
- Works with App Engine, MongoDB, and SQL dbs.

## Pros

- Exactly like normal django
- Has introspection from djangotoolbox

## Cons

- Forks ALL of django. (1.3...). Maintenance headache potentially.
- Multidb usage is confusing
- A bit idealistic...

## Summary

- pymongo is low level
- monogengine is schemaless django models
- mongokit ~ pymongo++

- django-nonrel is a django fork

### Thoughts: Danny

- Can we build a “simple” bridge?
- What about a 3rd party app which combines standard django apps with mongo db? (e.g. contrib.auth, forms, social-auth etc)
- “Let’s extend the django admin” doesn’t work...

Why add schemas to schemaless when:

- Relational DBs
- South
- High level caching tools

allow you to do fast moving dbs easily.

### Introspection tool idea:

Immediate introspection: if there’s no title then don’t show a title! Treat it like MongoDB queries.

### Thoughts: Audrey

- Schemaless dbs promise performance at the expense of ACID. Lose the guarantees for the higher availability.
- This is OK when performance is more important than being consistent 100% of the time.
- Schemaless models != schemaless collections. MongoEngine is best case unless you need *schema anarchy*! (Props to @harrymarr again)

### Using Django with Mongo

- Big hurdles, but it’s improving rapidly.
- Needs:
  - New tools
  - forms bridge
  - admin bridge
  - replacement for auth
  - creation of best practices
- Nothing wrong with mixing DBs.

### Django mongonaut

Introspection for MongoEngine. Works so far. Want to make it independent from mongoengine and make more generally useful.

Integrate some graphing tools? (e.g. graphviz) Should be based off immediate introspection rather than ahead-of-time.

## Summary

Consider all of the tools. It's not impossible!

### 1.8.5 Class-based Generic Views: patterns and anti-patterns

- BY BRUNO RENÉ
- CBVs added in Django 1.3
- <https://speakerdeck.com/u/brutasse/p/class-based-views-patterns-and-anti-patterns>

---

**Note:** couldn't keep up with his code samples

---

## Controversy

Blog posts last week

- Luke Plant
- Nick Coghlan

## What are views in Django?

“A view is a callable that takes a requests and returns a response”

## Deprecation

- Functional views are not deprecated
- Generic functional views are

## Pre Django-1.3 Django CBVs

- Admin
- RSS feeds
- Sitemaps

## CBV API

```
class View(Object):

    @classmethod
    def as_view(cls, **init):
        def view(request, *args, **kwargs):
            self = cls(**init)
            return self.dispatch(request, *args, **kwargs)

        # TODO find the rest
```

## Declarative vs Imperative

- CBVs have a much steeper learning curve
- [ccbv.co.uk](http://ccbv.co.uk) is a handy resource

## Usage Tips for Django CBVs

- try to keep `urls.py` for URL definition and nothing else
- Keep decorators in `views.py`

## Decorating

```
# TODO show Python 2.7 version here

class MyView(generic.ListView):
    pass
# Complete this
```

## MRO, extend, don't override

Unless you're 100% sure of what you're doing

## Case Studies

Useful recipes

## Form processing

TODO - get the form processing example

## Nested Navigation

# TODO - get example

## Pagination

# TODO - get example

## Registration

```
from le_social.registration import views

class Register(views.Register):
    form_class = blah
```



```
# TODO get example
```

## Settings

Don't set so many settings:

```
from le_social.registration import views

class Activate(views.Activate):
    expires_in = 3600 * 24 * 7 # 7 days
```

## Shooting yourself in the foot

The problems with using CBVs

## The 500 Error

```
class Handler500(generic.TemplateView):
    template_name = '500.html'
```

No matter what goes into this, it will throw out blank pages.

## 1.8.6 Django and the Real-Time Web

- by Zachary Voase
- <https://speakerdeck.com/u/zacharyvoase/p/django-and-the-real-time-web>

---

**Note:** Very thoughtful talk. Zachary scored some very critical points.

---

## WWW: Changelog

- Since July 2008 Chrome has stolen the market from IE.
- Chrome is about to take over IE in the desktop.
- JavaScript and long polling has come around.

## Can't miss this opportunity

But If we spend all your time playing with bright and shiny we'll lose our existing customer base.

## Zachary's definition of Real-Time

- UI before technology
- Proactive, not reactive
- Synchronized with the 'real world'.

## Stories

### MVC

- Parc 1978-1979
- Originally part of Smalltalk-80
- Now the dominant UI design pattern

How it works on the web:

- Listen for requests
  - Load session state for this user
  - Persist session state, clean up objects

### REST

- **RE**presentational **S**tate **T**ransfer
- Roy T Fielding (2000)
- Descriptive, not prescriptive
- Constraints
  - Client-server
  - Stateless
  - Cacheable
  - Layered
  - Code-on-demand (optional)
  - Uniform

### WebSockets

- Real TCP connection
- Magic HTTP request to port 80
- Reduces latency
- Enables real-time push

## REST & WebSockets

- Full-duplex communication
- Long-running connections
- Direct TCP connection
- TODO missed

## Version Control

- Centralized VCS (CVS, SVN, etc)
- Distributed VCS (Git, Mercurial, etc)

## Summary of the stores

- Imagine a system where the client is a full MVC stack, as is the server.
- Content is a matter of pushing/pulling like with DVCS
- Backbone.js does this, as well as Cappucino

## Caching

- Read RFC 2616
- Seriously.
- Etags
- Cache-Control
- Conflict resolution:

`(uri, etag, dirty?)`

## Implications

- Assumption of orthogonality
- Lossless representations
- Authn & Authz are hard topics
- Pub/sub
- Resource-oriented client

## Pub/Sub

- AMQP
- 0MQ
- Django Signals

## Barriers

- Django ORM - be opinionated!
- Content Negotiation
  - Don't have a separate API app
  - Created separately from the standard architecture
  - This is a good use case for Django CBVs
- JavaScript
- proxies and middleware

### 1.8.7 Building secure Django websites

- by Erik Romijn
  - [hello@solidlinks.nl](mailto:hello@solidlinks.nl)
  - <http://twitter.com/erikpub>
  - slides: <https://speakerdeck.com/u/erik/p/building-secure-django-websites>

## Three Areas

- Integrity
  - Internal consistency or lack of corruption in electronic data
- Confidential
  - To keep data secret that was intended to be security
- Available
  - ability to be used or obtained

## How cookies and sessions work

```
Set-cookie: name: value
Cookie: name=value
```

## Sessions

```
sessionId=8f70xxxxa3d9
session: {
  key: 8f70xxxxa3d9,
  user: Erik
}
```

If you can access the session of another user, you can impersonate the other user.

## Cross Site Request Forging

Fortunately for us, if you use POST, Django by default has CSRF protection enabled via:

```
<form>
    {% csrf_token %}
    ...
</form>
```

## XSS Injection

Injecting HTML or JavaScript into things like field data

```
<p>Injecting issues <script>alert("I'm a JavaScript injection!");</script></p>
```

## Reflected vs. Stored XSS

- Previous examples are reflected XSS
  - Have to try the user into visiting my link
- Other possibility is stored XSS
  - Store some data which is later sent back to users, e.g. blog comments

## Cookie security

- HTTPOnly flag will prevent reading cookie from JS
- Alternate attack is Cross Site Tracing (XST): disable TRACE on your web server
- Note: if cookie domain is set to e.g. djangocon, every website under djangocon.eu is at risk.

## Server side injections

### SQL injection

- No concern, Django ORM prevents injection
- If you don't use it, stick to prepared statements

### LDAP Injections

- You can play creative games if you know the LDAP specification

---

**Note:** I saw this at NASA HQ before we rolled out my first professional Python application back in 2006.

---

## Shell Injection

- Always use `subprocess`

## Trusting the Browser

- The browser is under the user's control
- Which means you cannot trust anything that the user is doing

## Be careful with ModelForms

Don't use the *exclude* Meta attribute in ModelForms!

```
class Profile(models.Model):
    user = ForeignKey(User)
    phone = models.CharField()
    is_admin = BooleanField() # added later

class ProfileForm(ModelForm):
    model = Profile
    exclude = ('user', )
```

```
<form>
    {{ form.non_field_errors }}
    Phone: {{ form.phone }}
</form>
```

## Passwords and SSL

- Don't use plaintext passwords
- Limit the number of attempts (django-axes, django-lockout)
- If you use logins, use SSL
- If you use SSL, look at django-secure

## Clickjacking and Django

- Protection in Django 1.4
- `django.middleware.clickjacking`
- etc

## Backups

- Run backups
- If you don't have backups, who owns your stuff?
- Test your restores!

## Introducing PLY

- PLY is an implementation of lex and yacc for Python
- Made by David Beazley

- <http://www.dabeaz.com/ply/>
- Naming conventions and introspection => very “economic” code

Let’s us compile things like:

```
groups name="XXX" AND NOT groups__name="YYY"
(modified > 1/4/2011 OR NOT state__name="OK") AND groups__name=="XXX"
```

into `django.db.models.Q` objects

## 1.8.8 Implementing DSLs in Django Apps

- by Mattieu Amiguet

### Initial Motivation: Searching Contacts

```
class Contact(models.Model):
    first_name = models.CharField()
    groups = models.ManyToManyField('Group')
```

- Client wants to customize things themselves
- giving them access to code is dangerous
- Limit their actions

### Other reasons

- Quick and easy to implement (if you use the right tools)
- Fun to code!

### Not for the end user

- Only to be used by power users
- Your DSL could be used as a scripting language

### How to make a DSL in Python/Django

- Basics
  - Lexer (vocabulary)
  - Parser (grammar)
  - Some kind of backend
- The lexer and parser part are quite generic
  - use code generator

## Sample

```
import ply.lex as lex

tokens = (
    'COMPA', # comparison operator
    'STRING',
    'NUMBER'
)

t_COMPA = r'=[<>]=?|~?'

literals = '()' # shortcut for 1-character functions

def t_STRING(t):
    r'"[^"]*"'
    t.value = t.value[1:-1]

def t_NUMBER(t):
    r'\d+'
    # TODO - finish this function
```

---

**Note:** Not sure how this works. Me need to read up on PLY

---

## Parser - The Grammar

```
expression : expression B_OP expression expression : U_OP expression expression : '(' expression ')'
value : STRING
        NUMBER
        DATE
```

## Parser in PLY

- Grammar rules go into docstrings
- Special argument p corresponds to rule parts

```
def p_expression_u_op(p):
    '''expression : U_OP expression'''
    if p[1] == 'NOT':
        p[0] = p[1]
```

### 1.8.9 I hate your database

by Andrew Godwin

- Lead developer for South <http://south.aeracode.org/>
- Cheese fan
- <http://twitter.com/andrewgodwin>
- Slides: <http://www.aeracode.org/static/slides/djangocon-eu-2012.pdf>



## Hate? Databases?

### Countering

- Misuse
- Ignorance
- Lies

## Different Databases, different occasions

- People use the same database for everything they touch
- You shouldn't use a database for things it was not designed to do.
- Types of databases:
  - Relational
  - Document
  - Key-value
  - Graph
  - Object / Heirarchial
  - Spatial
  - Time-series / RRD
  - Search
- Relational
  - PostgreSQL
  - MySQL
  - SQLite
- Document
  - MongoDB
  - CouchDB
- Key-value
  - Redis
  - Riak

## Some quick theory

- ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- CAP Theorem (you can only have 2 of the 3 of them)
  - Consistency
  - Availability
  - Partition Tolerance

## MySQL

Very interesting database system

- No transactional DDL
- Poor query optimizer
- MyISAM: full-table locking, no transactions
- Oracle
- Very fast for some operations

## SQLite

- Little integrity checking
- Impossible to do some table alterations
- No concurrent access
- Low overhead
- Very portable

## PostgreSQL

- Slow default configuration
- Can be a little harder to learn
- Almost too many features
- Incredibly reliable

## MongoDB

- No fixed schema
- Very similar to Python types
- Immature (but improving)
- No transactions
- No integrity checking

## Key/value stores

Redis, Riak, memcached

- Horizontal scaling (but with drawbacks)
- Extremely fast
- Can only fetch objects by key
- Batch/map-reduce queries
- Transactions not possible

## Spatial databases

- Knowledge of projections useful
- Spatial indexes really speed up some problems
- Generally add-on to existing DB

## Filesystems

- Hierarchal key-value store
- Allows multiple writers for appends
- Supports very large files

## Graph Databases

- Allow efficient neighbor queries
- Generally not much use for anything else

## Round-Robin Database

- Deliberately loses old data
- Useful for logging or statistics

## Summary

- It's unlikely your data all fits in one paradigm
- Just buying bigger servers goes a long way to overcoming shortcomings of a particular database
- Try multiple things before making a decision. Educate yourself!

### 1.8.10 LFS - Lightning Fast Shop

<http://www.getlfs.com/>

by Kai Diefenbach

- Living in Germany
- Does Python and Django
- Lead on LFS
- <http://diefenba.ch/>
- <http://twitter.com/diefenbach>
- Slides: <https://speakerdeck.com/u/diefenbach/p/lfs-an-online-shop-based-on-django>

#### LF = Lightning Fast WTF?

- Faster than the old Plone shop they used to support
- Calculation ~200ms / ~100ms
- Page 0.5 - 2 seconds
- Renders pretty fast

#### Numbers & Facts

- Django
- JQuery
- BSD license
- 100,000 downloads on PyPI
- Google Group > 170 members
- 9 committers
- On github
- ~40 known shops

#### Samples

- <http://demmelhuber.net>
- <https://www.anwaltakademie.de/>
- <http://www.holzimgarten.de/>

#### Features

- Custom management interface
- default theme is attractive
- Products can have variants

- Downloadable products coming soon!
- Topsellers
- Vouchers
- SEO
- Sitemaps
- Prepared for Google Analytics
- Good URL patterns
- Portlets
- Filtered navigation

## Properties

- Extend products
- Create variants & configurable products
- Filtering
- Select field, float field, and text field

## Accessories

- Lets you tack accessory products to a product
  - Roofs could list nails and tiles
  - MBA could list Sublime Text 2

## Variable Payment Methods

- Credit
- Debit
- PayPal
- Pluggable so we can add Stripe

## Development

- On Github
- PEP8 & pyflakes
- Every new feature must have a real live use case
- Every new feature must have tests
- Every new feature must have documentation
- Using Jenkins for CI
- Deprecations over two releases

## 1.8.11 Using CSS preprocessors effectively

by Jonas Wagner

- Known for doing crazy and creative stuff
- Porting Physic engines to JavaScript via Python
- Works as Software Engineer at local.ch
- 3.1 million unique clients

### Don't make a mess

- Most programming languages encourage good code pattern
- CSS is not one of those languages

### Issues with CSS

- No Variables
- No hierarchy
- Prefixes
- Sprites
- Lack of abstraction

### Solution: CSS Preprocessors

Choosing a Preprocessor

#### SASS

- Official implementation is in Ruby
- Two dialects scss and sass
- Sassy CSS
- Syntactically Awesome Stylesheets
- PySCSS is an implementaton of SCSS in Python
- Compass is

#### LESS

- Originally written in Ruby
- Now implemented using JavaScript in Node.js
- Can be compiled on the client and using Node.js
- Twitter bootstrap uses LESS

## Which one?

- Features virtually equivalent
- Both are a superset of CSS
- He recommends SCSS
  - More explicit syntax
  - Compass offers lots of goodies
  - Decent Python implementation

## Common Features

- Variables
- math and functions
- nesting
- avoiding CSS hacks
- More elegant comment system
- Mixins
- Prefixes

## Doing it with Django

```
pip install webassets cssmin
```

```
STATICFILES_FINDERS = (
    ...
    'django_assets.finders.AssetsFinder',
)

INSTALLED_APPS = (
    ...,
    'django_assets',
)
```

in assets.py:

```
from django_assets import Bundle, register
js = Bundle('common/jquery.js', 'site/base.js', 'site/widgets.js',
            filters='jsmin', output='gen/packed.js')
register('js_all', js)
```

```
{% load assets %}
{% assets "js_all" %}
<script type="text/javascript" src="{ ASSET_URL }"></script>
{% endassets %}
```

## Tools

- Good editor support for Preprocessors
- Graphical tools like LiveReload and Compass.app
- FireSASS

## Warning

- Increased complexity
- Might not work with IE
- Makes debugging harder
- Potential for bloat

## Conclusion

- Preprocessors solve common problems
- Allow us to focus on writing clear and meaningful CSS
- Try it on at least one project
- Plain old CSS feels very silly

## 1.8.12 Arkestra: semantic information publishing for organizations

by Daniele Procida

- Works at Cardiff University School of Medicine
- <http://medicine.cf.ac.uk> is his main site
- <http://arkestra-project.org/>
- <http://readthedocs.org/docs/arkestra/>
- slides: <https://speakerdeck.com/u/evildmp/p/arkestra-at-djangocon-europe-2012>

---

**Note:** Good talk but some slides had too many bullets.

---

## What typically happens when working with a CMS

- You have to repeat yourself
- data gets wasted and lost
- content & presentation becomes inconsistent
- info in templates gets broken and petrifies
- information ages, withers & dies
- users play fast & loose
- The larger the site the worse the problems get



## His idea

Create a model of the real world

## Information, not just data

- information not useless stupid data
- templates don't hold information
- Semantic modeling real-world relationships
- user semantics!

## Organization

- Can you model a CMS based off an organization?
- He created a concept of `entity`
- Many interconnections of content and data
- He did it on <http://medicine.cf.ac.uk>
  - Entities are associated with pages
  - Not entity needs to have a page
- We did a very similar effort on <http://science.nasa.gov/>, but...
  - not so well organized.
  - grown organically during the course of a number of badly run meetings

## Don't waste people's time

- Make everything re-usable and re-use it
- Make it easier to re-use than re-enter

## Django-CMS and Arkestra

- Django CMS & Arkestra grew up together
- have been developed alongside each other
- portions of Django CMS conceived as part of Arkestra
- integration with pages, placeholders/plugins, menus

## The Semantic Presentation Editor

The problem:

- Create complex, flexible, multiple-column layouts
- produce well-structured semantic HTML
- Need no HTML/CSS skills

Solution:

- The Semantic Presentation Editor
- Special text editor that renders out things in a lovely, semantic fashion
- See <https://bitbucket.org/spookylukey/semanticeditor/wiki/Home>

### 1.8.13 Django Chuck - Your powerful project punch button

by Bastian Ballmann and Lukas Bünger

---

**Note:** Looks to be an amazing, modular tool for standing up projects easily. Missed most of the talk so my notes are incomplete.

---

#### Why the name Chuck?

- Chuck is not informal term for meal
- Not meaning vomit
- Chuck has no times to anything

#### Use case for django-chuck

- Same setup all the time
- Manual project setups
- Same conditions apply all the time

#### Why not Pinax?

- No modular template structure or code base
- Monolithic Python script
- Addresses project creation only
- No flexible build process management

#### Installation

```
pip install django-chuck
```

copy example\_conf.py to:

~/django\_chuck\_conf.py

See [django-chuck.rtd.org](http://django-chuck.rtd.org)

## Example usage

```
chuck create_project <prefix> <name> [modules] -a [pip modules]
chuck create_project ni djangocon django-cms,test,nginx
```

## What happened?

This got generated:

- settings
- requirements
- uwsgi
- fabfile
- hosting
- jenkins
- templates

## Setup an existing project from source

```
chuck setup_project git@whatever.com:your-project.git
```

---

**Note:** Stepped away for things.

---

### 1.8.14 It's about time

by Aymeric Augustin

- Django core dev
- [http://static.myks.org/data/20120605-DjangoCon-It's\\_about\\_time.pdf](http://static.myks.org/data/20120605-DjangoCon-It's_about_time.pdf)

---

**Note:** Had to deal with a business thing so didn't get all of Aymeric's talk down. What I got was from some really awesome material.

---

## RFC 3339

- Current era
- Stated offset
- universal time
- instant in time

```
from datetime import datetime
datetime(
    year=2012, month=6, day=5
    hour=16, minute=10, second=0,
    microsecond=0,
    tzinfo=FixedOffset(120)
)
```

Time zones add complexity

### aware vs naive datetimes

```
>>> naive = datetime(2012, 6, 5, 16, 15)
>>> tz = timezone("Europe/Paris")
>>> aware = tz.localize(naive)
>>> naive - aware
```

From the Python docs: Whether a naive datetime object represents UTC, local time, or time in some other timezone is purely up to the program.

### DST transitions

```
from datetime import datetime
```

### dates and datetimes

- dates are always naive
- they don't suffer from the same problems as naive datetimes
- using an aware datetime as a date is an accident waiting to happen
- Django supports mixing naive datetimes and dates

### Django >= 1.4

- Uses aware datetimes in URC internally
- stores naive datetime sin UTC in the database (except PostgreSQL)
- converts to aware datetimes in local time in forms and templates
- supports multiple time zones!

### default and current time zones

- *default = settings.TIME\_ZONE*
  - used in models for conversions between naive and aware objects
- *current = end user's time zone*
  - used in templates and forms
  - for multiple time zones support

## auto-conversions

- ensure backwards compatibility
- avoid surprises for single time zone sites
- but support sloppy constructs e.g.,
  - filter a DateTimeField with a date
  - save a datetime in a DateField

## Utilities

```
>>> from django.conf import settings
>>> from django.utils import timezone

>>> settings.USE_TZ = True
>>> timezone.now()
<snip>
```

## limitations in Django 1.4

- The database works in UTC (ticket #17260)
- QuerySet.dates()
  - \_\_year/month/day/week\_day
- Author of pluggable apps may have to handle two cases

## Key learnings

1. A datetime is a point in time. A date is a calendaring concept.
2. Use aware datetimes in UTC and convert to local time for humans.
3. learn how to use pytz properly especially *localize* and *normalize*

Time isn't as simple as it seems. Learn and practice!

## 1.8.15 Healthy Webapps Through Continuous Introspection

by Erik van Zijst

- <http://twitter.com/erikvanzijst>
- <https://bitbucket.org/evzijst>
- Slides: <http://t.co/V0rHYjIu>

## Case study: Wasted cycles on Bitbucket

=> SSHD => conq (Python) => git/hg

- conq is our custom SSH shell

- conq imports Django ORM and Bitbucket code
- takes ~1.41 seconds to start (spawns ~50/second)

### Solution after analysis: Stop the imports and just write native SQL

- 16 times faster to start up (0.09s vs 1.41s)
- 60% load decrease on all web servers!

### Lessons learned

- Test your stuff
- Monitor your servers

### Common problems

#### Slowness in Web Apps

- Slow SQL queries (or too many!)
- lock contention
  - between threads
  - database table/row locks
  - fine locks (git/hg)
- excessive IO (disk/network)
- evil regex: `r ' ^ (a+) +$ '`

#### consequences

- 503 - worker pools full
- 500 if requests time out (Gunicorn SIGKILL)

The latter is best avoided as it destroys forensic evidence and leaves stale state (e.g. lock files)

#### Dogslow

- Django middleware
- emails tracebacks of slow requests
- no performance penalty, safe on prod
- <https://bitbucket.org/evzijst/dogslow>

## django-geordi

Designed to profile your production environment without impacting performance

- selectively profile individual requests
- add “?\_\_geordi\_\_” to any URL
- products PDF call graph
- <https://bitbucket.org/brodie/geordi>

## interruptingcow

Designed to let you catch and then bubble up a system locking issue

```
import re
from interruptingcow import timeout

try:
    with timeout(20.0, RuntimeError):
        #evil regex
        re.match(r'^(a+)+$', 'aaaaaaaaaaaa')
except RuntimeError:
    print 'Interrupted'
```

### 1.8.16 Adding tests to an uncovered application

by Zach Smith

- Slides: <https://speakerdeck.com/u/zmsmith/p/adding-tests-to-an-uncovered-app>

## Instagram started as burbn

- Instagram started as a HTML5 version of 4square
- Pivoted
- If they had written tests those tests were wasted

## When to automate testing?

- Think about time: can you spend the time to write these tests?

## Types of Tests

- Acceptance tests
- Functional tests
- Unit tests

### Libraries to help increase test coverage

- lettuce
- sprinter

## 1.8.17 Implementing real time web apps with Django

by Kristian Ollegaard

- Works at Divio
- Django since 0.96
- Danish, but lived in Zurich for 1.5 years
- <http://kristian.io>
- <http://twitter.com/oellegaard>
- slides: <http://www.slideshare.net/oellegaard/implementing-real-time-web-applications-with-django>

### Why real time?

- Better user experience
- More options in front end
- Make the web feel like native apps
- Showing live data
- Collaboration is much easier

### Finding the right tool

- Criterias
  - Use websockets but has fallbacks
  - Good browser support including old IE
  - Should be usable from Python
  - Does not require extensive changes in frontend
  - As fast as it can be

### The tools you want

- Node.js
  - Built on Chrome's JavaScript runtime
  - Uses an event-driven non-blocking I/O model
- Socket.io
  - one interface for all transport methods (sockets, polling, etc)
  - Compatible with almost everything



## Why not implement it in Python?

- Already active community
- Can be used from python without too much trouble
- Most people know very basic javascript
- More importantly, frontend engineers know javascript and can therefore contribute to the different browser-specific implementations.

## Using redis for cross-language communications

- Support for many datatypes
- Can be used both as storage and as a queue
- Implemented in many different languages
- For the usage in this talk, any other queue could have been used as well

## Basic Concept

- Something happens, the user must be notified in real time
  - From Django we insert the new value into the queue
  - Node.js listens on the queue

```
var io = require('socket.io').listen(8001);
var redis = require('redis').createClient();
redis.psubscribe("socketio_*");

// TODO add the rest
```

```
<!-- Add this part -->
```

```
import redis
import json
redis_subscribe = redis.StrictRedis()
redis_subscribe.publish('socketio_news',
                        json.dumps("Hey, how are you?"))
```

## Hosting socket.io

- Nginx does not support websockets!
- Needs its own app, if hosted on an application cloud (e.g. Heroku)
- Recommended to expose the node server directly
  - But hey, it's node.js, it scales!

## Using this today?

- Maybe not
- Do some research

## Client Authentication

- Socket.io handles authentication from node -> client
- Currently no authentication between django and node
- Could possibly be solved by storing your sessions in redis and checking them between systems

## 1.8.18 How Heroku Uses Heroku To Build Heroku

by Craig Kerstiens

- Works at Heroku
- <http://twitter.com/craigkerstiens>

### What is Heroku?

- Platform as a Service (PaaS)
- focuses on developer productivity
- 4000 heroku apps
- 500+ releases a day
- 200+ deploys a day
- 105 public github repos
- 85 people across 21 teams
- a cloud unix

### Philosophies

- Do 1 thing and do it well
- Run and forget
- Defined Contract/API
- Developer Environments
- Environment Parity

### Do 1 thing and do it well

- Small functional apps
- KISSMetrics Data Loader
  - Open DB connection
  - Run query
  - Post data
- Others
  - OAuth

- Vault
- API
- Core

## Run and forget

- Start an app
- Let them run
- Forget about them
- Alert me when things break

## Sample standing up an app

```
git clone git://github.com/opencomparison/opencomparison.git
heroku create -s cedar
git push heroku master
```

## Environmental Parity

Dev and staging and production should be identical

```
DEFAULT_FROM_EMAIL = os.environ.get('DEFAULT_FROM_EMAIL', 'pydanny
↪<pydanny@cartwheelweb.com>')
AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
```

## More philosophies in use by Heroku

- ownership
- productivity
- agility

## Specifics

- “Let engineers be engineers”
- You choose the tools to get the job done, you support the API for others, you own the features to make users happier, you ensure it works, you carry the pager
- Broad focus around quality, quality comes from solid engineering, give time to engineers, get stuff done.
- Iterate fast and often, a failed attempt is a successful experiment, culture around seeing/doing over talking
- Github issues alone doesn’t fix communication

## Focus on Quality

- Make good art
- Hire for quality and culture
- Quality doesn't work with deadlines

## 1.8.19 Involving women in the community

by Lynn Root

- <http://twitter.com/roguelynn>
- Founder of the San Francisco chapter of pyladies (hundreds of members)
- Event coordinator for Women learning to code

## New Developer

- Started last fall
- Ran into problems with signing up for classes because of her last name
- Comes from a financial background
- correlation vs causation
  - <http://xkcd.com/552/>

## Opened the floor to questions and discussion

“I don't think we should care that much, because if we care too much then the problem gets bigger.”

---

**Note:** Cause if you ignore the problem, it goes away, right? This is a really bad path to take.

---

Paraphrasing: “Sauna statements with mention of female body parts.”

---

**Note:** WHAT THE FUCK?!?

---

- “Hire anyone because they are smart, not because of gender.”

---

**Note:** YES! If you are having trouble finding developers, hire smart people regardless of their race, gender, creed, etc and train them up.

---

- “I have a game: Any time someone says, ‘Women tend to’ or ‘Men tend to’, interrupt them and yell ‘BULL-SHIT!’”

---

**Note:** Brilliant! I'm going to play this game.

---

“Is this the right place or platform for this discussion? With so many people watching it's all too easy for me to screw this up and offend someone.”

---

**Note:** Good point. This is very, very hard. We've all made mistakes. I'm not sure what the answer is.

---

"For those of you wondering about if it's okay to sponsor women/minorities and give special treatment, it makes a huge difference in the lives of those who benefits. Including Audrey Roy"

---

**Note:** I agree. :-)

---

"Now is the time. Don't let this fade. Our ancestors fought hard for our rights, lets' keep up the cause"

---

**Note:** Simple summary of my statement

---

"Don't make sexist jokes"

---

**Note:** Awesome

---

## 1.8.20 Flasky Goodness (or Why Django Sucks?)

by Kenneth Reitz

- <http://twitter.com/kennethreitz>
- Works for Heroku

### Hos Open Source work

- <http://python-guide.org>
  - Documented best practices
  - Guidebook for newcomers
  - Reference for seasoned veterans
  - Don't panic & always carry a towel
- <http://python-requests.org>
  - HTTP for humans
- <http://httpbin.org>
- legit: Git workflow for Humans
- Envoy: Subprocess for Humans
- Tablib: Tabular Data for Humans
- Clint: CLI App Toolkit
- Autoenv: Magic Shell Environments
- OSX-GCC Installer: Provokes Lawyers

## Open Source All The Things!

- Components become concise and decoupled
- Concerns separate themselves
- Best practices emerge
- Documentation and tests become crucial
- Code can be released at any time
- Abstraction

## Let's build something

Why pick Django?

- Makes modular decisions for you
- Makes security decisions for you
- Excellent documentation
- Installable third-party Django apps
- Tremendous resources & Community
- much more cause anything is possible!

## Django Application

- Tools & Utilities
  - Management Tools
  - Supporting Services
- Web Process
  - User Interface
  - API
  - Data Persistence
  - CRUD Admin
  - Authentication
- Worker Process
  - Deferred Tasks
  - Scheduled Tasks

## Single Codebases are great

- All the benefits of the Django stack
- Figure out architecture as you go
- Shared modules keep you dry

- Make broad, sweeping changes
- Only need to deploy once

### Single codebases are EVIL!

- Tight coupling of components
- Broad tribal knowledge required
- Iterative change of components difficult
- Technical debt has a tendency to spread
- Forced to deploy everything at once.

Anything is possible... but that ends up with a monolithic application.

### CONSTRAINTS FOSTER CREATIVITY

- Having rules gives you an environment in which to play.
- Text Editors vs IDEs (Vim lets you do so much, Sublime Text limits what you can do)
- Prime vs Zoom Lenses
- Mac OS X vs Desktop Linux
- Pen/paper vs electronic notes

### Build for services

- Components become concise & decoupled
- Concerns separate themselves
- Best practices emerge
- Documentation and contracts become crucial
- \_\_\_\_\_

**Note:** missed some here

---

### Results in composability

#### Django: For API Services

- Significant boilerplate code for simple views
- No need to templates, tags, etc
- API Libraries are buggy; could use some love
- `if request.method == 'POST'`

## Django: For API Consumer

- Keep in mind, database is handled by the API
- Makes modular decisions for you
- Deals with the database for you
- Installable third-party Django apps

## Enter Flask

- HTTP Web Framework based on Werkzeug
- Excellent for building web services
- Elegant and simple

## Flask Familiarities

- WSGI Application Framework
- Jinja2
- activity community
- Started an April Fool's joke
- Just 800 lines of code
- Heavily tested, 1500 lines of tests
- Exhaustively document; 200 pages of docs
- Layered API; built on Werkzeug, WSGI

## Flask Differences

- Explicit & Passable app objects
- Simple, elegant API. No boiler player
- BYOB: Bring your own batteries
- No built-in ORM or form validation
- Context locals. Keeps things looking clean

## Flask Improvements

- Fewer batteries == greater flexibility
- Jinja2 is an incredible template system
- Everything harnesses actual references
- Configuration is a simple dictionary
- It's hard to build monolithic applications
- Response objects are WSGI applications



- Werkzeug debugger
- No import-time side effects
- Signals system outside of ORM
- Tests are simpler with real app objects
- `return (content, status)`

### Popular Flask Extensions

- Flask-SQLAlchemy: Database Mapper
- Flask-Celery: Delayed jobs
- Flask-Script: Management commands
- Flask-WTF: Form Validation

### Services are agnostic

- Just speak HTTP
- Use both Django and Flask

### 1.8.21 General Notes

- Location: Zurich, Switzerland

## 1.9 Pycon 2012

---

**Note:** Github was gracious enough to donate their booth to our startup, [Consumer Notebook](#). So instead of going to many awesome talks and taking notes, I spent most of the time at our booth. Tons of people came by and checked us out. On Sunday I learned just how painful standing for two days on concrete combined with feet pounding Capoeira action can be. ;)

---

### 1.9.1 Diversity in Practice

“How the Boston Python User Group grew to 1700 people and over 15% women”

- **by Jessica McKellar , Asheesh Laroia**
- Boston Python user group organizers
- PSF members
- PSF outreach and education committed members
- Open hatch
- FOSS

## Basics

- Diversity membership makes user groups **better**.
- Diversity outreach helps user groups group

## Motivation

- No women at user group events
- No pipeline for newcomers / beginners.
- To fix it, they decided to change things from within

## Goals

1. Bring more women into the community. Get to 15%
2. Show examples of great women programmers
3. Encourage other user groups to think about diversity

## Workshop goals

- workshops + follow up events
- Over 200 women alums
- Large volunteer base
- Beginner's stay inside!

## Schedule

How they do it:

### Friday: Spaces

- Windows and Python sucks.
- Fixing tabs versus spaces
- Practice with the interpreter
- All their materials are available on the web

### Saturday: Lecture and practice

- Basics of python objects and structures (2 hours)
- Lunch
- Build your own project (2 hours)
  - A couple games
  - Play with the twitter API

## Post event

- Hack nights
- Discussion groups

## The results

### Before

- 1 organizer (Ned Batchelder)
- 700 members

### After

- 3 organizers
- 1800 members
- Monthly lecture-style events
- hack nights
- classes
- more

## Reflection & Sharing

- Volunteers are awesome
- Why'd you sign up? "Women, judgement-free, free"
- Staff wrap-up. Lesson's learned:
  - More coding practice
  - simplified projects

## How they share the work

- Curriculum on wiki
- Share their code:
  - codingbat.com

## Scaling out: impact beyond Boston

### How they influenced the world

- Montreal Python: Women & friends workshop
- Pystar: Workshops and material
  - PyStar Philly

- PyLadies: women-friendly python user group(s)
- Ladies learning code: women-oriented tech programming in Toronto
- 7 of the 50 poster sessions came out of women who got involved because of these groups

### Next steps

- Continuing innovation of organization
- Get people in via workshops, user groups, PSF memberships,
- project nights

### Resources

- <http://bostonpython.org>

### Other

- Lessons from Railsbridge:

```
from sf.ruby
import railsbridge
```

## 1.9.2 Transifex: Beautiful Python app localization

by Dimitris Glezos

### Intro

- Github for translations
- Develop for an international audience

### Workflow

- Mark translatable strings
- Release string freeze
- Translator: VCS checkout
- Translate w/special tools
- Get 'em files back
  - SSH, email, tickets
- For every frigging release

## Python & Gettext

```
# TODO show Django import
from gettext import gettext as _

thing = _("I'm going to be translated")
```

```
{% load i18n %}

{% trans "person" %}

{% blocktrans count ppl|length as num %}
TODO show moar
```

- Generates PO files

## How to render PO files

TODO: show the command-line actions for pure Python and Django

## Traditional model of translations

- Content owner/developer
- Localization manager
- Content management system
- Developers

## Existing solutions before Transifex

- Emails to translation agencies
  - loss of control
  - Expensive
- Build own L10N system
  - Lots of work
  - Expensive

## Transifex

- SaaS product
- Open source platform
- Built for developers to maintain

## Simpler model of translations

- Content owner/developer
- Content Management System
- Transifex

## Transifex size of project

- 17,000 users
- 3,000 products

## Tech

- Django, Python, PostgreSQL, MongoDB, Redis
- Celery AMQP
- Django Add-ons
- Mercurial, Git

## Workflow automation

```
$ pip install transifex-client
$ tx set --auto-local -r myrproj.myres --source-lang en etc...
```

Creates a local .tx file that set up the configuration file. This can be uploaded to git.

```
# commands to interact with Transifex
$ tx pull --source
$ tx push --translations
```

## Workflow automation

- Continuous integration
- VCS commit hooks
- API to translate content
- Services on Github and Bitbucket
- Heroku Addon

## Nifty features

- Social interactivity, comes with a onboarded community

### 1.9.3 Python Web Summit March 8, 2012

- Hosted by
  - Michael Ryabushkin
  - Chris McDonough
- url: <https://us.pycon.org/2012/community/WebDevSummit/>

---

**Note:** Taking notes on a panel is really challenging. Apologies on whatever or to whoever I miss. Any misquotes are my fault and not the fault of the speaker.

---

#### Introduction

by Michael Ryabushkin

#### How did this start?

- People:

Chris McDonough Mike Orr Phil Jenvey Mike Bayer Danny Greenfeld Audrey Roy
---

#### During the summit

- Panels are important, but take people aside if need be

#### Creating a Better Deployment Story

Moderated by Jacob Kaplan-Moss (Django DBFL)

Panelists:

- Tarek Ziadé (Distribute/Packaging)
  - New distutils lets you specify versions of third-party packages. But... redhat and other OS tools have their own package names. Ugh.
- Nate Aune (DjangoZoom, Appsembler)
  - All tools (Plone, Django) makes it hard to do deployments. Hence his deployment startups (DjangoZoom and Appsembler).
  - We need to come up with a standard and insist on using it.
- Kenneth Reitz (Heroku)
  - pip needs to be able to set versions of Python
  - PyPI needs more attention

- Ian Bicking (Paste/WebOb/Silver Lining)
  - Let’s create a formal Application Package specification.
  - Problem to overcome is the difference between developers and sys admin
- Jim Fulton (Buildout/Zope)
  - Company does development and maintains 500 applications
  - “Packaging needs to be better so we can deploy more easily.”

## Questions

1. How come Java .war files do it better than Python?
2. How do we make it so that Python is as easy to deploy to the web as PHP.
3. What is the status quo?
4. What are people working on to make this better?

## Formal Application Package Specification?

Ian Bicking’s thought on how to do it. Would have these things:

```
Formal path specification
Formal dependency listings
Project description
Python version labeling
Include non-Python components (database, libraries, fortran, etc)
```

Jacob’s comment: Is this going down the route of Chef/Puppet?

## Namespacing Distributions

- Tarek as to play namespacing games to make sure that we get to use DistUtils backporting across versions of Python.
- Armin Ronacher commented: “*The fact that having the same package name for distutils2/packaging would be a problem shows the root of the issue: no proper version deps.*”

## Takeaways

- Force deployed applications be a package.
- Formal application specification?
- Specify Python versions in virtualenv/buildout
- The challenge of dealing with vendor named projects
- Jacob is going to publish a specification that will hopefully get the community moving. And he invites others to participate on his work.
  - The others pledged to help out or work on best practices.



## Porting to Python 3

### Moderated by Barry Warsaw (Canonical, Python FLUFL)

- Chris McDonough (Pyramid/WebOb)
- Armin Ronacher (Flask, Jinja2)
- Guido van Rossum (Python)
  - I don't know much about how you guys do Python web development and want to know so I can make it easier for Python 3 conversions.
  - There was artificial ambiguity introduced in Python 2 in regards to strings.
- Mike Bayer (SQLAlchemy/Mako)
  - The Python database API needs some love. I agree in that a huge, unmentioned hurdle for Python three are other libraries besides web frameworks and unicode. DBAPI, PIL, etc.
  - PEP 249 doesn't mention unicode. <http://www.python.org/dev/peps/pep-0249/>
- Robert Brewer (CherryPy)

### Questions

- Chris McDonough: Who runs web apps on Python 3?: crickets
- Barry: What are the big blockers
- Me: What about auxiliary library blockers like PIL, lxml, DB-API?
  - Answer: <http://stackoverflow.com/questions/3896286/image-library-for-python-3>
  - Answer: <https://github.com/gpolo/pil-py3k>
  - Answer: <https://github.com/sloonz/pil-py3k>
  - Answer: <http://www.lfd.uci.edu/~gohlke/pythonlibs/#pil>
  - Answer: <http://www.imagemagick.org/download/python/>
  - ---

**Note:** Pillow does not solve the Python 3 issue

---

- Dylan Jay: Why are library writers having to maintain two copies of their library code?

### Compelling arguments for/on Python 3

- Armin: Python 3 has some powerful features for sockets and other components that Python 2.x lacks.
- Barry Warsaw: Newer and more powerful libraries being written in Python 3.
- Wayne Witzel: Give a ton options for porting to python3, they won't choose any of them. Most people just want to be told what is right.
- Barry Warsaw: 2-to-3 tool is useful for getting started, but once in the weeds I find that I dive into the code.

## Final Thoughts

---

**Note:** my summary of their statements.

---

- Chris McDonough: We need to make people more enthusiastic about Python 3.
- Armin Ronacher: Improve the guides on porting.
- Guido Van Rossum: This will be resolved. It's going to be a while, but we can make it. We'll remember how hard it was to move forward to Python 3.
- Mike Bayer: This will be resolved when we think in Python 3 by default. And make Python 2.x a boring backport.
- Robert Brewer:

## Factoring Code for Reuse

Moderated by Danny Greenfeld ([consumernotebook.com](http://consumernotebook.com))

- Tres Seaver (Zope/CMF/Pyramid)
- Mariano Reingart (Web2Py)
- Alex Gaynor (Django/PyPy)
- Michael Foord (IronPython, Mock)
- Carl Meyer (Virtualenv, Pip)

## “State-Of” Multi-Talk Round 1

Each of these speakers, a leader in their field, gets time to talk about his subject.

## Graham Dumpleton (WSGI 2 ideas)

- PEP 333 was created back in 2003
- PEP 3333 was created back in 2010
- Wanted something better:
  - Make it simpler
  - standardized high level request/response objects
  - Async support (not possible because so different)
  - Resource management
  - Unknown request content length
    - \* no compressed request content
    - \* No chunked requests
    - \* no full duplex HTTP
- Has the boat sailed?

- Too much legacy code relying on WSGI 1.0
- Missed opportunity with Python 3
- Graham's ideas:
  - use context managers to improve resource management
    - \* need to override close() of the iterable
  - Implement wsgi.input as an iterable
  - TODO add what I missed

## Benoît Chesneau (gunicorn)

The state of gunicorn:

- First commit was November 30, 2009, Three users at first
- preform model
- Thread-safe
- Automatic worker process management
- Manage using signals
- Natively support WSGI, Django, Paster
- HTTP streams: decode on the fly http chunks
- Supporting sendfile & FileWrapper
- Simple Python configuration
- Multiple workers (sync and async)
- Various server hooks
- use your own logger
- Some issues:
  - Reload - graceful (HUP) or reexec (USR2)
  - The Django case: python-manage.py
  - Performance issues due to WSGI
  - CGI compatibility: headers
  - CONTENT\_TYPE,CONTENT\_LENGTH,SCRIPT\_NAME
  - The WSGI spec needs to be completed
  - Async workers & blocking issues
    - \* IO access like the filesystem are not greened
    - \* C drivers
  - Web app configurations & deployment: we need a standard
- Challenges:
  - Python 3
    - \* the case of sync workers

- \* Handle extensions/plugins
- New needs on the web: websockets, SPDY
- modular HTTP & WSGI server in Python

## Ben Bangert (Pylons Project)

State of the Pylons Project:

- Lots of community plugins developing
- Larger frameworks on top taking off along with bootstraps
- Ploneconf Pyramid track
- 62 reports, 31 devs with commit
- lots of `pyramid_*` packages

Challenges

- Porting to Python 3 (is doing very well)
- organizing and simplifying sometimes overly pedantic documentation
- TurboGears
  - Still exists
  - Some progress on community migration
- Standardizing deployment
- Foundation efforts
- Pylons conf (189 members of the SF Bay Pylons meetup)
- Increasing presence at conferences (not just Python ones)
- More awareness of professional support
- Moar books?

## Robert Brewer (CherryPy)

State of CherryPy

- It works in Python 2.7x
- It works in Python 3.x
- Being broken up into modular components so the WSGI HTTP server can be used in things like Pyramid

## Promoting Python for Web Use

Moderated by Paul Everitt (Pyramid)

- Steve Holden (PSF/DjangoCon)
- Liz Leddy (Plone/PloneConf)
- Eric Holscher ([Readthedocs.org](http://Readthedocs.org))

- Leah Culver ([Grove.io](http://Grove.io))
- Danny Greenfeld ([consumernotebook.com](http://consumernotebook.com))

## “State-Of” Multi-Talk - Round 2

### Glyph Lefkowitz (Twisted)

- It works for the web!
- Lots of cool features
- Works more as a container rather than a platform.
- Has excellent support for Windows. Has an MSI, Executable, etc
- Needs to fight the impression of being a giant library. It is actually small.

### Jannis Leidel (Django Project)

- Used in a lot of places around the world in small and gigantic projects
- 21,700 user list
- 7,000 developer list
- 33 committers
- 2,5000 downloads/day
- 2,100 projects on PyPI
- Django 1.4 almost out:
  - New project layout
  - Custom project templates
  - Standard WSGI entrypoint
  - Full timezone support
  - In browser testing
  - Cookie session backend
  - Clickjacking protection
  - New form wizard
  - i18n URLs
  - No exception wrapping templates anymore
  - `*args, **kwargs`

New orm functions:

```
select_for_update()  
bulk_create()  
distinct('filename')
```

New template stuff:

```
@assignment_tag
{% elif %}
{% static %}
{% truncatechars %}
```

Django 1.5 news:

- Cool stuff is coming

## Quotes

“Django isn’t a functional unit. You include it and it just sits there.”

## 1.9.4 Keynotes

### Stormy Peters

---

**Note:** I was working a project issue and couldn’t take notes. Sorry Stormy!

---

### Paul Graham

- PyCon is the center of Silicon Valley.
- His notes are online are at:
- The biggest startup ideas are frightening:
  - The threaten your identity
  - Think the John Malkovich room
- Let’s say you want to start the next google?
  - Microsoft tried and

## 1.9.5 Other Pycon Notes

Where I’m listing other people’s notes until I get a dedicated site up:

- William McVery, @wam, [https://www.dropbox.com/sh/i5jibofn60y14dt/rEs\\_RzYFxs](https://www.dropbox.com/sh/i5jibofn60y14dt/rEs_RzYFxs)

## 1.9.6 Other events I’ll be attending

- Saturday’s [Consumer Notebook](#) booth on [PyCon Startup Row](#).
- Saturday’s [Capoeira Roda](#)

## 1.10 Scale 10x

### 1.10.1 It's all good- Decorating Python like Martha Stewart

- by Matt Harrison
- [http://twitter.com/\\_mharrison\\_](http://twitter.com/_mharrison_)
- Works for <http://fusion.io>
- <http://hairysun.com/books/decorators/>
  - His talk is under creative commons

#### Impetus

You can get by in Python with basic constructs but...

- you might get bored
- be confused by other's code
- want more power

#### Function Review

A function is an instance of type function

```
>>> def spam():
...     "A function"
...     print 'eggs'
>>> spam
<function 0x2342342>
>>> callable(spam)
True
>>> spam()
'eggs'
```

Functions have attributes

```
>>> spam.func_name
'spam'
>>> spam.__doc__
"A function"
```

A function knows about itself

```
>>> def foo2():
...     print "NAME", foo2.func_name
```

A function can have attributes assigned:

```
>>> def foo3():
...     print "STUFF", foo3.stuff
>>> foo3.stuff = "Data"
>>> foo3()
Data
```

## Function Definition

```
def func_name(arg1, arg2=value, *args, **kwargs):  
    """ docstring """  
    # implementation
```

## Function Gotcha

When a function is created, the named/default parameters are defined when the function is created

```
def named_param(a, foo=[]):  
    if not poo:  
        foo.append(a)  
  
print named_param.func_defaults  
([])  
  
named_param(1)  
print named_param.func_defaults  
([1, ])
```

Lists and dicts are mutable. When you modify them you don't create a new list (or dict). Strings and ints are immutable

Parameters are evaluated when the def they belong to is imported

Don't default to mutable types.

```
def named_param(a, foo=None):  
    foo = foo or []  
    if not foo:  
        foo.append(a)
```

## \*args and \*\*kwargs

Looksee:

- **\*args** is a tuple of parameter values.
- **\*\*kwargs** is a dictionary of key/values

```
def param_func(a, b=2, c=5):  
    print [x for x in [a, b, c]]
```

The '\*' before args flattens the tuple of parameters values.

```
def param_func(a, *args):  
    print [x for x in [args]]  
    # TODO check I got this right  
  
def kwargs_func(a, **kwargs):  
    print [x for x in [kwargs]]  
    # TODO check I got this right  
  
def param_func(a, b='b', *args, **kwargs):  
    print [x for x in [a, b, args, kwargs]]
```



## Closures

- PEP 227 and came out in Python 2.1
- Don't be afraid of them
- In Python a function can return a new function. The inner function a closure and any variable it accesses that are defined outside of that function are free variables.

```
def add_x(x):
    def adder(num):
        # we have read acces to x
        return x + num # x is a free variable here
    return adder

sadd_5 = add_x(5)
add_5 # doctest: + ELLIPSESS
<function add at 0x12324ewe>
print add_5(10)
15
```

Nested functions only have write access to global and local scope.

```
x = 3
def outer():
    x = 4 # now local
    y = 2
    def inner():
        global x
        x = 5 #
    print x
    inner() # only changes the local inside the function
    print x
print outer()
4
4
print x # since global the global value
5
```

Python 3.x has a non-local keyword that replaces the global in Python 2.x

## Decorators

- PEPS 318, 3129, implemented in Python 2.4
- allow you to
  - modify arguments
  - modify function
  - modify results

```
# count how many times a function is called
call_count = 0
def count(func):
    def wrapper(*args, **kwargs):
        global call_count
        call_count += 1
```

```
        return func(*args, **kwargs)
    return wrapper

def hello():
    print 'invoked hello'
```

```
>>> hello = count(hello) # invoking count with the argument being the hello object
>>> hello()
>>> print call_count
>>> 1
>>> hello()
>>> print call_count
>>> 2
```

```
# Decorator Shortcut
@count
def hello():
    return 'hello'
```

Better decorator:

```
def count2(func):
    # TODO - show this one out
```

## Decorator Template

```
def decorator(function_to_decorate):
    def wrapper(*args, **kwargs):
        # do something before invocation
        result = func_to_decorate(*args, **kwargs)

        # do something after
        return result
    # update wrapper.__doc__ and .func_name
    # or functools.wraps
    return wrapper
```

```
# class as a decorator
class decorator_class(object):
    def __init__(self, function):
        self.function = function
    def __call__(self, *arg, **kwargs):
        result = self.function(*arg, **kwargs):
        # do stuff to result
        return result

@decorator_class
def hello():
    return 'hello'
```

---

**Note:** Anything that is callable can be used to create a decorator

---

```
# using a class instance as a decorator
# instead of using __call__ use __init__ and then instantiate the class before using_
↪ it.
deco = Decorator()

@deco
def hello():
    return 'hello'

# You can modify deco later! This is UBER powerful!
```

---

**Note:** Not the same as “Class Decorators”. See PEP 3129

---

## Paramaterized decorators

- need 2 closures

```
def limit(length):
    def decorator(function):
        def wrapper(*args, **kwargs):
            result = function(*args, **kwargs)
            return result[:length]
        return wrapper
    return decorator

@limit(5) #notice parens
def echo(foo):
    return foo

# usage
echo('123456')
'12345'

#syntactical sugar for
echo = limit(5)
```

## Warning: Function attributes get mangled in decorators

- I’ve run into this - when you wrap a function a decorator the attributes get lost
- Docstring kills me
- Do this:

```
def limit(length):
    def decorator(function)
        def wrapper(*args, **kwargs):
            result = function(*args, **kwargs)
            result = result[:length]
            return wrapper
        wrapper.__doc__ = function.__doc__
    return decorator
```

You can also use functools to deal with this issue, but it’s not as clear a read

```
import functools
def limit(length):
    def decorator(function):
        @functools.wraps(function)
        def wrapper(*args, **kwargs):
            result = function(*args, **kwargs)
            result = result[:length]
            return wrapper
        wrapper.__doc__ = function.__doc__
    return decorator
```

## Uses for decorators

- caching
  - I wrote a cache decorator that uses Raymond Hettinger’s LRU cache code.
- monkey patching stfio
- jsonify
- logging time in function call
- change cwd
- timeout a function call

## What if I want to tweak decorator paramers at runtime?

What if I made a mistake in a param and want to change values?

- Use class instance decorator
- Tweak wrapper attributes
- Use context manager
- or...
  - Since a decorator is just a class you can invoke it at runtime. Like this:

```
# TODO get example
result = limit(4)(echo)
```

### 1.10.2 Juju Charm School

Some kind of deployment tool.

- <http://www.socallinuxexpo.org/scale10x/presentations/juju-charm-school>
- <https://github.com/charms>
- <https://juju.ubuntu.com/>

---

**Note:** Demo was powered 100% by the shell. And conference internet is always flaky. This is why I don’t do command-line demos.

---

## 1.11 Mongo LA 2012

### 1.11.1 Keynote: Welcome and What's new in Mongo DB

by Paul Pederson, Deputy CTO, 10gen

#### Design Goal: Rich data model

- JSON/BSON documents
- Good mapping to native programming language types
- Flexibility for dynamic data
- Better data locality
- Schema-free or dynamic schema

#### Footnote to design goal

(DB degrees of freedom)

- Zero degrees of freedom: static queries, static data
- One degree of freedom: dynamic queries, static schema (Relational DB)
- Two degrees of freedom: dynamic queries, dynamic schema (NoSQL DB)

#### General-purpose DBMS

- Sophisticated secondary indexes
- Dynamic queries
- Sorting
- Rich updates, upserts
- Easy aggregation
- Viable primary data store

#### Design Goal: Web Scale

- Scale linearly with *sharing*
- Say 'no' to distributed joins
- Increase capacity with no downtime
- Make scaling transparent to the application

## Design Goal: Minimal knobs

- Make it easy to deploy and manage
- Find natural default configuration options
- Do the right thing out of the box

## Release History

- 1.0 August 2009: supported bson and BTree range query optimization
- 1.2 December 2009: map-reduce
- 1.4 March 2010: Background indexing, geo indexes
- 1.6 August 2010: sharding, replica sets
- 1.8 March 2011: journaling, sparse, and covered indexes
- 1.10 = 2.0: September 2011: cumulative changes

## Changes in 2.0

Here we go...

### Journaling improvements

- Enabled by default for 64-bit platforms
- Journal is compressed for faster commits to disk
- *-journalCommitInterval* command line option exists for specifying some neat feature
- May wait for group commit on write with *j=true* arg to *getLastError()*

### Compaction improvements

---

**Note:** run this after adding an index

---

- Collection-level command:

```
db.mycollection.runCommand('compact');
```

- Copies extent-by-extent using a single 2GB scratch space
- BUilds all the indexes at the end in parallel:
  - First half off external sort occurs while copying extent data. For each doc find all index keys and store these and process.

## Index improvements

---

**Note:** Once you migrate to 2.0 the index changes are not reversible

---

- Keeping the index working set in RAM is important
- v.20 indexes are 25% smaller than v1.8 indexes
- Index compression arises by optimization of BTree index key BSON representation

## Concurrency improvements

- Yielding mitigates reader-writer lock contention
- In general mongodb yields all the time long table scan, yield every 100
- IN v2.0 we now yield around page faults.

## Map-reduce performance

- About 3x faster in 2.0 over 1.8

## Things to follow up on:

---

**Note:** TODO find out what was given in terms of improvements

---

- Priorities
- Replica set force reconfigure
- Durability

## New features

- Multiple location geo search
- Map-reduce sharded output
- Query syntax: \$and
- Custom shell prompts

## Links

- <http://v.gd/mongodb20>

### 1.11.2 Schema Design

by Kevin Hanson, Solutions Architect, 10gen

## Parallels

Tables == Collections Row == Document Column == Field Index == Index Join == Embedding & Linking Schema Object == None

## The Big Question

Do we link or do we embed?

Blog posts and comments

## Embedded

- Faster
- But large embeds can make the master document slow. Ex: If a post has a billion comments

## Linked

- Slower
- Returning the master document requires extra logic

## Each comment gets own doc

Comment gets its own copy of the master blog post

- Fast but inverted
- Great if you have gajillions of comments
- Even more logic

## Denormalization

- Caching via memcached, redis, etc are functionally denormalized instances of data sets.
- NoSQL means you cut out the middleman

More thoughts on denormalized data

- Faster than normalized
- More object-oriented
- application level applications

## Managing Arrays

- Pushing to an array infinitely
  - Document will grow larger than Pre-allocated size
  - Document may increase max doc size of 16MB



## Sometimes you have to limit size of an array

Logic idea:

```
first 200 comments are insert into the blog document
After that have a linked comment document
```

## Schema decisions when sharding

- Can we intelligently partition data?
- Will this partitioning create hotspots?
- Can our partitioning actually improve overall performance?

Bad shard key:

```
Sharding on "date" field and constantly inserting most recent data...
```

Good example:

```
sharding blog posts on "author"
```

---

**Note:** TODO find out why the Good example is actually good

---

## 1.11.3 Running MongoDB in the Cloud

- by Dan Crosta, Software Engineer, 10gen
- I know speaker from twitter and him answering questions about MongoDB to help me with <http://consumernotebook.com>

---

**Note:** Late cause we were intercepted by Redhat/OpenShift marketing who wanted our advice on logos.

---

## MongoDB components

- Config Servers send config data to shards
- Shards can run with the config server down, but it is not fun.

## Replica Sets

Different methods of setup:

1. The most popular
  - Primary
  - Secondary
  - Secondary
2. Another way

- Primary
  - Secondary
  - Arbiter
3. The big option
- Primary
  - Secondary
  - Secondary
  - Secondary
  - Secondary

## Amazon EC2 Instance Types

**Warning:** Never deploy with 32-bit. Don't do it!!!

- Go for a Large or Extra-Large on-demand instance. More expensive but worth it.
- ConfigD / Arbiter can be done via micro on demand instances

## Operating System

- Use ext4, xfs
- Use RAID:
  - Raid 10 on MongoDB
  - Raid1 on ConfigbDB
- Turn off atime
  - File descriptor limits:  

```
cat >> /etc/security/limits.conf << EOF * hard nfile 65536 * soft nfile 65536 EOF
```

### 1.11.4 Turning JSON into info

by Roger Bodamer

---

**Note:** Did not show the query code. I have to look it up online. Ugh.

---

## Relative Queries

- 2 aggregations at the same time
  - 1 by user
  - 1 by location
- Break up into several queries

- Fairly complex
- Easiest in Python or other programming languages

### A note on queries

- There is no notion of declared schema
- The augmented scheme is coded in queries
- Reuse is very hard, happens at a query language

### A word on rendering graphs and reports

- Some libraries
  - Gruff
  - Scruffy
  - HighCharts (Paid for)
  - JRafael
  - JQuery Vizualize
  - MooCharts

Services:

<http://getgauag.es>

- But basically you have to know how to program

### Punchlines

- Fluid requirements are what you get to handle when you use MongoDB
- Know how to program Python (or anything else)
- If you are a business analyst, you're screwed (Not an acceptable answer)

## 1.11.5 Diagnostics & Performance Computing

by Dan Crosta, Software Engineer, 10gen

### Speed

- MongoDB is a high performance database
- But how do you know you are getting the best performance

## Tools

### 1. mongostat

- give it a host and port number. So we can connect to production. Woot!
- tons of useful columns
- mapped
- vsize
- res
- faults - how many disk faults
- locked %
  - In a given window of time, measures two things (TODO find out)
  - Rough percentage measure - not perfectly accurate

### 2. serverStatus

What powers mongostat

```
> db.serverStatus();
{
  "host": "MacBook.local",
  "version": "2.0.1",
  "process": "mongod",

  // lots more stats
}
```

### 3. Profiler

```
> db.setProfilingLevel(2)
{"was": 0, "slowms": 100, "ok": 1}
```

This saves the data into a collection within the MongoDB. Which is nice cause you can reference it later.

See it in operation!

```
> db.system.profile.find()
{
  "ts": ISODate,
  "op": "query",
  "ns": "docs.spreadsheet",
  "query": {"username": "dcrosta"},
  "scanned": 200001,
  "millis": 1407
  // tons more!
}
```

---

**Note:** This is a capped collection of 1MB. So it stores only the most recent. You can change this with some hacks. TODO - find it out

---

## 4. Monitoring Service

- MMS: 10gen.com/try-mms (Free service provided by 10gen)
- Also check out Nagios
- Also check out Munis

## Common problems

### 1. Slow Operations

Check the logs! From the shell:

```
query docs.spreadsheets ntoreturn:100
reslen:510436
nscanned:19976 { username: "dcrosta"}
nreturned:100 147ms
```

This means you need to index the username field

### 2. Replication Lag

Every time you do a read/write, it hits a capped collection called the oplog. Replication lag refers to the time between when a read/write is called and when it is performed.

Example: If you have a very high write rate on the Primary, your secondaries can have trouble keeping up.

### 3. Resident Memory

Always use 64-bit!

```
> db.serverStatus().mem
{
  "bits"64, // need 64, not 32
  "" resident: 7151
  "virtual": ???
  "???": ??
}
```

```
> db.stats()
{
  // other things
  "avgObjSize": 5107.02342342, // capped at 16MB
  "dataSize": 234424323423, // make sure this doesn't exceed your server space!
  // other things
}
```

Equation:

```
indexSize + dataSize <= RAM
```

## 4. Page Faults

```
> db.serverStatus().extra_info
{
  "heap_usage_bytes": 2313132,
  "page_faults": 2381
}
```

## 5. Write Lock Percentage

```
> db.serverStatus().global_lock
{
  "totalTime": 23234234,
  "lockTime": 134646546,
  "ration": 0.002342342
}
```

What to look for: ???

## 6. Reader and Writer Queues

```
> db.serverStatus().globallock
{
  "blah": "blak=h"
}
```

What to look for: Things that are eating up tons of process. To stop it, run:

## 7. Background Flushing

```
> db.serverStatus().backgroundFlushing
{
  "flushes": 5634,
  "total_ms": 83556,
  "average_ms": 14.832342342,
  "last_ms": 4,
  "last_finished": ISODate
}
```

In some case you should flush more frequently then MongoDB does by default

## Disk Considerations

- Raid: Use it
- SSD: If you can get your server on a SSD, then things will go much, much faster.
- SAN?:

## 8. Connections

```
> db.serverStatus().connections
{"current": 7, "available": 19993}
```

- Make sure you have enough connections.
- On Linux, change the number of connections that can be opened.
- MongoDB can handle up to 20,000 open connections

## 9. Network Speed

```
> db.serverStatus().network
// data here
```

Check this as one of the things that might be bottlenecking

## 10. Fragmentation

```
> db.spreadsheets.stats()
{ // data here
}
```

- When you move data around frequently, fragmentation occurs.
- This will cost you more memory, slowing things down
- “2 is the magic number”. Your disk should be at least twice as big as the MongoDB memory

```
// blocking command.
// Be careful!!!
db.spreadsheets.runCommand("compact");
```

Compacting fixes the problem, but it stops operations on that server. So run it against a secondary instead of the primary.

### 1.11.6 Indexing & Query Optimization

by Kevin Hanson, Solutions Architect, 10gen

## High level

- What's an index and why do we need one?
- As we insert data into MongoDB, we store that as a linked list
- So if you search for something in 7 documents, it has to search in all 7 of them

## Creating indexes in MongoDB

- You can index anything
- All docs have an `_id` field that is auto-indexed
- new indexes:

```
db.blog.save({author: "James", ts: new Date()})
db.blogs.ensureIndex({Author: 1, ts:-1})
```

## Things to know about indexes

- Slows down writes
- But speeds up reads!
- Forces uniqueness on a title

---

**Note:** TODO - check that we don't have dupe titles

---

## Covered Index

- Query resolved in index only
- Eliminated need to pull documents from DB
- Need to exclude `_id` from items projected.

```
db.blogs.save({
  author:"Kevin",
  editor:"Katie"
})
db.blogs.ensureIndex({author: 1, editor: 1});
db.blogs.find({author}) // TODO finish this
```

## Spare Index

- Key value included if and only if the value is present
- Reduces the size of index
- Limited to a single field

```
// TODO fill this out
```



## Unique Sparse Index

- Key value included if and only if the value is present
- Reduces the size of index
- Limited to a single field
- Null and not-present are different

```
// TODO fill this out
```

## Geospatial indexes

- Geo hash stored in B-Tree
- First two values indexed

## Query Performance Analysis

---

**Note:** Speaker had to go super fast here because he kept answering questions. This is why you ask people to wait for the Q&A at the end.

---

### 1.11.7 Closing session and MongoDB roadmap

- by Paul Pederson, Deputy CTO, 10gen
- [paul@10gen.com](mailto:paul@10gen.com)

### v2.2 projected 2012 Q1

- Concurrency: yielding and db or collection-level locking
- Improved free list implementation
- New aggregation framework
- TTL collections
- Hash shard key
  - Hashing gives you flat distribution

## Concurrency Issues

- No excessive blocking
  - dropIndex
  - getLastError
  - isMaster
  - etc...
- May block for long times

- foreground index creation
- reindex
- compact (is that a startup name? ha ha ha)
- repair database
- creating a very large (many gb) capped collection
- validate connection

## Aggregation Framework

- Declarative, no JavaScript required
- Pipeline model: \$match, \$project, \$group
- Easy to add new operations
- C++ native (non-JavaScript) implementation

## TTL Collections

- Currently: Evict old data to make room for new records by creating a timestamp index, and create a cron job to delete stale items with update
- Coming: per object or per collection: automates deleting documents older than some limit.

## Harsh Shard Key

- **If** you are not expecting range queries on the shard key
- **Then** it makes sense to shard by hash key, you naturally get a flat distribution
- In a sense this is the easiest possible case
- MongoDB started by solving the hard case.

## Short List (not in 2.2 but coming up)

- Full text Search (so you don't need SOLR)
  - The absolute number one requested feature
  - Done but needs to be vetted and tested better
  - Text searches can generate bajillions of extra records and other issues
  - Sounds like they are trying to do it right.
- More concurrency
- Online compaction
  - Make the system smaller on the fly
  - This way you don't have to play replica set games to clean things up
- Internal compression
- Read tagging

## 10gen Hiring

- NYC/Silicon Valley - may see us
- EU
  - London
  - Dublin
- Anywhere - Language Evangelist

### 1.11.8 Sponsors

#### 10gen, makers of MongoDB

The main hosts of the event

#### Redhat for Open Shift Paas

Redhat's cloud hosting system with git powered deployments.

#### Joyent Cloud

Better, cheaper, faster system in the cloud. They sell the systems that PaaS are built on.

#### VMware Cloud Foundry

Completely open source PaaS - same was what ActiveState Stackato uses

## 1.12 PyCodeConf 2011

- Organizer: Github!
- Venue: Epic Hotel, Miami, USA

### 1.12.1 Future is Bright

- By Jesse Noller
- <http://bit.ly/qqxpt8>

#### What he does

- PSF board member
- Pycon chair
- Python core dev
- Dad, developer

## What is Python?

- Language
- Community
- PyPy, PyPI
- Heroku: <http://bit.ly/o73sOR>
  - Humble community - no rockstar personas
  - Approachable
  - etc

## Where is Python used?

- Disney Animation Studios
- NASA
- Many other things
- Too many cool places to list
- Python is everywhere
- Everyone uses Python

## Python is amazing

- Easy to learn
- Easy to use
- Very fast
- Large scale, small scale

## Status of the Language

- Approx 123 accepted PEPs
- about 80 Built-in functions
- 250+ stdlibs
- Python 2.7.x is last of the 2.X series
- How about status of <http://www.python.org/dev/peps/pep-0397/>?
  - Windows installer for Windows
  - I don't use Windows, but my students usually do

### Jesse's Personal Wishlist

- Better messaging systems
- Actor support in stdlib
- Support for gevent and other things

### Jesse says we need

- More Pythonic APIs (mentions Kenneth Reitz)
- ...to remain conservative in changing the language too much
- ...but adding to the stdlib is a problem
  - Barely fits in the head
  - stdlib stalls things

### PyPy!

- Super-fast
- Gets things done
- A bit complex - needs people like Alex Gaynor to do the work
- Doesn't handle cpython stuff that touches C stuff.
- predictions:
  - Will be used more and more
  - Will continue to be based off the cpython implementation

### Python interpeters

- Need to work together, tests, compatibility, etc
- BFFs:
  - PyPy
  - CPython
  - Jython
  - IronPython

### Python 3

- Keep calm and carry on
- Python is 21 years old, a 5 year plan to migrate to it is nothing
- Python 3 porting is getting finding
- The PSF is willing to give out grants

## Community

- Look at the number of Python conferences!
- I got mentioned by Jesse! Yeah!
- Come to workshops and meetup groups
- Get involved
  - Outreach
  - sprints
  - <http://pyladies.com>
- Don't be a jerk
  - Stay positive
  - Not all criticism is constructive
  - It can be hard to fight through vitriol and find what's worthwhile

## Questions

- CW: What PEPs will affect the language
  - Answer: Hard to say cause there are so many things going on
  - Answer: Twisted components into core is on the docket
  - Answer: Some API sugar

### 1.12.2 Embracing the GIL

- by David Beazley
- slides: <http://dabeaz.com/talks/EmbraceGIL/>

#### Embracing the GIL could be better

- People love to talk about it
- Rant about it
- Godwin's Law of Python?

#### premise

- People love to hate on them
- That's because threads are useful
- Threads make great stuff work
- Even if you don't see them directly

## The Gil in a Nutshell

- Every Python file gets compiled into VM instructions
- In cpython, it is unsafe to execute instruction concurrently
- Hence: Locking

## What GIL protects

---

**Note:** duh missed this

---

## Major GIL issues

- Threads using multiple CPUs
- 

## The Challenge

- The GIL is unlikely to go away anytime soon
- Can it be improved? Yes!
- How can it be done?
- How about Python 3!

## Experiment

- request/reply server for size-prefixed messages
- each method has payload/header

## Why this experiment?

- Comes up in a lot of contexts
- Involves IO
- Used as a foundation for a lot of other things

## Five different implementations

- 1000 iterations of some simple code
- Done on EC2 with nothing else running
- implementations
  - C + 0mq
  - Python + 0mq

- Python + multiprocessing
  - Python + blocking sockets
  - Python + nonblocking sockets
- Results
  - All finish in about 13 seconds

### What happens when you introduce a thread?

- What does it do to the performance?
  - C + 0mq (*samish seed*)
  - Python + 0mq (*7x slower*)
  - Python + multiprocessing (*8.9x slower*)
  - Python + blocking sockets (*approx 10.x slower*)
  - Python + nonblocking sockets (*approx 10.x slower*)

### Commentary

- Simple test
- Not a hard-core realistic talk
- How about PyPI?
  - What? Older version was 567 slower!
  - New version is much faster. .. note:: Get results!

<b>Warning:</b> Distracted by some work stuff. Missed some awesome stuff here.
--

### Performance Explained - thread priorities

- To fix this, you need priorities
- The original “Fix GIL” patch had priorities
- That should be revisited

### Another experiment

- David’s 3.2 fork with priorities
- Not suitable for real work
- Interesting for testing
- Lets you set the priorities manually



```
import sys
import threading
def spin(value):

    sys.set_priority(-1)
```

### Some thoughts

- Huge boost in Python with only minor changes to a few files
- Is this the only GIL improvement?
  - No
  - There are other ways to do it
  - GIL released on non-blocking I/O operations

### PyDanny take away

- Now I think I grok the GIL issues finally
- Ya, me is slow. :)

## 1.12.3 Python is Awesome

---

**Note:** Watched this at PyCon AU. Copied over my notes from there so I can fill in the holes here.

---

- By LA Python's own Raymond Hettinger

### Context for Success

- License
- Commercial Distributions
- Zen
- Community
- Repository of Modules (PyPI)
- Killer Apps and Success Stories
- Win32
- Books

### License

- Most Python releases are GPL-compatible. This makes it free.
- Going to a closed source language means you are trapped.

## Community

- Mailing lists
- Newsgroups? HA HA HA
- Python User Groups

## PyPI

- Repo for Python programming language
- Over 16,000 packages
- *pip install ordereddict* works for Python 2.5!

## Killer apps

- Zope, Django, Pyramid
- Numpy and Scipy
- Bittorrent and Twisted
- YouTube
- Blender and Maya
- Win32 - Factoid: Me, @pydanny, has done all his windows programming using cpython and Win32!

## Easy to learn!

- Good teachers.
- Think how fast you got the types and control structures in Python. General 3 hours
- In a day you can learn special methods and stdlib
- Critical because if you need good Python developers it doesn't take long to get up to speed. Converting developers takes:
  - C takes 2 years to get competent
  - Java takes 6 months to get competent
  - Python takes a week to get competent
- Rapid development cycle
  - Scripting languages are unbeatable for development speed
  - Programs are grown organically
  - Interactive testing lets people work with their code results immediately.
  - Bang out real code fast

## Economy of expression

- Not many words or characters to get things done.
- clear English means non-coders can understand your work
- **Pydanny factoid:** One of the first times I wrote Python on a whiteboard for a boss at NASA/SAIC they thought it was very legible pseudo code representing a complex process.

```
import hashlib
import os
import pprint
hashmap = {}
for path, dirs, files in os.walk('.'):
    for filename in files:
        fullname = os.path.join(path, filename)
        with open(fullname) as f:
            d = f.read()
            h = hashlib.md5(d).hexdigest()
            filelist = hashmap.setdefault(h, [])
            filelist.append(fullname)
pprint.pprint(hashmap)
```

## Beauty Counts

- Readability is the #1 mentioned characteristics of why organizations choose Python
- The beautiful appearance on the page directly affects a programmer's sense of joy
- Makes us go home and write code
- If you can read other people's code that makes it easier to maintain
- Because we all *mostly* share the same idiom it means we can read each other's code. That doesn't stifle creativity - it just means we can get along.
  - As a parent I can say I would have *loved* having a formal uniform at school. As a geek in school I would have loved that too. :P

## Interactive Prompt (REPL)

- Python experts don't memorize Python
- They use the interactive prompt often (I try to write tests...)
- This is a killer features that runs circles around compiled languages
  - Python shell
  - IPython
  - BPython (My favorite)

## Behind the Scenes

Philosophy of core dev

- Conservative growth
- *We read Knuth so you don't have to*

- Aim for simple implementation

## Protocols

To interact with these we have defined protocols

- DBAPI
- Hashlib
- Compression
- WSGI for the web
- Conversion protocols

## Specifics of Python: The Foundation

- Dictionaries and Lists
- Automatic memory management
- Overridable syntax
- Exceptions
- **You can reprogram the brackets?**
- **And we can reprogram the dot?!?**

## Winner Language Feature: Iterator Protocol

- High level glue that holds the language together
- Iterables: strings, lists, sets, dicts, collections, files, open urls, csv readers, itertools
- Um... I know this. I've had to construct these on my own in other languages. But not Python... Wow - I just realized this just now.

```
# When Raymond wrote **sorted** he wasn't thinking about sets
# But they still just works
sorted(set('abracadabra'))
sorted(set(open(filename)))
sorted(set(open(filename)))
```

**Warning:** If you say “Python has iterators, you have to explain how it is globally implemented. Other languages have iterators, but they have to be implemented and extended and stuff”

## Winner Language Feature: Generators

- List comprehensions give us joy
- Logical extension to list comprehensions and generators to unify the language
- List generators are amazing. No one else has them
- Serious magic

- A million rows in a generators is nothing
- Simple syntax to do them. You only need the YIELD keyword.

```
# Sample generator code
def pager(lines, page_len=60):

    for lineno, line in enumerate(lines):
        yield line
        if lineno % pagelen == 0:
            yield FORMFEED

# genexprs setcomps and dictcomps
sum(x**3 for x in xrange(10000))
```

---

**Note:** I've used list generations to super-optimize slow code

---

### Proposal: Generators that accept inputs

- Generators support **send()**, **throw()**, **close()**
- Unique to Python
- Makes it possible to implement **Twisted**'s *inline deferreds*
- Add one line of **Twisted** to your code and it infects your whole app
  - Twisted forces you to write in callbacks
  - Callback coding is hard to follow and debug
  - Wouldn't it be great if we could have the benefits of Twisted in procedural code?

```
# two way generator example
@inlined_deferred
def session(request, cleared=False):
    while not cleared:
        cleared = yield authenticate(request.user)
    db_result = yield database_query(request.query)
    html = yield format_data(db_result)
    # TODO finish getting this down
```

### Winning language Decorators

---

**Note:** I have problem writing these things. Serious problems. :(

---

- Expressive
- Easy on the eyes
- Works for functions, methods, and classes
- Adds powerful layer of composable tools
- Raymond shows sample code from Daniel Lindsley's Itty!

– <https://github.com/toastdriven/itty>

### Winning Language Features: *exec*, *eval*, *type*

- Not a fan of *exec* and *eval* because when used in my experience they are done badly
- But *type* is awesome

### Winning Language Feature: With Statement

- Clean, elegant resource management: threads, locks, etc
- Important tool for factoring code
- Factors-out common *setUp* and *tearDown* code.
- The reverse of functions

```
with locking:
    access_resource()
```

### Winning Language Feature: Abstract Base Classes

- Uniform definition of what it means to be a sequence, mapping, etc
- Ability to override *isinstance()* and *issubclass()*
  - New duck typing method: **Just say you are duck!**
- Mix-in capability
- Sample:

```
class ListBasedSet(collections.Set):

    def __init__(self, iterable):

        self.elements = lst = []
        # TODO add more

    def __iter__(self):
        return iter(self.elements)

    # TODO add more methods
```

### Winning Language Feature: Indentation

- Makes the code really clear
- We write our pseudo code this way
- Less errors!
- Less ambiguity!

### 1.12.4 Backbone.js + Django

---

**Note:** I'm having trouble keeping up when it comes to writing JavaScript fast. :P

---

- Question: Why not JQuery templates?
- Question: Best Practices?
- by Leah Culver
- Works at convore.com, a YC Combinator funded project
- LeafyChat - Django Dash 2009
- web-based IRC client

#### Convore issue?

- Who is supposed to use it?
- Internal company stuff
- What kind of discussions

#### Grove!

- IRC for your company
- Internal for companies
- <https://grove.io>

#### Leafy Chat

- Pure JavaScript
- very barebones - just JQuery
- Very dirty in that their construct HTML manually

Each submit for chat:

1. handle form submit
2. create new message
3. display mesage in list
4. POST method in AJAX

#### Backbone!

- MVC style of programming for AJAX/JavaScript
- More like DJango: MTV

```
// models are easy!
window.Message = Backbone.model.extend({
  model: Message,

  initialize: function() {
    this.model.bind('add', this.addMessage)
    // TODO

  },

});

// form submits
submitForm: function() {

};
```

## Handlebars templates

- handlebars.js templates
- looks like Django/Jinja2 templates
- See include\_raw template tag as per <http://djangosnippets.org/snippets/1684>

## Addition Goodies about backbone.js

- Uses similiar routing to Django
- Handy code snippet by Leah for Django CBV usage:
  - <https://gist.github.com/1265346>
- Event based asynchronous
  - One thing can fire off multiple request
  - So if I am watching and someone else posts then I see the results

## Router

- Can do overlaps of views

### 1.12.5 PyPy talk

by Alex Gaynor

- Student at RPI
- Core Python Dev
  - cpython
  - PyPy
- Core Django Dev



- Interned at Quora and got them on PyPY
- Dressed very classy.

## Two things go faster than C

- neutrinos
- PyPy

## Story of PyPy

- psyco was JIT python
  - Managing it was hard
  - hardcoded for 32 bit CPUs and we are on 64 bit
  - Any changes to core Python killed pysco
- Years ago created a Python interpreter inside of Python
  - 2000x slower than cpython
  - ran on restricted Python (**r-python**)
  - Wrote a great compiler: now 10x slower than cpython
  - Added better garbage collection: now 4x slower than cpython
- New JIT for Python
  - Writing a JIT for Python sucks
  - Writing a generator for making JITs for any language is easier
  - **Alex Statement:** “PyPy is the only project I know of that uses SVN branches. That’s the most impressive part”
  - Doing it this way made it faster? How? Wizard Magic?
- Now PyPY is going fast:
  - Crazy that it runs so much faster than cpython
  - Hard to believe
  - Python using a JIT generator to create a JIT library?
  - Faster than cpython
  - <http://speed.pypy.org/>

## Why you should use PyPy

### Science!

- Fast and scientist friendly
- Now works with numpy!
  - Not complete

## Tools

- jitviewer
  - Finding slow spots in existing code
  - Looks at Python, byte code, assembler, et al

## Fast!!!

- Faster than cpython
- They now metric it's speed against C, not Python
- Compatibilities
  - Much work with third-party integration
  - lots of people are using Python instead of C extensions
  - PEP in place so that new stdlib stuff has to be pure python
- Quora is much more snappy

## Python 3

- PyPy is beginning the move to Python 3
- JIT generator to the rescue!

## JIT generator means...

- **They can branch the Python 2.7 PyPy stuff to Python 3**
  - Make the Python 3 version work
  - And since the JIT generator makes the code, both versions **will just work**
- Can do a GIL version for single CPU or non-GIL for multicore
  - JIT generated so...
  - **Both versions just work!!!**

## The Future

- Lets make Python faster!
- Give us problem children to fix!

## 1.12.6 Processing Firefox Crash Reports With Python

- by Laura Thomson
  - Web tools engineering manager
  - author of two books:

- \* PHP and MySQL Web Development
- \* The Surrealists
- Done about 100 talks!
- Mozilla is hiring like crazy

## Overview

- The basics
- The numbers
- Work process and tools

## The Basics

- Socorro crash information collector thingee
- Lots of companies use it to track this data:
  - Steam (game stuff)
  - Other things

## How crashy is the browser?

- Mozilla Crash report - please use it!
- Will email you if you have malware they detect
- Generates <https://crash-stats.mozilla.com/products/Firefox>
- Mozilla needs your data to make Firefox better.

## Basic Architecture

- Database is PostGres
- HBase for map-reduce, she wants to replace it with something else
- Lots of components powered by Python
- Front-end is PHP but will be converted to Django in 2012

## Lifetime to a crash

- Browser crashes
- Sends data to Mozilla in a big binary dump with a JSON header
- Mozilla processes the header and tries to generate a signature of the crash
  - They need more than just the function that created the crash
  - Doesn't cover all cases
  - Uses a regex to glean out other things from the binary crash dump

## Back end processing

Large number of cron jobs, e.g.:

- Calculate aggregates: Top Crashers (Farmville if you want to know)
- Process incoming builds from ftp server
- Match known crashes to bugzilla bugs
- Duplicate detection
- Match up pairs of dumps (OOPP, content crashes, etc)
- Generates extracts (CSV) for engineers to analyze

## Middleware

- Moving all data access to be through REST API (by end of year)
- (Still some queries in webapp)
- Enable other front ends to data and us to rewrite webapp using Django in 2012
- Upcoming (2011 or 2012) each component will have it's own API

## Webapp

- Hard parts: How to vizualize some of this data
- Ex: Nightly builds, moving to reporting in build time, not clock time
- Code crufty (old KohanaPHP)

## Implementation Details

- Python 2.6 mostly (PHP is the exception)
- Post Gres 9.1
- memcache for the webapp
- Thrift for HBase access
  - HBase is meant to work with Java
  - Could do it in Clojure/Scala but finding resources would be hard
  - Thought about Jython then backed off
  - Considering alternatives
- 100 users
- 100 Terabytes of data

## Some Numbers

- At peak 2300 crashes per minute
- 2.5 million per day
- Median crash size 150K, max size 20MB (reject bigger)
- ~110TB stored in HDFS (3x replication, ~40TB of HBase data)

## What can they do?

- Does a version of FF crash more than others?
- Analyze differences between versions of Flash
- Detect duplicate crashes
- Detect explosive crashes
- Find “frankenstalls” that can happen on Windows
- Email victims of malware

## Implementation Scale

- > 115 boxes (not cloud cause that won't cut it)
- Now 8 devs + sysadmins + QA + Hadoop ops/analysts
  - Hiring: <https://whitespacejobs.org>
- Deploy approximately weekly but could do continuous if they need

## Development Process

- Fork
- Hard to install (must use VM)
- Pull request with bugfix/feature
- Code review
- Jenkins polls github master, picks up changes
- Jenkins runs tests, builds a package
- Package picked up and moved to dev
- Wanted changes merged to release branch
- Jenkins builds release branch, manual push to stage
- QA runs acceptance on stage
- TODO missing
- TODO missing

## Absolutely Critical!

**Build all the machinery for continuous deployment even if you don't want to deploy continuously**

- You don't want to install HBase

## Upcoming

- ElasticSearch implemented for better search
- More analytics; automatic detection of explosive crashes, malware, etc
- Better queueing
- Grand Unified Configuration System

## Everything is Open Source

- <https://github.com/mozilla/socorro>

## 1.12.7 The Future of Collaboration - Daniel Greenfeld

---

**Note:** Audreyr took these notes. :)

---

### Intro

Danny cartwheels, still blogs, works on Django Packages and whitespacejobs.org

Mark Pilgrim is gone

- feedparser, httpLib2, Dive into Python, Dive into HTML5
- How much did we lose with Mark leaving the developer community?
- kennethreitz created a mirror at <https://github.com/diveintomark>

Where is httpLib2?

- PyPI? No
- Not Google code
- Hard to find cached download
- Many libraries depend on it

Repeating history?

- django-piston, python.org, opencomparison.org all have bus factor and need active maintenance

## Dark future?

- Critical Python packages vanish
- Build scripts fail
- Can't always replace from caches/backups
- Legacy projects unmaintainable
- Domain knowledge leaves
- Hard to move forward
- 3rd party community as critical as core
- Actually, this is not the future. It's today

Like the Library of Alexandria

- When we lose our history, we lose ourselves

## Trust issues

- External and internal social issues
- Makes collaboration hard
- Causes "Not Invented Here" plague

## Solutions?

Sponsorships

- But focused on short-term development, unusable code from interns
- Server costs are not the issue

Community managers

- Needs core/senior devs
- They're busy
- Volunteers have different priorities

Paid community managers

- Work with package authors/maintainers
- Mitigate social issues
- Precedence: Ubuntu, Fedora, Twilio, Github

How do we keep Python's community projects active?

- PSF project incubation: YC-style seed funding
- Helping market projects via python.org, blogs, other channels
- Help community projects find a business model, sustain themselves
- Copy startup model for projects that benefit Python

## 1.12.8 The State of Packaging & Dependency Management

by Craig Kairsterns

---

**Note:** started late cause was coming off my talk

---

### Pip

- replaced easy\_install
- Actually supports uninstalling
- Lots of small improvements
- Supports version control (**Use only pinned versions!**)

### virtualenv

- Sandbox tool
- Destroy and recreate often

### Best practices

- pin your versions
- Don't use repos for production
- Not for deployment!

### What's missing?

- locks

### Recap

- packaging
  - use PyPI
- Dependency Management
  - Pip
  - virtualenv

### Thoughts

- We're in better shape than realized
- Just need to use the tools we have effectively



### 1.12.9 API Design and Pragmatic Python

- by Kenneth Reitz
- Works for readability.com
- Used to work for changelog
- loves open source
- Author of requests and tablib libraries

---

**Note:** Kenneth's mic kept going out. Hope all his words are captured!

---

#### Alternative Titles?

- Python is a Ghetto
- Python Jumped the Shark
- Python for Humans!

#### His libraries

- Requests: HTTP for humans
- Tablibs: Pythonic Tabular Datasets
- legit: Awesome Git Interface
- OSX-GCC-Installer (angers lawyers)
- Clint: Command-line Interface Tools
- Httpbin.org: Request & Response Server

#### Philosophy

We share a dark part:

- PHP
- Java
- ColdFusion

We all love the Zen of Python:

```
>>> import this
```

Bits:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- If the implementation is hard to explain, it's a bad idea (unless you are PyPy)
- There should be one and only one way to do things

## HTTP as an example of API issues

### Github API client in Ruby

TODO: Get Ruby example from his slides. Ruby makes this easy

### Github API client in Python: Ugh

- Pick the right std lib `http/url/lib/2`

TODO: Show the the example Python code from Confessions that I stole from Kenneth. :)

TODO: Show his example from trying to hit a private repo

### Admit it

- If this were you coming into Python, you would leave and not come back
- This is a serious problem

### urllib2 is toxic

- Over-engineering
- Makes the simple/trivial hard
- Hard to debug
- Hard to test

### Solution

- Python needs more pragmatic packages
- Pragmatic means implementation without focusing too much on theory
- Figure out the actual reqs

### Contention

- If you have to revisit docs every time to use an API the API is bad
- 

### Subprocess

- Powerful
- Effective
- Second worst stdlib module ever

## Solution

- **envoy**
- Replaces/wraps Subprocess

## File and System Operations

- `sys | shutils | os | os.path | io`
- Really difficult to run external commands
- The blocks dev+ops guys from wanting to use Python

## Installing Python

- Installing Python
- Decisions, Decisions (Binary, Homebrew, 32 bit, Macports, Source, what?!?)
- What happened to just one way to do it?
- Pain on Mac, Windows

## XML

- *etree* is terrible
- *lxml* is awesome, but difficult to install

## Packages and Dependencies

- pip and easy\_install?
- setuptools not included with python?
- Distribute?
- No easyt\_uninstall

## Date[time]s

- datetime, time, calendar? Which one?
- What third-party libraries are around

## Unicode

- Kenneth says it's a simple problem
- Danny: Maybe the core docs should have an easy-to-find good description? Am I missing something? Is it an SEO issue?
- Example: <http://farmdev.com/talks/unicode> is great, but how do you find it?

## Installing Dependencies

Hard to do:

- PIL
- TODO: get more

## Hitchhikers Guide to Python

- <http://python-guide.org>
- A guide to newcomers
- References for seasoned veterans
- Install resistance to doctest
- Stays with just one way to do things

## Solves

- makes Python more accessible
- Great references

## Manifesto

- Simplify terrible APIs
- Document best practices

## 1.12.10 Python is Only Slow If You Use it Wrong

- by Avery Pennarun
- Google employee
  - But this talk has nothing to do with them
  - If you apply to google and say his name he get's money. :)
- Trying to talk about bitter

## Stuff he's done

- bup: Python software doing massive things for real problems
  - <http://github.com/apenwarr/bup>
- sshuttle: VPN software tht handles 802..11 g/n speeds in python
  - <http://github.com/apenwarr/sshuttle>

## Easiest way to do Python wrong

### tight inner loops

```
chars = open('file').read()
for char in chars:
    ...
    # slow
```

- Don't do this
- Apparently for dynamically typed languages, this is a very, very slow operation

### Speeding things up

- Use regexes and c modules
- No such thing as 100% pure python
- forget about SWIG
  - writing C modules is easy and integrating them easy too
  - SWIG is a code generator for C++
- python + C is so far the winning combination
- C is simple; Python is simple; PyPy is hard
  - The concept behind PyPy is really hard
  - Python and C are relatively straightforward compared to the concepts of PyPy

---

**Note:** I want to learn how to write C and then add it to my Python work.

---

## Other way to do things wrong

- Computation threads
  - Worthless because of GIL
- Threads are okay for I/O
- fork() works great for both
- C modules that use threads are fine

## Garbage Collection

### RefCounting

- Every time I use a variable I increase its reference count by one
- Every time I don't use something its reference count goes down by one
- When it hits ZERO then it goes away

## Refcount... and threads: BAD COMBO

- Variable shared between threads forces a lock on the refcount
- One reason why removing the GIL is almost impossible
- There are tricks...

## Python is not a garbage collected language++

```
for i in xrange(1000000):  
    a = '\0'*10000
```

- Sample code in Python
- Metric test done in **Python, PyPy, Java, C, and Go**
  - **Java**: Running this loop takes more memory and more time than CPython!
  - **PyPy** takes about the same time as Python
  - **C** is much faster
  - **Go** is much slower

## Java is a garbage collected language

- Three different collection strategies
- See his upcoming research paper: **Seriously Java, WTF?**
- Amusingly, the new threaded java system is slower and takes more memory
- “Ever notice complex Java programs seem to run slow and take up tons of memory?”

## ++Exception sometimes python is a garbage collected language

- Refcount sometimes fails
- Did you know Perl never drops objects?
  - This is why you **can** have memory leaks with it.
  - Avoiding this requires a deep understanding of Perl

## Get the most out of Python's GC

- JUST AVOID IT AT ALL COSTS
- Break circular references by hand when you are done
  - trees are a good example
  - TODO: find out what he meant somehow
- Better still: use the weakref module

## Deterministic Destructors

### Quiz: Does this program work on win32?

```
open('file', 'w').write('hello')
open('file', 'w').write('hello')
# YES!!! Cause Python doesn't do Garbage Collection. refcounting FTW!
```

With “real” GC you have to manually manage resources:

- files
- database handlers
- sockets
- locks

When you are done with a variable, it should go away. It shouldn't stick around. Predictable behavior!

### Don't take away our Deterministic Destructors

- Maybe the GIL is a good thing
- refcounting is good

### JIT vs ???

---

**Note:** TODO - find out the missing half of this title

---

- HelloMark benchmark language
  - Simple process benchmark for command-line tools
1. C
  2. Go
  3. Perl
  4. Ruby 1.8
  5. Ruby 1.9
  6. Python
  7. mono
  8. Java
  9. java-client
  10. java -XX:+UseConcMarkSweepGC
  11. pypy
  12. C + valgrind
  13. jython
- Many of these commands run in about 2x the time of C hello world.

- This is not good for Git
- Slow speed hurts user experience

### **.pyc rocks**

- are awesome
- compiles Python files so you get fast
- Ruby tools like *Rails* take forever to reload after a file change
- Django, Pyramid, Tornado, et al does it really fast

### **Summary**

- Love refcounting, hate gc
- Don't write tight inner loops
- If you are using the JIT, you are doing it wrong

## **1.12.11 Amazing Things in Open Source**

- by Audrey Roy
- Lots of volunteer work: PyLadies and opencomparison.org

### **Python community is a meritocracy**

If your work has merit people use it

- Anyone can build anything
- Anyone can start a user group
- Anyone can be a leader

### **No permission needed**

- Just implement or experiment with what you want/need
- Fork if necessary
- Ask forgiveness later!

### **It's fun**

- Rewarding to see your work published



## Who's in charge

You are if...

- ...you deliver code
- ...you maintain it

## Decisions

- Forced to make quick decisions during Django Dash
- All packages are added manually, using:
  - package name
  - PyPI URL
  - repo URL
- No spiders, no automation, good decision?
  - Doesn't matter, looks like it's successful
- Be careful of mailing lists, IRC, et al
  - don't talk too much before implementation
  - Just get something done

## Your gut instinct is often right

- Django Packages
  - **Fun fact** Many of the grids were created as test fixtures and have remained
  - You can change them but keep in mind we'll track the changes and hunt down people who do wrong

## Django

### Why she uses Django these days

- Lots of packages
- Can wire things together

### Django Core vs Apps

- Many, many batteries included
- Gives you one obvious way to do things
- Third party apps: "Django apps"
- Good
  - One pretty clear way to do things
- Bad

- Stuck with one way to do things.
- Example: URL routing differences

### Clear pattern for Django apps

- Simple
- Easy to understand, implement, install
- Documented
- Repeatable

### Django's Ecosystem over time

- More and more new innovations implemented as 3rd-party packages
- Problem with adding all to core is then you are stuck
  - Deprecation becomes challenging
  - Additional complexity

### Observation on Packages

- Umpteen JQuery plugins
- Perl: 100K modules
- Python: 17K packages

So very useful!

### Pyramid Core vs. Add-ons approach

- smaller core, more add-ons
- Anyone can write add-ons
- Some are “officially endorsed”
- Easier to do extensions of the core
- Young, but potential for rapid growth
  - Hopefully <http://pyramid.opencomparison.org> will help that growth

### Pyramid's Ecosystem over time

- Past: Pylons, Repoze.BFG, TurboGears
- Present: small core, docs for doing add-ons - but not many yet
- FutureL Lots of add-ons!

### Checklist: What 3rd Party Package Devs need

- “Best practices” doc on how to write 3rd party packages
- Well defined, easy-to-understand spec
- Sample code (as much as possible)

<b>Warning:</b> Telling people to “read the source code” is <b>not</b> the answer.
--

- Active Community
- Mailing list, IRC
- Docs, tutorial, sample projects

### What about too many options?

- Zen of Python: “There should be one– and preferable only one –obvious way to do it”
  - This is about Python language constructs
  - Not about 3rd party packages
- Sometimes packages are close duplicates
  - Please document how you are different from other tools
  - deprecate when your stuff gets old, don’t leave people hanging!

### Too much fragmentation?

- Lots of Python groups and tools! Maybe too much?
- NO SUCH THING. MOAR IDEAS PLEASE!
- We need diversity of ideas and approaches

### What makes a package useful?

- Unix philosophy: Do one thing and do it well
- Usability: install docs, pip, PyPI
- Reliability: tests, maintained, community

### Anti-patterns

- Don’t underestimate the impact of your notes on-line
  - Your snippet on your blog can get hit 25K+ times
  - Package up your stuff and deploy to PyPI
- Don’t over-engineer things to make them pluggable, abstract
  - urllib2 is a good example
  - counter: sometimes a single file is good

- Too much functionality
  - Kitchen-sink base platforms
  - utility, do-everything packages
    - \* django-extensions: ugh
  - duct-tape packages that try to fix everything once
    - \* HTML world: CSS resets that also do typography, layouts, and more

### Glory Pattern: Be Pythonic

- Why do we love Python?
  - Elegance
  - Ease of Use
  - Explicitness, clarity
  - Simplicity

### Community Building

- Mentorship
  - Today's new users are tomorrow's contributors & leaders
  - Mentorship groups: Positive encouragement
    - \* PyLadies
    - \* Python Core Mentorship
    - \* (need more like this)

### Diversity of Ideas

- Look at schedules & slides from PyCons around the world
  - KiwiPycon: Home of Twisted!
  - PyCon AU: Focus on core Python, PyPI
- Ideas differ country-to-country
- Same goes for other types of diversity besides geographic
  - SoCol Python is often more about deployment/scaling
  - LA PyLadies is often about asynchronous

### Summary

- Build what you want
- Encourage the 3rd party community to support your effort
- Be helpful

### 1.12.12 The Prejudgement of Programming Languages

- by Gary Bernhardt
- runs *Destroy all software*

Alternative Title: 10 years of failures and bad mistakes

#### 2001-

- Used to be ignorant of software

#### 2001

- C is the best thing!
- Java Sucks cause it has garbage collection
- Programming is supposed to be hard, right?

#### 2003

- Learned Lisp
- Started his own crazy language
- But Python did the same thing, so went with that instead
- Python <3

#### 2006

- Exhausted by his company, Python
- BitBacker

#### 2009

- Started doing Ruby/Python 50/50
- Every day of the year switched languages halfway through

Quotes of the time:

- “I can integrate Python lib in 10 minutes, Ruby lib in an hour. . .”
- “Ruby syntax tricks can be hard, but other languages might want to take note”
- “Wrote a Python specer that would have been trivial to do in Python”

Not sure Ruby is serious cause the docs have some crazy stuff

## Q2 2010

Writing tests in python:

```
class TestCount(unittest.TestCase):

    def test_counter(self):

        c = Counter()
        c.count(1)
        self.assertEqual(1, c)
```

Writing tests in ruby:

```
require 'counter'
describe Counter do
  if "increments" do
    c = Counter.new
    expect { c.increment }.to change { c.count }.by(1)
  end
end
```

- claim: “RSpec is confusing”
  - But not really true
  - Feigned ignorance
- Python is based off of SUnit: 1994

Awesome tweet he made: “**Python programmer rejects without considering its value, Ruby accept without considering it’s value**”

## Instance Variables in Ruby

- Ugly things in Ruby

```
class Horse

  def what
    @mustard
  end
end
puts Horse.new.what # => 5
```

- Really? Let’s take another look...

```
class Horse

  def what
    @mustard ||= compute_it
  end
end
puts Horse.new.expensive
```

- This is how you do memoization in Ruby.
- Really trivial to do something really important
- Some bits being added to Python already exist in Ruby

- Generators
- Decorators

## Summary

- It is really clear in Python why certain decisions were made
- No other language makes the design decisions so clearly
- Ruby’s design is not that hard for a good developer
- Ruby is different
  - More testing
  - Crazy bleeding edge that often doesn’t work
  - Community changes things in weird ways sometimes

## Why does this all matter?

- You can’t evaluate something until you really play with it
- Blocks rock

### 1.12.13 Cherry-picking for Huge Success

- by Armin Ronacher
  - @mitsuhiko
  - <http://lucumr.pocoo.org/talks/>
- Part of the Pocoo team
- Guy behind flask, jinja2, much more
- slides: <http://www.scribd.com/doc/67925053/Cherry-Picking>

## Preface

- Doesn’t care about language fights
- Use the best tool for the job

## Consider Twitter

- 2006: off the shelf Rails application, static HTML, basic XML API
- Now: The API is the service. Website itself is a JavaScript app. Scala/Erlang backend

### Does this mean Ruby sucks?

- Not it does not
- Neither does Python
- Ruby / Python are amazing for prototyping
- Expect applications to change and grow in implementation over time

### Proposed Solution

- Build smaller apps
- Combine apps to make bigger apps

### Cross boundaries

- Pygments is awesome
- Need Pygments in Ruby
  - A: rewrite in Ruby
  - B: Use different syntax highlight
  - C: Use pygments as a service called by Ruby

### “It only does Django”

- You wrote a library that does something useful (thumb-nailing for example)
- Don’t make it depend on Django if you can help it
- Try to make it independent
- Then implement a separate Django app that calls your tool

---

**Note:** This also happens with the Zope community. They did “Zope Only Projects” first. ;)

---

### Protocol Examples

#### Flask Views

- Views can return response objects
- Response objects are WSGI apps
- no typecheck
- Return any WSGI app
- WSGI server doesn’t care if it



## Difflib + Genshi

- Genshi is valid XML
- Difflib returns a string
- Change Difflib to be `<ins>/<del>` so Genshi can render it
- Instant pages!

## Serializers

- pickle, phpserialize, itsdangerous, json
- Within the compatible types it

## Loosely couple all the ways!!!

Many small bits with specific merge points that are loosely coupled

- WSGI
- HTTP
- ZeroMQ
- Message queues
- Datastore
- JavaScript

## WSGI

- Dictionary passed around
- Framework independent, but only for Python
- Tornado and Twisted don't do it, but everything else does
- Middlewares are unused and hard to make
  - TODO: Get his middleware example for use as possible sub-domain hack
- WSGI middleware has issues:
  - Can't consume dform data
  - Processing response from application is complex
  - Can't inject HTML
  - TODO - last bullet?
- Libraries
  - Werkzeug
  - WebOb
  - Paste

## Django & WSGI

- Django used to do WSGI badly
- Getting documented

## HTTP

- Pure HTTP is more work than WSGI
- Easily debugged
- Language independant
- Need syntax highlighting with Pygments but your project is Ruby?
  - Write small Flask app that exposes Pygments as a service

## Libraries

- Python-Requests
- TODO
- TODO

### 1.12.14 Breakdancer

- by Dustin Sallings
  - memcached contributor
  - <http://dustin.github.com/>
- Does a lot of programming languages
- <https://github.com/dustin/BreakDancer>
- <http://dustin.github.com/2010/10/27/breakdancer.html>

## Testing is a boring/hard subject

- How do you find those edge cases?
- How do you detect crazy pairs of edge cases?
- Wait until there is a reported error?
- **pydanny contention: unittest makes it hard to document patterns in tests. User controlled**

## Wrote a framework to help set tests

- Actions are the tests
- Drivers performs the tests and include specific conditions
- So that means your tests are independent of the conditions but defined in code?

---

**Note:** Seems like a design of separation of presentation from content in tests!!! MVC anyone?

---

### MVC test framework? Pydanny Thoughts...

- Can this be implemented on top of unittest?
- If so, it can be back ported to other systems
- Not actually MVC, just a defined pattern
- BreakDance seems to generate code

### Conclusions

- Unit Testing isn't enough
- Find ways to detect and fire off edge case tests
- itertools will save you
- python makes otherwuse tedious tasks boring

## 1.12.15 The Many Hats of Building and Launching a Web Startup

- by Tracy Osborn
  - @lmedaring
  - Founder of <http://weddinginvitelove.com>
- Designer not a programmer
- piloshophy: "An entrpreneur tends to bire off a lottle than he can chew hoping he'll quick;y learn how to chew it"
- project is 'ramen profitable'

### 1. Start off on the Right Foot

- Have a good amount of money in hand to lesson stress
- Be in good health
- Relationships in good shape
- Quit your job cause you can't do this as a side project

### Redefine success

- Don't try to beat google
- Make your goals modest
- Small goals to begin with

### I know HTML! I can program!

- Started in Computer Science but graduated Graphic Design
- Joined a startup as the Designer that grew into a medium sized company
- Got bored
- Wanted to do something that could impact the world

### Tried to find a co-founder

- Why weddings?
- Work on something you love to work on
- Cofounders are awesome, but no cofounder is better than the wrong cofounder
- Things didn't go well with the cofounder didn't go well

### Did it herself

- Learn Python The Hard Way (Zed Shaw) **Great programming starter book**
- Got recommended to use Django because of so many apps available
- Launched weddinglovely.com in 6 weeks!

## 2. Launch as fast as possible

- Don't try to build giant from the start
- Easy to get discourage
- Everything is going to change once you launch - you can't predict the direction of things
- Take out as many features as possible - add them later if you need them

### Work on the hard stuff first

- Programming before HTML layout/design
- Then make it pretty

### Active Learning

- Don't go through a whole book chapter by chapter. Learn the things that work
- <http://gigantuan.net>
- <http://gettingstartedwithdjango.com> got mentioned!

## Launch with bad code

- Don't worry about optimization (speed, caching, code smell, etc)
- get it done!

## 3. Have a plan for monetization

- Have at least a vague plan to making money
- Raising money is a pain in the butt

## 4. Don't be forever alone

- Network and meet people!
  - Surround yourself with experts (in her case - programming)
  - Marketers
  - Designers
  - find the people who can help you do the work you want to do
- NDAs suck
  - Most people won't sign em, so they won't be able to help you with advice
  - Most people aren't going to try and implement your idea
  - Competition isn't necessarily a bad thing
- Ask when you get stuck
  - Don't waste time banging your head against an impossible problem
  - Go ask for help
    - \* Stack Overflow
    - \* twitter
    - \* local meetups
- Surround yourself with good influences

## 5. Take Shortcuts

- Skip the hard things if you can, eventually you'll understand them
- Django is plug & play!
  - <http://djangopackages.com>
  - South for migrations
  - hosted on <https://www.dotcloud.com> (special pricing for startups)
  - launchrock.com

## Summary

1. Start off on the Right Foot
2. Launch as fast as possible
3. Have a plan for monetization
4. Don't be forever alone
5. Take Shortcuts

### 1.12.16 Future of Python and NumPy for array-oriented computing

- by Travis Oliphant
- NumPy
- SciPy
- Array-Oriented Computed
- Enthought is hiring!

---

**Note:** I took Travis' tutorial on it in 2006. I want to use this for serious number crunching. Why bridge out to another language/server if NumPy can do it for me fast and right in Python?

---

## Python fits your brain

**Thesis:** Software engineering today is more about neuroscience than computer science

- Even fits the brains of Scientists
- Engineers say things differently than scientists

```
# engineering solution
from scipy.signal import lfilter, lfilteric
from numpy import zeros

# TODO get values here

def fibonacci(value):
    x = zeros(N)
    y, zf = lfilter(b,a,x,zi=zi)
```

- But this is not fast enough for scientists
  - C speed
  - CPU speed
  - FASTER!!!

## Conway's Game of Life

- [http://en.wikipedia.org/wiki/Conway%27s\\_Game\\_of\\_Life](http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)

### APL: first array oriented language

- 1964
- Descendants still alive: J, K, matlab
- NumPy is a descendant of J
- Crazy non-standard unicode characters
- Very compact
- Can do Conway in a single line of very dense code

### Derivative Calculations

- Complex data can be memory intensive
- Big sets break even list generators
- NumPy can do this for you

### History of SciPy and NumPy

- Travis started in 1997 on Python 1.4
- Early contributors added *numeric* as a Python extension
  - Jim Hugenin (*numeric*)
  - Jim Fulton
  - Paul Dubois
- Fortran still exists because of complex numbers. Most languages got it wrong for a long time, including C and Java.

### Travis Found Python and Numeric in 1997

- Was good at MATLAB but it wasn't efficient
- Loved the expressive syntax of Python
- Loved that slicing didn't make copies
- Love the multiple data types
- Much more flexible than MATLAB
- Loved that he could read source code a year later

### 1999: Early SciPy emerges

- Wanted something more complete than numeric
- A set of libraries and stuff
- Lots of early contributors

## NumPy started in 2006

- Wasn't happy with some of the directions of Numeric
- Got it working after 18 months and the work of 6+ dedicated people

## SciPy Today

- Conferences
- Collection of Tools (NumPy, et al)
- Community
- being looked at by the Financial community

## What SciPy Does

### SciPy

- Lots of cool data shaping tools

### NumPy

- We aren't talking about simple lists but gigantic multidimensional arrays
- Super-duper fast
- Terse but understandable notation
- See *Zen of NumPy*:
  - strided is better than scattered
  - contiguous is better than strided
  - descriptive is better than imperative
  - TODO: finish writing this out!

## Call to Action: Collaboration between Python Core and the Scientific Communication

**Contention:** Collaboration between Python core and scientific developers needs to be tighter

- Index array operator (matrix multiplication is not domain specific)
- Use of slice notation inside function calls
- Array overloading of **and** and **or**
- DSL blocks?



### Call to Action: NumPy and PyPy

- Stop chasing C, start chasing Fortran. Against an example:
  - **Python**: 202 seconds
  - **PyPy**: 4.71 seconds
  - **NumPy**: 5.56 seconds
  - **Cython**: 2.21 seconds
  - **Fortran 90**: 0.8 seconds
- Mock Fortran if you will, but it is blazing fast for some important stuff.

### 1.12.17 Discussions

- Talked to Mark Ramm and Wayne Witzel about SourceForge API

## 1.13 DjangoCon US 2011

### 1.13.1 Keynotes

#### David Eaves

- Professional negotiator
- Professional speaker
- Good, positive message

#### Russell Keith-Magee

- President of DSF
- Django core dev
- Presenting on Django Software Foundation

### Organization of the DSF

#### Board members

- Russell Keith-Magee
- Adrian Holovaty
- Jacob Kaplan-Moss
- Dan Cox (president of Mediaphormedia)

## Officers

- Treasurer: Joseph Kocherhans
- Secretary:

## Committees

- Infrastructure

## Developer Members

- Contributed to Django in a material fashion
- Admissions approved by the board
- Can be anyone “sufficiently material”
- Members can nominate new board

## Corporate Member

- Small: \$500/year
- Medium: \$1000/year
- Large: \$500/year

## What does the DSF do?

- Trademark management
  - The DSF has two lawyers, including a core dev (Justin Bronn)
- stuff

## Manages Ownership of Django

- Django is owned by the individual contributors

## Idan Gazit

### **Title** Designers make it go to **eleven**

- Benevolent Designer for Life of Django
- BDesignerFL
- Took the role in Spring of 2011

## Compromise is the soul of design

- You can't get everything you want
- You have to make choices

## Usability Stuff

- Color charts
- chunking 8001234567 vs 800-123-4567

## Audience

- who
- what
- why
- where

## Critical issues that hurts Django in regards to designers

- Python on windows is a problem
- local setup
- project templates - some basic architecture layouts
- deployment has been a pain point
- trac is sucky

## Glyph Lefkowitz

**Title:** Why does Django hate Python

---

**Note:** Trying to follow Glyph in notes is probably going to fail. He is that awesome

---

- Started with prose
- I come to bury Python, not to praise it
- "Adding manpower to a late software project makes it later." - Fred Brooks
- "Over the centuries native Americans came up with sign languages. Developers have a zillion languages" - Alan
- "Systematically identify top designers as early as possible" - Fred Brooks on getting a software architect in place
- **Give in to your hate** - Glyph on hating things that aren't Python

## 1.13.2 Read the Docs

by Eric Holscher

---

**Note:** arrived late. :(

---

### Intro

- launched in Django Dash 2010
- Makes documentation hosting trivial
- uses sphinx

### Things you can do

- Post commit hooks on Github
- Add custom sphinx theme
- PDF generation via download think
- Use their REST API for links to <http://djangopackages.com>

### CNAME support

- Request for docs.fabfile.org
- docs.fabfile.org > (need to finish this out)

### Architecture

- Python
- Front end caching via varnish
  - Varnish is the current single point of failure.
- Django front ended via gunicorn and nginx
  - Docs are hosted out via nginx
- Postgres SQL
- Haystack and SOLR
- Chef for deployment
- Nagios & Munin
- CoffeeScript (*Where is the Python version? This is only in Ruby*)
- CLI support via <http://rtd.rtd.org> (**need to check this out!**)

## Open source!

Pros:

- Patches
- People trust you most because they can see the code
- BSD license

Cons:

- Known architecture information was on github
- Early version had exposed data like IP addresses and other things

## Hoping it makes people write more docs

- `mod_wsgi`
- `django-piston`

## Lessons Learned

- Think about your URLs. Really hard
  - Adding versions was hard
- Lay your project out sanely
  - started with no tests
  - Shoved code in
  - Racked up a lot of technical debt
  - worked hard to make the project layout a bit more sane
- Write tests!
  - Had a complicated code base without tests
  - They have hosted continuous integration
- Build around a standard tool
  - Lots of good communication between rtd and Sphinx
- Passing data through systems is hard
- Serving static files is annoying
- Log. Everything.
  - Hard to find people's problems until they added sophisticated logging
  - I personally like the build reports
- Promote your projects!
  - Blog
  - Tweet
  - <http://djangopackages.com>

- G+
- Find a designer
- Follow the Unix Philosophy
  - Do one thing and do it well
  - Stay to your project goal and don't deviate
  - RTFD does so well cause all it does is Sphinx documentation hosting
- Have a mission
  - RTFD fixes the problem in open source that projects are not well documented
  - WIKIs are where your project goes to die
  - Sphinx lets you accept Pull Requests

## Sponsors

- Revsys
- PSF
- Divio
- pyladies

## Questions

---

**Note:** ask Russ his question

- How easy to deploy internally?
    - Open Source
    - Documented
    - Chef
  - Designer thoughts?
    - Started with the project with a designer
- 

## Mirror your project in your test layouts

---

**Note:** I love this pattern and use it myself!

- Create test modules following the same file layout as your project
  - Have as few root test utils as possible
    - Use it sparingly
    - just a few simple helper functions!
  -
-

### 1.13.3 Testing: The Developer Strikes Back

---

**Note:** Sandy really, really, really rocked this talk

---

by Sandy Strong

- Been doing Django since fall of 2009
- Pyladies co-founder

#### Hard to do it right

- Delegate responsibilities correctly
- won't always get it right first time
- Presents reasons for refactoring
- increases stability because you can test updates and patches

#### What is unit testing

- A unit is the smallest possible part of an application
- Integration tests the whole and is often built out of unit tests

#### Practices

- Each class, method, and function should have its own test
- Starting Django test.py files are limited
- Organize however you want
- Maintain consistency in your test patterns
- Separate your tests between models, views, and hitting other services
  - keep things simple
  - easy to understand tests
- Don't keep all your tests into monolithic test.py files
- make multiple assertions

#### Mirror your project in your test layouts

---

**Note:** I love this pattern and use it myself!

- Create test modules following the same file layout as your project
- Have as few root test utils as possible
  - Use it sparingly
  - just a few simple helper functions!

- Your tests should copy the model/view/whatever tightly
- 

### Use ObjectCreator classes for mocks instead of fixtures

- Mock your data by using the ORM or whatever persistence your system uses
- Better than fixtures because mocking your objects this way means you are doing an additional ORM test
- The *mock* library is supposed to be good

### Beyond the business logic

- Testing third-party libraries should be separate from other unit tests
- Third party APIs go down. Even the big ones.
- **Mock 3rd party API responses**
- Means you can continue to work when Facebook, Google, et al go down

### Dealing with cache

- Very hard
- I tend to blow away the cache in a `tearDown` method
- Her issues are beyond mine. Sandy rocks!

### Writing tests can improve coding tests

- Small functions can be tested. 200 lines functions cannot
- Write more tests
  - Find good test patterns
  - Functions should perform a single function
  - Units of code should be true to the definition

### What about T.D.D.?

- Step 1 - Write your tests
- Step 2 - Then write the code

### Sandy doesn't believe it exists.

---

**Note:** I've done it for short periods.

- Goes against prototyping
- Requires full team buy0in to really work
- Business owners rarely get it



### Well tested code is often a happy medium

- More realistic
- More practical
- Allows for a more individual style in coding

### Options to get people to test

- webhooks tests to block code that isn't test
- coverage.py makes it a game
- Public shaming!

### Junction between unit and integration

- Difficult areas to test because behavior is driven upon environment
- Some code doesn't always work the way you want because people don't script/document things out

### Testing a virgin codebase: 0-100%

- You may find yourself faced with a project without tests
- Set a pattern for tests, establish a framework follow it and get the team on board
- Smaller tighter tests really help
- Jenkins (continuous integration) is critical
- Test Debt is part of Technical debt
- Enforce the rule that **All future code MUST have tests**

### Graceful code degradation

- Developers need to think outside the box - their local machine is not the same as Staging/Production
- Service unavailable should not be an unavailable site

### With good coverage You can survive these things down...

- Search. 3rd party API, Cache
- Test your dependencies on these things when they are shut down
- This way your site doesn't just die

## Test Infrastructure

- No one gets staging environments that match production
- Run SOLR and RabbitMQ on staging environments
- Don't overdo logging as it will slow everything down

## Useful tools

- coverage
- nose
- one-more

## How to sell testing at your Django shop

- Pretty coverage charts
- *Code not tested is broken by design*
- Saves money when you have problems
- Makes it easier to add features
- Happier developers

## Questions

- If someone breaks a test pattern and won't fix it go back and make their tests follow the pattern
- Interesting to see that Sandy is into Behavior Driven Development

## 1.13.4 Fireside Chat with a BDFL

Moderated by Sean O'Conner and starring **Jacob Kaplan-Moss**

### How did Django get started?

- 2002 - 2003 LJ World hired Adrian Holovaty to become their dedicated Web Developer
- Previous to 2002 had Frank Wiles making things for them in Perl
- Adrian hired Simon Willison as the most overqualified intern until Alex Gaynor
- Because of the fast news cycle at a newspaper you don't have the luxuries of other jobs.
- Sometimes you have to build a project within 24 hours
- Adrian and Simon decided to do it all in Python
- The Python ecosphere didn't have all the tools they needed so they made things up
  - Zope/Plone couldn't cut it
  - No light frameworks existed
- At PyCon 2005 they went to a success story workshop and Adrian demonstrated a blog in 5 minutes

- Nothing else in Python could do it at the time
- People asked them to open source it
- LJ World was okay with open sourcing it
- In July 2005 they released it as open source

### Why did Django get traction and other light Python frameworks didn't?

- Luck of timing
- They did a lot more documentation than any other emerging option
  - Jacob and Adrian focused on the on-boarding of incoming developers
  - Good input from Wilson Minor on design and the result was that it looked better than it's competitors
- Spent a lot of time considering how the community ought to be built

### How has Django's community and structure has changed?

- Jacob mentions there are 6 developer journalists. The reality is that there are a lot more
- Linear community growth until 2006 and then it's grown exponentially
- 20,000 members on the mailing list
- **Huge issue:** Not enough man hours for the leadership to increase the number of contributors.

### What about Django 2.0?

- Jacob never wants to have a **not backwards** compatible version of Python
- Doesn't want to see Django 1.11, wants to go Django 2.0 at that point
- Massive incompatible rewrites are really challenging

### Wishlist?

- Wants to see a better job for formalizing the app interface
- Wants to see Django become a micro-kernel.
  - Less features and more APIs and hooks
  - No more features in Django
- Delighted that **Python Package** has been shaped by the Django community!
- Wants to see more people empowered to help build Django

### How can Django can become a better part of the Python community?

- 2 years ago the communities were a bit separate
- Some tension caused by Django because we as a community sometimes get aggressive
- Some tension caused by Python because the anti-establishment guys don't like winners

- Django needs to help push for better Packaging in python
- Let's port to Python 3!

---

**Note:** I think having the Django community cook up a good alternative to PIL and make it pure Python.

---

### In your role as BDFL have you had to go against the community?

- Jacob/Adrian admit being wrong about the template language not auto-escaping
- Doesn't want to overlord too much

### How have things changed for the Django community since the last hate talk?

- Changed the core developer rules so that the quorum for new core devs could work
- Any time a core developer says “no” to something, they have to explain why

## 1.13.5 Real World Deployment using Chef

by Noah Kantrowitz

- Ops Code guy
- Really good at Python!
- Django hacker
- Ruby user (tolerates it and not enthusiastic about ti)
- Developer

### Who is here

- Sys admins
- Designers
- Developers
- Designers

### How and why

- Infrastructure as good
- Rebuild a system via a script that includes servers and database
  - Configuration management
  - System integration
    - \* Don't use wikis
    - \* don't use spreadsheets
    - \* use code!

## How does thus work?

- Provision
  - Servers
  - Load balancers
  - etcs
- Configure
  - Servers
  - Load balancers
  - etcs

## About chef

- Reasonability
- flexibility
- libraties and primitives

## Bits

- ohai
- chef-client
- chef-server
- knife (command line utility)

..note:: mixed a section here

## ecosysye,

- Apache license 2
- 400 contrib
- open source code
- On github

## Infrastructure

- Recipe
  - Have a type
  - have a name
  - have parameters
  - take actions to change state

- can send notifications to other resources
- Resources
  - resources take action through providers
  - What operating system you are on will determine what action is run

### common resources

- *package* `'apache2`
- *template* `"/etc/apache2/httpd.conf"`
- *cookbook\_file* (Load a recipe that does this sort of thing)

### Idempotence

---

**Note:** what does that word mean anyhow? Ha ha

- Convergence
  - Guard clauses
- 

### Chef Recipes

- Runs just like a script. Doesn;t that make them... scripts?
- Recipes can include other recipes
- Extend recipes with Ruby
- Dynamic configuration through search - so you can search your servers for stuff

### Chef Roles

- Things can be assigned server roles
- Roles describe nodes
- Roles have a run list
- Roles can have attributes

### Other chef terms

- Cookbooks are collections recipes
- JSON blobs
- Other thing 1
- Other thing 2

## Python cookbook

- You can use pip and virtualenv! Yeah!
- *gunicorn::default*
- *supervisor::default*
- Debian-style fot now
- servicer service

## Case Studies - Packaginator

- Inatll users
- configure sudo
- apt-get update
- install gcc

**Warning:** Some of the Chef stuff Noah goes over isn't public yet. Probably in a few days

## notes

- Always run the migrations!
- Apparently we have our reqs in a weird place. I kind of agree
- Old-style custom collectstatic. Need to finish the 1.3 integration!
- <https://github.com/coderanger/djangocon2011>
- Mentions the issues with settings that Jacob Kaplan-Moss taught me.

## 1.13.6 Making the Django ORM Multilingual

By Jonas Obrist

- Lead of the django-cms project: <https://django-cms.org>

### What this is all about

- Models
- The ORM
- Admin
- Forms
- not gettext!

## Que?

- Because you may not understand this title!
- You might lose customers and users

## What does he want

- Multilingual content in the database
- Editable in a usable admin interface
- Easy, Django-like API
- Good performance
  - Most of the existing tools are slow
  - Bad performance

## State of now

- 10 packages at <http://djangopackages.com/>
- No consensus on how this should be done
  - API
  - Base solutions
  - many ideas floating around

## Approaches

- 1 table, 1 extra field
- 1 extra table with key/value translations
- 2 tables, one for translated fields one for translated fields (dual-table) - **How I've done it**
- Translations serialized into a single field (Pickle/JSON) - **No search without a ton of hacking!!!**
- gettext
- Google Translate
  - This is not a serious service for a real project
  - Third party and relies on Google management

## Single Table Approach

### Pros

- Somewhat easy
- few queries
- fallbacks
- Hard to implement filtering



#### Cons

- Multisite this falls apart. Doesn't work
- Migrations are painful because each language requires a schema migration
- Size of query result can get big
- Hard to make nice admin
- Hard to handle required fields

Example:

```
.. sourcecode:: sql
```

```
select book.isbn, book.title_en, book.title_de from book;
```

### Dictionary Table Approach

#### Pros

- Easy to implement

#### Cons

- No filtering
- No sorting
- Admin

No example cause the Query is too big

### Two Table Approach

#### Pros:

- Can be made very fast
- few queries
- Works with south
- makes sense
- possible but hard to make nice admin

#### Cons:

- Hard to implement
- joins
- Usually done with bad performance **I addressed this with caching and celery**
- Incompatible with lots of other packages (requires custom queries unless you are really careful)

## Common problems

- Admin doesn't like new ideas
  - *django.contrib.admin.validation* is a blocker
  - Extensible but not customizable
  - Forms are a weak spot in Django, and Admin uses them in a really odd way
- ORM just wasn't written to be extended, was written to be used
  - Relations: Starting model controls everything
  - Not intended to be changed
  - Nice things: *QuerySet.iterator*
- Performance issues on all of them
- Usually written under time pressure (deadline)
- Many packages are undocumented and lack tests

## Summary: The Situation

- Translatable models are hard
- All available solutions have their problems
- Maybe something needs to be done in Django

## What could Django do?

- Do nothing
- Provide hooks/APIs to make this easier
- Provide support for translated models

## If Django does nothing

- List of multilingual libraries grow
- Many custom undocumented implementations

## If Django adds new APIs

- Probably too low-level
- abstract solutions add overhead

## Django adds multilingul support

- Bikeshedding potential
  - What approach to take?
  - What API should look like?
- Easiest way to implement
- Could be done backwards compatible

## 1.13.7 Why the Django Documentation Sucks

by Steve Holden

- PSF chairman
- Rambles

### Rambles

- All documentation sucks because the mind of the writer can't match the mind of the reader
- Use cases mostly appear to be "I want to know about X"

### Use cases

- Documentation wasn't necessarily done with use cases in mind

### Images and pictures!

- No pictures? **A picture is worth a thousand words**
- Some people need visual pictures to process things - visual thinkers

### No jokes?

- Jokes cut through barriers and allow people to interact more intimately
- Humor negates fear
- But you run the risk of looking silly

### Problem: overview

- google "django overview" <https://docs.djangoproject.com/en/dev/intro/overview/>
  - shouldn't a glance be visual?
  - Gigantic document

### Problem: Tutorial

- Put your apps in project subdirectories
- It's like they'd never heard of the Python PATH
- `manage.py startapp` *still* does it that way.

### Problem: SEO optimization

- Why isn't the first Google hit on every 'django' somewhere in the docs
- Problem: Curious noob gets odd things
- django users doesn't return good results

### Problem: People might get smug about Django docs

- Because they have become smug

### Solutions

- Make documentation submissions process easier
- Ask for all doc submissions and reserve the right to edit

## 1.13.8 Best Practices for Front-End Django Developers

by Christine Cheung

- One of the co-founders of Pyladies
- Works for Red Interactive Agency
- Does front end development
  - Also does a lot of backend stuff
  - Quickly mastered CBVs so she has serious chops
- @webdevgirl on <http://twitter.com/webdevgirl>
- <http://xtine.net>

### Presentation is important

- Polixh front-end is as important as the front end
  - It may “scale”
  - But bloated markup and JavaScript will slow things
- The implementation really matters

## Start Organized

- Structure the templates
- Template Tags should express presentation, not logic
  - Presentation and iteration over data, NOT manipulation

---

**Note:** I wish people remembered this bullet about template tags and logic

---

## Cascading Style Sheets

- Define a style guide. **Write it down!!!**
- Consistent Variable naming
  - Camel case
  - dashes
  - underscores
- Keep your css files small. Combine it in deployments

## JavaScript

- Use a framework!
- Pick one and stick to it:
  - JQuery
  - Don't mix in other things like Dojo
  - Avoid plug-in overkill. No more than 3-4
    - \* Reduce performance hits and code conflicts
    - \* Analyze if you can write your own
- Namespace your Javascript
  - TODO - get sample code from Christine
- DON'Ts:

```
<script type="javascript">
  document.write('foo');
</script>
<a onclick="blarg();">Stuff</a>
```

\* Lots of JavaScript? Use backbone.js

## Don't do HTML from scratch: Use html5boilerplate

- Comes with a layout via 960
- JQuery

- Modernizr
  - You can do wicked class tricks with this tool. Wow
  - Need to really look at the slides cause she did this better than the docs

### Sass/Compass instead of CSS

- CSS Authoring Framework + Utilities
  - SASS - nested rules, variables, mixins
  - Image Spriting
- <https://github.com/Kronuz/pyScss>

### All about the data

- Leverage the power of both the front and back end
  - Share the work between the front and back side of things
  - Generic Class Based Views for quick prototyping

### Define you datatypes

- Make an API
- Share the models between back and front end

### Tests

- CSSLint
- JSLint

– **Warning:** it will make you cry

- Google closure compiler

### Performance

- Minify
- Control this via a *settings.py* value

### Documentation

- Consistent **folder structures** and **document style guides**
- Does Sphinx autodoc doesn't work with docstrings in JavaScript or CSS?
- Use a **JavaScript library** and modern authoring **techniques**

## Wrap Up

- Leverage data loading between front and back ends
- Use **HTML Boilerplate** + **CSS/JS** tools to your advantage
- Test and performance check the front end!

### 1.13.9 Security

---

**Note:** Had some things to do so did not get a chance to focus on this talk

---

by Paul McMillon

- Security expert since 1998
- Just now a core developer of Django

#### SSL - safe and secure

##### Are you doing it right?

- Standard way is to redirect from HTTP to HTTPS
- That doesn't stop images on HTTP from sneaking in cookies
- Use HTTPS secure cookies!
- Use HTTP Strict Transport Security (HSTS)

#### Use django-secure

- Identifies these issues

#### Don't put your database in your github repo

- Django needs better hashing
- Even on open source projects

### 1.13.10 Scraping the Web

by Katharine Jarmul

- Co-Founder and VP of PyLadies
- [twitter.com/kjam](https://twitter.com/kjam)

## Content is what you need

- You need to get content
- good content exists all over the web
- Scrape it!

## lxml

- lxml.etree good for good formatted xml
- fast for SOAP or other xml-formatted content

```
def parse_feed_titles(rss_feed):  
    data = []  
    dohtml = html.document_fromstring(rss_feed)  
    for x in dohtml.cssselect('title'):  
        data.append(x)
```

## lxml: cssselect

Uses a JQuery style language to grab bts

```
article_title = html.cssselect('div#content h1.title')
```

## cssselect

- Learn css to help you scrape
- Various developer tools help to find stuff (firebug)

## iterlinks

- Good for high link pages or finding related links

```
page_links = html.iterlinks
```

## sourceline

- tells you the line the element is on
- Helps you determine distance of one element is from another

```
pos = element.sourceline
```

## find,findall

- Find the element you want
- Grab only what you need
-



```
spans = element.findall('span')
```

### nodes of content

- Elements have children
- Elements have siblings
- Elements have ancestors

```
h1s = list(h1_element.itorsiblings())  
the_kids = [c for c in element.iterchildren() if len(c.text)]
```

### Web pages change

- All your code can break
- Make a monitoring system to let you know it things change
  - Code that checks the pattern of the layout

### forms

- Think: log in pages
- Don't use **lxml** for evil
- See lxml docs on how to process them

### text, text\_content, iter\_text

```
text = element.text  
text_w_content = element.text_content()  
text_bit_by_bit = list(element.itertext()) # The best way!
```

### Tips for maintainable scrapers

- Skip ugly parsing
- Text = Content = Boss

### XPATH fundamentals

- Not fun but you need to learn it to handle XML. I hate it
- lxml supports it, of course

## Building LXML w/LXML

- Writing templates for things like xml are boring
- **Don't be that guy/gal**

---

**Note:** Not showing the example because kjam says it is evil!

---

## Tweepy Innards

- If you API lib just returns the API with no frills, that's not really helpful
- IF API data is fairly standardized, do nice thngs like create models
- "Don't make me convert datetimes or I'll put a nasty mark on.djangopackages.com about your project" - @kjam

## Other tools

- Feedparser
- HTMLparser
- re
- html5lib
- mechanize
- Feedparser for parsing RSS or Atom
- Sometimes you want a lighter tool
- Sometimes LXML doesn't install easily

## Don't forget

- Content is 1/2 the equation
- Ugly web pages with good content is lame
- Work with your front end people
- Find some good designers and befriend them

## Read up on these!

- <http://pypi.python.org/pypi/requests> library for pulling data
- <http://lxml.de>
- <http://wwwsearch.sourceforge.net/mechanize>
- <http://code.google.com/p/html5lib/>
- <http://scrappy.org/>

### 1.13.11 Cache rules everything around me

By Jacob Burch

- Engineer at Revsys with me
- Former CTO of Mahalo

By Noah Silas

- Engineer at Causes.com
- Former head architect of Mahalo

#### Intro to caching

- Caching is post-process data
- Stored in a key/value store
- Usually Done to:
  - Speed up your app
  - Lesson load on third party apps

#### Big Picture

##### Jacob's rule of architecture

- No rules only principals
- Start with assumptions/advice
- Benchmark/inspect before you break principals

#### Ask some questions

- Do you really need caching? Caching adds...
  - complexity
  - additional point of failure
- Modern database are stupidly optimized
  - May be all you need!

*There are only two things in Computer Science, cache invalidation and naming things* - Phil Karlton

#### Relying on your cache always being up

- Rely on a single canonical data source
- This data source IS NOT YOUR CACHE

## Common Cache Patterns

- Rollup values often called like settings
- Only do it for common data calls
- What's easy:
  - cache invalidation is relatively easy
- What's hard
  - Known when to invalidate

## Thundering herd problem

- If everything in your cache tries to reload at once because you had a service outage
- Huge load on your system and third party apps
  - You may get throttled by other systems

## The New Hotness Pattern

- Cache Forever, Invalidate, explicitly
  - Keep the cache forever
  - Invalidate only when the data changes
  - Add the data back right then and only for that bit
- Use celery or deferred to handle long processes

## Key-holding

- How about storing cache keys in a file?

## How to avoid Cache Invalidation Hell

Yay!

## What's in the box

- `django.core.cache`
  - simple setup
  - Multi Cache Support
- `django.core.cache.backends` (best way)
  - `memcached.MemcachedCache` (works with PyPy)
  - `memcached.PyLibMCCache` (faster but on C)
  -

- View decorators are interesting because they cache HTML
  - But you don't know cache key names so that makes invalidation hard
- *{% load cache %}* DON'T USE THIS!!!
- django.core.cache
  - settings.CACHES[alias]['VERSION']
  - settings.CACHES[alias]['KEY\_PREFIX']
  - settings.CACHES[alias]['KEY\_FUNCTION']

```
def make_key(key, prefix, version):  
    return ':'.join([prefix, str(version), smart_str(key)])
```

### Good cache keys

- Make them unique
- Use separators that don't appear in your values
- Don't ever write the same format string once

### Key files!

- Don't define the same key format string in more than one place
- Put all the app key names into one file

---

**Note:** What I tend to do is as follows

---

```
# in core/cachekeys.py I store all my cache keys in functions  
def profiles_profile(user):  
    return "profiles:profile:{pk}".format(pk=user.pk)  
  
# Then I refer to the key thus in a models file  
from django.contrib.auth.models import User  
  
from core import cachekeys  
  
class MyModel(models.Model):  
  
    user = models.ForeignKey(user)  
  
    def get_profile(self):  
        key = cachekeys.profiles_profile(self.user)  
        profile = cache.get(key)  
        if profile is not None:  
            return profile  
        profile = self.user.get_profile()  
        cache.set(key, profile)  
        return profile  
  
    def save(self, *args, **args):  
        profile = self.user.get_profile()
```

```
cache.set(key, profile)
# TODO add superclass call here
```

### Cache related code stuff

- try MyModel.cache as a manager
  - `User.cache.get_top10_users.all()`
  - This is really djangsta!
- invalidation:
  - try a post\_save signal to separate concerns. Hrm...
  - Using signals can make cache invalidation hard
  - They have to do model.save() magic to make their system
  - QUESTIONS: Aren't they losing that separation? Why not a save() override instead of signals? Wouldn't it be less engineering to just put it all in the save()?
    - \* The answer is that they are breaking principals
    - \* Not a clear answer on what they are getting out of post\_save stuff then

```
# TODO - Get their OriginalStateModel code
```

### Third Party tools

- TODO - get this list!

### Last minute Advice

- Don't let your cache servers be accessible
- consistent hashing is a neat trick – use it!

### DoesNotExist Deserves Cache love

sfsf

### 1.13.12 Django Core Dev Panel

by cast of thousands

- Alex Gaynor
  - First code was a test
  - Isn't excited by the reinvention of things
- Andrew Godwin
  - Wrote a breakout clone
  - Finds web based games interesting

- Carl Meyer
  - Ice hockey
  - first code was a patch
  - likes playing with Haskell
- Chris Beaven
  - async fan of eventlet, node, etc
- Gary Wilson
  - Learned on TurboPascal in high school
- Gabriel Hurly
  - First wrote a log parser in Perl
  - Open Stack fan!
- Idan Gazit
  - finds backbone.js exciting
  - Doesn't like sproutcore cause it tries to change your system entirely
- Jacob Kaplan-Moss
  - BDFL
  - Farming
  - Apple ][ basic code just like me!
- James Bennett
- Joseph Cochrane
  - Riak
  - Geographical data into distributed systems
- Julien Philip
  - Soccer, Rugby
  - First code was a calculator
- Justin Bronn
  - Linux kernel hacking
- Karen Tracy
  - Played with Basic on the IBM PC
  - Rescues kittens
- Paul McMillon
- Russ Keith-Magee
  - His little kids rock
  - Wrote a line based basic game on the Commodore 64
  - Excited by the change in the platform

## What about App Refactor?

---

**Note:** Zoned out and missed this one. Argh

- infrastructure API has been cooked up
  - configurable foreign key
- 

## New community roles

- Security Manager: Makes Django to be more secure
- Front end BDFL: Makes Django prettier
- No one should be irreplaceable

## How about Django on other interpreters?

- Alex Gaynor is also core dev on CPython and PyPy and says it works
- JKM asks about any issues with Django on Iron Python

## Talking on django-dev

- be decent
- Don't just *+I* or *-I* that doesn't give them any knowledge
- Explain why you are sharing your opinion

## Code of conducts

- JKM: Django community ought to have a written code of conduct
- The Django core dev leadership is very approachable

## How can the Django third-party ecosystem be managed?

- Miguel Araujo wanted to know if we can manage the community of third-party world. Cause some packages are just unmaintained
- Possible ideas:
  - unmaintained code could fall out of ownership
  - How do you maintain/control a community of volunteer projects?
  - Core devs are already overloaded
  - The DSF can give Django Packages and other projects more support



### 1.13.13 sprints kickoff

#### 1.13.14 General Notes

- slides: <http://lanyrd.com/2011/djangocon-us/coverage/>
- Remember that I'm engaged to Audrey Roy.
- Flew up on the PyLadies Party Plane on September 4th on just 90 minutes of sleep.
- Taught the Ultimate DjangoCon Tutorial Workshop. Thoughts:
  - Not having enough time to QA the content
    - \* Having errors in your slide code can be good
    - \* the class pulled together and we debugged as a team.
    - \* Maybe do this on purpose in the future
  - Mentors rock
  - The existing tutorial teaches some anti-patterns. :(
  - I think there was no confusion until the very end. Which is really good!
  - Next time I will do better

## 1.14 Kiwi Pycon 2011

### 1.14.1 Talks

#### Keynotes

##### Python in the Web: Can we keep up?

By Audrey Roy

*I'm partial but I think she did a great job of pointing out where Python might/should go.*

#### Metaprogramming

By Jeff Rush

*"Metaprogramming is the programming of programming"*

---

**Note:** Apologies, but I couldn't keep up with the firehouse of knowledge. This is woefully incomplete.

---

**Warning:** This is from me: Metaprogramming should be done carefully. Only use it when you have no other options!

### This talk makes use of

- metaclasses
- decorators
- attributes
- more

### What is metaprogramming orientation

- Code **Manipulates** Data
- Data **Feeds into** Code
- Metaprogramming sits about the Code/Data
  - Add/adjust elements
  - register elements
  - tag elements
  - event management
    - \* Pre/post initialization
    - \* read/write
    - \* call-and-return

### Addressing a problem

```
class Request(object):  
  
class HTTPServer(object):  
    def handle_request(self, ...):  
        req = Request(...)
```

- objective
  - subclass a class deep inside a module
  - Requires rearrangement content of the module

```
old_import = sys.modules['__builtin__']  
sys.modules['__builtin__'].__import__ = self.__import__
```

### Sample code

```
class Member(object):  
    __metaclass__ = DatabaseTable  
  
    dbtable = "Members"  
  
class DatabaseTable(type):
```

```

def __init__(cls, name, bases, class_dict):
    col_defs = db.query_cols()
    for col_def in col_defs:
        db_column = wrap_col_rdwr(col_def)
        setattr(cls, col_def.name, db_column)

def wrap_col_rdwr(col_def):
    def get_dbcol_value(self):
        return self.__dict__.get(col_def.name, None)

    def set_dbcol_value(self, value):
        value = col_def.validate(value)
        self.__dict__[col_def.name] = value

    return property(get_dbcol_value, set_dbcol_name)

```

### Meta classes? class decorators

- The latter are much simpler
- The latter can do almost everything the former can
  - only a metaclass can add methods to the class itself
- class-level services (methods)
  - @classmethods provide them to instance
  - metaclass methods provide them to the class itself
- only a metaclass can add to class attrs not visible to self
  - meta-methods
  - meta-properties
- Class decorated can be (more easily stacked)
- MISSED BULLET

### Example using class decorator

```

def CallTracer(attr):
    """ Do custom logic stuff here """
    return attr

def tracecalls(cls):

    def my__getattr__(self, name):

        attr = super(cls, self).__getattr__(name)
        return attr if not callable(attr) else CallTracer(attr)

    cls.__getattr__ = my__getattr__
    return cls

@tracecalls

```

```
class MyClass(object):  
    pass
```

## Diving into attribute manipulators

The talk dived a bit into things like:

- `__getattr__`
- `__getattr__`

Diving into this sort of code is tricky, because the reasons for use of these tools is not necessary in 99% of Python projects. I prefer to rely on *decorators* to alter behavior because they are syntactical sugar. Easy to find and very explicit.

## relate or !relate

by Mark Ramm

- “*Python has been very good to me*”
- Double major in Philosophy, Literature and minors in Theatre and something else
- Wrote the TurboGears 1 book and became a web developer

## Opening

- Tools matter
  - Python is a great tool
  - “SQLAlchemy is the best ORM on the planet regardless of language”
- Data matters just as much

## Tools matter

- Know your tools
  - Screws and nails
    - \* deck (screws are stronger than nails, but harder to use)
    - \* siding (nailing yourself to the siding)
    - \* mongodb (nailed the SourceForge team to the wall via MongoDB)

**Takeaway:** If you don’t know your tools when you hit production you are going to be trying to debug something critical.

## SQL

“Who here has used a non-relational database? Not many? Oh, there’s this thing called a *FILESYSTEM*.”

## ACID

- Atomicity
- Consistency
- Independent
- Durable

## Why I NEED relational

- I can't use NoSQL because
  - It's financial data (need consistency)
  - my data is relational

Mark says **BULLSHIT**

- Amazon tracks financial data using NoSQL
- You can make things more durable and consistent if you know what you are doing

## Reason to use SQL

- Well known, easier to find people who know it
- Robust, scalable, flexible, simple
- pretty ACID

## Really good reasons to use RDBS

- DSL for ad-hock queries understood by everyone
  - Business and marketing types can run their own queries
- blah blah inconsistencies blah blah latency
- Being down is better than being wrong
  - Critical for medical applications!

## NoSQL

Many different types:

- key/value store
- distributed key/value store
- column oriented stores
- map-reduce store/system
- document oriented store
  - MongoDB

- CouchDB
- XML (now we use JSON instead of this)
- ZODB (A really good system in Python - just so long as you don't want to share your data with other languages)
- graph oriented stores

## CAP theorem

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- Consistency
- Availability
- Partition tolerance (the system continues to operate despite arbitrary message loss)

Google and Facebook scale has to handle downtown gracefully so has to choose which of these three things they'll focus on

## Focused

NoSQL systems tend to have a few but not all of the following bullets:

- Scalable
- Simple
- Fast
- Flexible
- Topic

## It's WebScale

Mark calls **BULLSHIT** on WebScale

- 99% of sites don't care about this issue
- 99% of people are okay if their sites have a feature that fails on a continent
- Think about what you actually need
  - Don't implement a database just cause it's cool
- merciless.sourceforge.net

SQL version:

```
select * from document where x=3 and y="foo"
```

MongoDB version:

```
b.things.find({x:3, y: "foo"});
```

## Conclusion

- Figure out what YOUR app needs
  - SQL, MongoDB and Cassandra is great at some things
  - Mark couldn't answer this for Hadoop
- Don't obsess about SCALE you'll never achieve
- Launch your project, then worry about scale

## Giving your website a command line interface

by Michael Hudson-Doyle

### About the presenter

Works for Canonical and Linaro.

### Enter Linaro!

The ARM ecosystem is very fragmented and the kernel has a lot of copy/paste code.

Linaro is a not-for-profit software engineering company investing in core Linux software and tools for ARM SoCs.

see: <http://validation.linaro.org/>

### Why a command-line interface?

- We want to do things like trigger test runs when a kernel build finished
- This basically means some kind of Remote Procedure Call (RPC)

### Need security

- The boards in our lab are a limited resource
- Some risk of mischief
- Eventually may have test results from unreleased hardware

### Protocol Choices

- They use XML-RPC
- Didn't think about it very hard
- will probably use JSON-RPC

## Authentication

### OAuth

- Hard to implement
- Hard to consume
- Doesn't sign the body of the request in XML-RPC

### Used Encryption instead

- Use HTTPS
- Followed RFC 2617 Basic Authentication
- They provide API tokens
- Similar to my Storymarket API work

### Live the Server Code

- <https://launchpad.net/linaro-django-xmlrpc>
- <http://djangopackages.com/packages/p/linaro-django-xmlrpc/>

```
from linaro_django_xmlrpc.models import ExposedAPI
from linaro_django_xmlrpc.globals import mapper

class ExampleAPI(ExposedAPI):

    def whoami(self):
        return self.user

mapper.register(ExampleApi)
```

### Live the Client Code

at: <https://launchpad.net/lava-tool>

```
# TODO find a code example
```

## Conclusion

- Don't try to be clever - just use HTTPS and Basic Auth

## Introduction to Matplotlib for Data Analysis

by Catherine Thwaites



## Why Matplotlib?

- Great way to visualize complex data
- Free
- Fast
- Works on any OS

## Dependencies

- Python 2.5., 2.6, 2.6, 3.2x
- Numpy 1.3+
- Matplotlib 1.0.1

## Install

Linux: <http://matplotlib.sourceforge.net/users/installing.html>

Windows: Download and install

## On-line Examples

Huge gallery of examples!

- <http://matplotlib.sourceforge.net>

## Ways to run matplotlib

- Interactively with pylab and python
- Interactively with the shell
- Normal Python modules
- Some other way?

## Simple bar graph using bar

```
import numpy as np
import matplotlib.pyplot as plt
data1 = [12,23,38,42,41]
figure = plt.figure(1, (6,6))
figure.clf()
ax = fig.add_subplot(111)

ind = np.arange(len(data1))
rects = ax.bar(ind+0.125, data1, width=0.75, color='thistle')
plt.show()
```

Clarified versionL

```
import numpy
import matplotlib.pyplot as plot
data1 = [12,23,38,42,41]
figure = plot.figure(1, (6,6))
figure.clf()
axis = fig.add_subplot(111)

ind = numpy.arange(len(data1)) # what is ind representing? An index?
rects = axis.bar(ind+0.125, data1, width=0.75, color='thistle')
plot.show()
```

### You can do more!

- titles
- plot range
- Axis labels
- Axil ticks and labels
- Add bar labels

### Things to think about

- Somewhat challenging learning curve
- People with lots of money can't understand why you aren't using Matlab

### Python distributed programming using gevent and redis

by Alex Dong

- [trunk.ly/?q=from:alexdong+gevent](http://trunk.ly/?q=from:alexdong+gevent)

### Roadmap

- Crawler: the unsung hero
- Async 101
- Gevent: the monkey king
- Redis: data structure server
- lessons learned

### How many links does google index?

- 18 million when it started
- Only 2-3 billion right now
- First project google employee worked on was the crawler

## Talking about a crawler

1. Get a url from a task queue
2. DNS resolution
3. Request HTTP Header
4. Download full content
5. Store to local file store, database and index

Add in scheduling, throttling, status monitoring, scale up by flicking on more servers.

## Async 101

- Whats wrong with multi-thread
  - GIL issues
  - Yield on IO/socket, but
  - Computational expensive will block
- What about multi-process?
  - Memory efficiency
  - Context switch overhead

The overhead of multi-process in Python causes a lot of server load.

## Another way

- controller + worker model
- Cooperative multitasking
- Some unix code:

```
epollfd = epoll_create();
epoll_ctl(epollfd, EPOLL_CTL_ADD, listen_sock, &ev)
epoll_wait(epollfd, events, MAX_EVENTS, -1)
```

## gevent - Monkey King and Pool

Monkey patches python and magically makes multi-processing work.

```
from gevent import monkey
monkey.patch_all() # patches the Python magically

from gevent.pool import Pool
worker_pool = Pool(size)
# get domain into payload
pool.spawn(socket.getaddrinfo, payload)
```

Question: Whats the downside?

- Alex says that it makes debugging harder

- Hence the lesson of making a dashboard!

### Redis - Data Structure Server

- High performance 15,000 req/sec
  - Lock free, single process
  - master/save ready
- Data Structures
  - FIFO queue: Lists - LPOP, RPUSH
  - Working hashtable - HSET, HDEL

---

**Note:** lots more I didn't get in!

---

### Lessons Learned - Dashboard

- Turning point: Most important code we've written
- 25% code for status update and monitoring
- What's causing the piling up?
  - Someone abusing the system?
  - DNS is down?
  - ISP's bandwidth?
  - Large file download?
  - Scheduler re-submit tasks?

### Lesson Learned - Fine balance

- Conflict between frontend and backend
- Capacity planning

Example: *If the worker takes too long to return control you can block your system*

### Lessons Learned - Use Profiler

- Structure the code to make it possible to run all steps in one non-gevent enabled process
- Carefully profile to make sure *socket.recv* becomes the main bottleneck
- Rule of thumb *load average* < 1 to saturate 10M Bandwidth

Question: Where they using regex to parse HTML?

## Behaviour Driven Development

By Malcolm Tredennick

- Slides at <http://github.com/malcolmt/bdd-and-testing-talk>

### Why have I asked you here today

- Testing landscape
- A different way to think about this
- Code

## Intro

### Data

- Stubs is empty data
- Mocking is pretend versions of data

### Mocks

- Substitute for external systems & components
- Need to be injected into the tests
  - “monkey patching”
  - Dependency injection
- Problem: Mocks need to be kept in synch with reality

**Warning:** I’ve run into this with mocks. This is why Open Comparison includes real API calls to it’s target systems.

## Testing Strategies

- formal verification
- test everything you can think of
- anti-regression suite
  - no bug fix without a test
- Test Driven Development (TDD)
  - Write failing test for next method or class
  - Write minimal code to make test pass
  - Rinse, wash, repeat
  - Kind of drives Malcolm crazy

- Really about “*specification, not verification*”
- Documentation Driven Development (DDD)
- Behavior Driven Development (BDD)

## BDD

Here we go!

### Specification

- On paper
- Capturing the idea is probably good
- “What was I *thinking* when I cooked this up?”

```
As a [user or role]
I want [feature]
so that [something happens]
```

## BDD

- What is the next most important thing the code should do?
  - Need to be able to test
  - Need to be able to run

### Scenario

```
Given that [initial context],
if and when [some event occurs],
then [ensure some outcomes]
```

### Interlude: Naming things

- Language helps guide our brain
- Focuses the mind
- Perhaps too exclusively

### The Sapir-Whorf Hypothesis

---

**Note:** Read [http://en.wikipedia.org/wiki/Sapir-Whorf\\_Hypothesis](http://en.wikipedia.org/wiki/Sapir-Whorf_Hypothesis)

---

## Test Titles

```
class BasicFieldTests(TestCase):
    def test_field_name(self):
        ...
    def test_show_hidden_fields(self):
        ...
```

Better:

```
class Fields(TestCase):
    def show_allow_name_override(self):
        ...

class ChoiceFields(TestCase):
    def should_permit_initial_values_in_hidden_widgets(self):
        ...
```

## Excercise

- write a *load\_test* that changes the *test\_* prefix with *should\_* prefix
- Cause language matters

## Closing thoughts on unittest

- Create more test classes
- Make them more explicit

## DSL Packages for testing

- Lettuce (via PyPI)
- Freshen (via PyPI)

These tools are unique and probably good showing to non-coder types with money.

## Conclusions

- Behavioral tests are worth trying
- Question the way you think from time to time
- Are your tests' purpose clear to the *future you*?

## Those niggly beginner's questions answered by a code craft ninja

by Tim Penhey and Rachel Penhey

---

**Note:** I love going to *beginner* Python talks. You often learn or relearn nice tricks you can implement today.

---

## Audience Design

- Aimed at Python beginners
- Aimed at programmers beginners
- Audience questions at the end
- Won't take the full hour unless there are questions later

---

**Note:** Had to leave early. :(

---

## Lightning Talks

### Web Scraping

by Chris Esther

- <http://pypi.python.org/pypi/selenium>
- <http://pypi.python.org/pypi/mechanize>

### Crowdsourcing from NZ

by Ben Healey

- Rant about Amazon US not allowing the outside world to participate because of terrorists
- Then talked about various services like <http://crowdfunder.com/>

### Web Socket

by Alexander Abushkevich

- Provides mechanism for bi-direction data exchange between server and clients
- Seems to more practice than XMLHttpRequests for long polling
- many python libraries

---

**Note:** I think Zed Shaw has ranted about web sockets. Need to ask him.

---

```
window.onload() {  
    // TODO is this how this really works?  
    var s = WebSocket('http://blarg.service.org/service')  
    s.do_action  
}
```



## Google App Engine

by Greg Fawcett

- Google will be charging for it soon
- old version from Django
- Lots of frameworks ported there: *Web2py*, *Zope*, old versions of *Pyramid*
- Sends out inbound and outbound email
- NoSQL database BigTable
- Runs with a Google version of memcached. Question: How do you ask for it?
- Has scheduled task system
- Google App Engine apparently ran the Royal Wedding website

## Wiki-to-speech Robot Presentations

by John Graves

- Converts presenters notes to speech
- This way it will convert your slide shows to full presentations

## Pythonic Slide Generation

by Leon Mathews

- Wrote some restructured text
- Using rst2pdf to generate slides
- Created a slides.style file to define the results better
- Generated PDF slides

---

**Note:** TODO - ask him to provide source code and instructions

---

## Valgrind Memcheck

by Tim Penhey

- “*Humans are bad at finding memory leaks*”
- Not really a python talk, but rather a review of how to check for memory leaks.
- Created an alias to run his valgrind check with all the arguments already defined

## Sorry state of the New Zealand Job Market

by Juergen Brendel

- At NZ bookstore only could find 1 book on Python and one Django 0.96 books
- Need more Python jobs
- Need to expand the community

## What Parkour Taught Me About Programming

by Danver Braganza (Organizes his clothes into an array)

- You learn by doing
- You don't become a good developer by wishing you were a developer and watching the Social Network
- The best practitioners get better by taking their work home with them.
- Learn how to apply your knowledge to other avenues

## Dependency Management in Python with Buildout

by Brad Milne

- They control the environments very tightly so they don't run into the problems I've had.
- His examples also seem much cleaner than the sort of thing I've suffered through

## Just because you can, doesn't mean you should OR An evil web framework/thing

by Aurynn Shaw

- Generators
- Generators.send()
- Wrote a web framework that uses send to store sessions
- Runs on twisted
- But has a race condition
- Silly prototype
- code: <https://github.com/aurynn/evil>

## Qtile: A Python Tiling Window Manager

by Aldo Cortesi

- A Windows layout tool for Python. He also presented this at PyCon AU.
- <https://github.com/cortesi/qtile>

## Job security in Python

by Chris Neugebauer

---

**Note:** This is going to hurt my talk but it's funny

---

- “Readability” is not a virtue if you want job security
- PEP 0008 is not a good style code if you want to keep your job
- Contention: Readability sucks
  - Other people can comprehend your code
  - You can maintain your own code - less billable hours
  - Your code will be more

## How do you write unmaintainable code?

- Bad variable names
- Metaprogramming
  - Breaks the help function
  - all kinds of ways to bury logic
- monkey patching
  - Roll your own standard library

## Porting (and staying ported) to Python 3

by Leon Matthews

---

**Note:** presenter had slides with gradient background so things were hard to read. Sigh.

---

## Overview

- Why bother?
- Possible strategies
- Porting to Python 3
- Maintaining Python 2 **AND** Python 3
  - Two concurrent code bases (Python 2 AND Python 3)
  - IF/ELSE logic for imports/code

## Why Bother?

- Python 3 is the future of Python
- Industry leading unicode support
- It's nicer - Python 2 after wart removal
- Other people may be waiting on you

## Sample

```
# python 2
print 1, 2

# python 3
print(2, 3)
print(2, 3, file=sys.stderr)
```

## Porting to Python 3

Already there if:

- You can run under Python 2.7
- You have handle on Unicode

## Lots of issues

- Syntax renames
  - urllib2 to urllib
  - print to print()
- Changed behavior
  - Sorting comparisons are different
  - division
- unicode
  - All strings in Python 3 are Unicode
  - Initial transition can hurt but then it gets easy

```
import io
io.open(path, 'rt', encoding='utf-8')
a = u'Unicode String'
b = b'Binary string'
```

## You now have a working port, now what?

- Two code bases

- Maintain both
- Leave one to die in the cold?
- Way too common a situation

## Django for n00bs

by Jen Zajac

- Works for Catalyst
- Uses Sublime 2. So do I!

## Awesomeness

Jen educated me on the use of '+' in related names. How cool is that?!? \* [https://docs.djangoproject.com/en/1.3/ref/models/fields/#django.db.models.ForeignKey.related\\_name](https://docs.djangoproject.com/en/1.3/ref/models/fields/#django.db.models.ForeignKey.related_name)

## Introduction

Quotes:

- *"Django is the web framework for perfectionists for deadlines."*
- *"A framework is something you can hang stuff off of - it is a structure."*

## Frameworks vs CMS's

Frameworks:

- Frameworks take longer to set up
- Frameworks are for specific applications
- Frameworks only do what you plan them to do
- Few wasted features

CMS:

- Take less time to set up
- Broad in design
- Does everything
- Lots of wasted features

---

**Note:** Did you know that Django was extraced from the Ellington CMS? So the next time someone tells you that Django isn't suited for CMS work, show them <https://www.django-cms.org> and tell them that <http://science.nasa.gov/> is powered by Django (used to be Plone but we rescued it).

---

## Django backstory

- Started in 2003 at LJ World
- Open sourced in 2005
- Hit 1.0 in 2008
- Other frameworks
  - Pyramid (awesome)
  - Zope (over engineered)
  - Bottle (I prefer Flask)
  - TurboGears (IMHO deprecated until it is converted to Pyramid)

## Pros and cons of Django

Pros:

- Big community
- Tightly integrated components
- Built-in interface
- Documentation
- Release process
- Authentication & Security

Jen Cons:

- Not playing so well with others

Danny Cons:

- No defined best practices by Django core devs
- Documentation has stagnated in places
- Documentation could use better organization
- Needs to be better at explaining itself as not a framework for programming beginners
- Some WSGI weirdness that is being resolved shortly

## Sample model code

```
from django.db import models
from django.contrib.auth.models import User

class Movie(models.Model):
    title = models.CharField(max_length=100)
    genre = models.CharField(max_length=100)
    description = models.TextField()

class Attendee(models.Model):
```

```
# see https://docs.djangoproject.com/en/1.3/ref/models/fields/#django.db.models.
↳ ForeignKey.related_name
user = models.ForeignKey(User, related_name="+")
```

## Sample view code

```
def home(request, template_name="movies/home.html"):

    movies = Movie.objects.filter()
    data = {'movies': movies}
    return render_to_response(template_name, data, RequestContext(request))
```

## Quick overview

- templatetags
- rss
- admin
- signals (ugh - I keep running into misuse of them)

## Struggling to Find an Open Source Business Model

by Grant Paton-Simpson

### Abstract

After more than 45,000 downloads, the open source Python project SOFA Statistics has netted \$90 (not \$90k, just \$90). This presentation will explain strategies tried so far, and the options available to open source projects.

- SOFA project
- 65,000 downloads
- Lots of positive reviews
- <http://sofastatistics.com>

### Monetisation

- Surely must be possible to sell affordable to at least 0.5-2%
- Apparently a common rate for single-vendor commercial open source firms

**Warning:** When you play the role of market spoiler it's much easier to be famous than rich

**Warning:** 99% of the people who say, "The app store is a gold mine! I'm gonna be rich" are wrong

## Why an open source business model?

- Low starting costs
- Lower risk of assets than an inventory system

## Successful open source business models

- Android:
- Redhat: Enterprise guarantees as a business model
- Mozilla: He says it is Google (sugar daddy) but this is not true: Mozilla cleans up on advertising
- Eclipse: IDE support

## Models

- Vulture Model: Pick up projects abandoned/dropped/screwed by Oracle
- Bounty Model: Throw out money to get something done
- Honey Model: TODO - find this out

## Statements

“People who complain about open source companies closing off parts of their code generally haven’t tried to make money off of their open source projects. I suggest they go and try to do it.”

## Automated testing in Python and beyond

by Brenda Wallace

---

**Note:** missed the beginning :(

---

## Have unit tests

- Debugging is fast
- Find bugs elsewhere
- Blame
  - Feel free to point fingers at who did it
- look GOOD
- Nose!
  - Finds tests and runs them
  - Logging capture
  - STDOUT



- Running single tests

## Disney Stuff

WETA digital uses Disney open source code!

- Pythoscope
  - <http://pypi.python.org/pypi/pythoscope>
  - Reads your python
  - Writes a stub unit test for all your methods
  - *“You’ll have 100% code coverage and everything will fail!”*

## Other language test frameworks

Everyone forks and no one works together

### Java

- JUnit
- JTiger
- TestNG
- Looks like Python, but with more words

### Perl

- PerlUnit
- Over 60 other test frameworks!!!

### PHP

- PHPUnit
- SimpleTest is cool

### Erlang

- EUnit
- Best intro to testing ever.

## Write Tests

- Write tests before you write code
  - I switched from the Shell to writing Unit Tests as well!
  - This is how I write a huge majority of my tests.

## Re-run

- Hudson/Jenkins
  - Oracle Sucks
- Just do it
- I want this to have a service to do setup for you.

## Asynchronous and Evented programming in Python

*With Rocks In*

by Aurynn Shaw

- Works at WETA Digital
- <https://github.com/aurynn>
- She likes shiny things

---

**Note:** This talk not really covering Tornado

---

## Introduction

- Much ado about Node.js
- Event driven!
- Tornado, from facebook
- Lots of buzzwords
- Async is ‘*sort of*’ doing two things at once

## Threading

Really awesome until you hit issues like:

- Locking
- Shared state is hard
- Even experts have problems with it

## Multiprocessing

- Let the kernel care
- Easy to write MP code on unix-likes
- Can even go multi-system

### Problems

- Hard to share data
- import multiprocessing can be... quirky

## Asynchronous is hard

- **You have to let go**
- Bending your mind to the Async way is still hard
- A single mistake can hang
- Probably going to be slower

## So really, what is async?

- The core of async is the event loop.
  - Basically a while loop running
  - Checks for events
  - Events are pretty generic
  - In Twisted they have to be callbacks
- But once the code is in the loop, **you have to let go**.
  - Once the code is in the loop it can slow everything down
  - Your functions have to be as small as possible
  - Keep data/functions/etc really tiny
    - \* should do this anyway

## Async code is hard

- functions don't work anymore - you are working with **Deferred's**
- Most APIs built on twisted return Deferreds

```
from twisted.internet import defer

df = defer.Deferred()

def gotARequest(request, response):
    df = conn.query('select * from user;')
    df.addCallback(_someData, response)
    df.addErrback(_someError, response)
```

```
    return df

def _transform(rows):
    # rows here is now the returned data
    return [my_object(row) for row in rows]

def _someData(rows, response):
    response.render(rows)
    response.finish()
```

This is how you fire off things:

- `.callback` starts the callback chain
- `.errback` causes the callback chain to explode and die messily

### Synchronous Example of same thing

```
def my_funcZ(request, response):
    rows = conn.query('select * from user;')
    # This will hang until DB gets back to us
    return response.row(rows)
```

### Asynchronous != faster

- Not actually faster
- The event loop is overhead
- The event loop is overhead without proper coding

### So why do this?

Aurynn says:

- Scales beautifully
- Terribly elegant (not sure I agree with this - need to try it)
- closer mapping to reality

### What about Tornado?

- Event loop + web framework
- uses inline callbacks

Incomplete code example:

```
def my_func():
    def rows(results):
        for res in results:
```

## So why Tornado over Twisted

- Supposedly faster
  - Aren't performance tests biased?
  - Tornado's benchmarks are on extremely insignificant tasks. Literally "Hello, World".
  - Twisted has libraries to interact with real blockers like:
    - \* AMQP
    - \* IMAP
    - \* POP
    - \* PostGres
    - \* SOLR

## New Zealand Python User Group - The Whats, the Whys and the Hows

by Danny Adair

- Trying to get Python into NZ
- Started around 2005

**Quote:** No one is forced to use Python. That doesn't apply to other languages.

- Website: <http://nzpug.org/>
- Kiwi PyCon: <http://nz.pycon.org/>

## Tourist suggestions

- Pickton
- Malborough Sound
- Nelson
- Golden Bay
- Waitomo
  - <http://www.waitomo.com/waitomo-glowworm-caves.aspx>

## 1.15 Pycon AU 2011

### 1.15.1 Social

- Women's breakfast packed the place. Lots of great people showed up!
- Everyone at the conference was lovely.

## 1.15.2 Talks

### Keynotes

#### Audrey Roy

Audrey was opening keynote speaker. I'm partial but I think she rocked it. :)

#### Mary Gardiner

Mary Gardiner spoke about:

- Be an open source super hero and try to change the world
- Find a project that helps people and get involved!

Sample projects:

- plover <http://plover>.
- Software Carpentry <http://software-carpentry.com>
- calibre TODO get link
- Sugar TODO get link

#### Raymond Hettiger

See me fill in all the holes at: [http://pydanny-event-notes.readthedocs.org/en/latest/PyCodeConf2011/python\\_is\\_awesome.html](http://pydanny-event-notes.readthedocs.org/en/latest/PyCodeConf2011/python_is_awesome.html)

LA's own Raymond Hettiger talked about *What makes Python Awesome?*

- Other languages are awesome, how is Python better?
- Most Python releases are GPL-compatible. This makes it free.
- Going to a closed source language means you are trapped.
- Commercial distributions like ActiveState and Enthought
- Python has Zen. Internalized as core developers and internalized by community devs

### Community

- Mailing lists
- Newsgroups? HA HA HA
- Python User Groups

### PyPI

- Repo for Python programming language
- Over 16,000 packages
- *pip install ordereddict* works for Python 2.5!

## Killer apps

- Zope, Django, Pyramid
- Numpy and Scipy
- Bittorrent and Twisted
- YouTube
- Blender and Maya
- Win32 - Factoid: Me, @pydanny, has done all his windows programming using cpython and Win32!

## Easy to learn!

- Good teachers.
- Think how fast you got the types and control structures in Python. General 3 hours
- In a day you can learn special methods and stdlib
- Critical because if you need good Python developers it doesn't take long to get up to speed. Converting developers takes:
  - C takes 2 years to get competent
  - Java takes 6 months to get competent
  - Python takes a week to get competent
- Rapid development cycle
  - Scripting languages are unbeatable for development speed
  - Programs are grown organically
  - Interactive testing lets people work with their code results immediately.
  - Bang out real code fast

## Economy of expression

- Not many words or characters to get things done.
- clear English means non-coders can understand your work
- **Pydanny factoid:** One of the first times I wrote Python on a whiteboard for a boss at NASA/SAIC they thought it was very legible pseudo code representing a complex process.

```
import hashlib
import os
import pprint
hashmap = {}
for path, dirs, files in os.walk('.'):
    for filename in files:
        fullname = os.path.join(path, filename)
        with open(fullname) as f:
            d = f.read()
            h = hashlib.md5(d).hexdigest()
            filelist = hashmap.setdefault(h, [])
```

```
filelist.append(fullname)
pprint.pprint(hashmap)
```

## Beauty Counts

- Readability is the #1 mentioned characteristics of why organizations choose Python
- The beautiful appearance on the page directly affects a programmer's sense of joy
- Makes us go home and write code
- If you can read other people's code that makes it easier to maintain
- Because we all *mostly* share the same idiom it means we can read each other's code. That doesn't stifle creativity - it just means we can get along.
  - As a parent I can say I would have *loved* having a formal uniform at school. As a geek in school I would have loved that too. :P

## Interactive Prompt (REPL)

- Python experts don't memorize Python
- They use the interactive prompt often (I try to write tests...)
- This is a killer features that runs circles around compiled languages
  - Python shell
  - IPython
  - BPython (My favorite)

## Behind the Scenes

Philosophy of core dev

- Conservative growth
- *We read Knuth so you don't have to*
- Aim for simple implementation

## Protocols

To interact with these we have defined protocols

- DBAPI
- Hashlib
- Compression
- WSGI for the web
- Conversion protocols



## Specifics of Python: The Foundation

- Dictionaries and Lists
- Automatic memory management
- Overridable syntax
- Exceptions
- **You can reprogram the brackets?**
- **And we can reprogram the dot?!?**

## Winner Language Feature: Iterator Protocol

- High level glue that holds the language together
- Iterables: strings, lists, sets, dicts, collections, files, open urls, csv readers, itertools
- Um... I know this. I've had to construct these on my own in other languages. But not Python... Wow - I just realized this just now.
- List comprehensions give us joy
- List generators are amazing. No one else has them

## Winner Language Feature: Generators

- Serious magic
- A million rows in a generators is nothing
- Simple syntax to do them. You only need the YIELD keyword.

## Winning language Decorators

- Expressive
- Easy on the eyes
- Works for functions, methods, and classes
- **Factoid:** I have problem writing them. Serious problems. :(

## Winning Language Features: `exec`, `eval`, `type`

- Not a fan of `exec` and `eval` because when used in my experience they are done badly
- But `type` is awesome

### Winning Language Feature: With Statement

- Clean, elegant resource management: threads, locks. etc
- Important tool for factoring code
- Contains the setUp and tearDown code.
- The reverse of functions

### Winning Language Feature: Abstract Base Classes

- TODO - go over this one

### Winning Language Feature: Indentation

- Makes the code really clear
- We write our pseudo code this way
- Less errors!
- Less ambiguity!

### Meta-matters: Using decorators for better Python programming

by Dr. Graeme Cross

---

**Note:** I have a lot of trouble writing decorators and I'm here to correct it!

---

Question: How do we write decorators so that Sphinx/Docutils can handle decorated stuff?

### The preamble

- An intro to writing reading/writing decorators
- Python 2.6/2.7 only
- Assumes you have a basic understanding of Python, writing functions and writing classes
- Slides and content at <https://bitbucket.org/gjcross/talks>

**Warning:** Code examples are designed for clarity and not for production code!

### What is a decorator?

- A function or class that modifies or extends another function or method
- Nothing fancy and nothing new
- aspect oriented programming

- Just syntactical sugar

```
@cache
def factorize(n):
    factors = []
    # calculate factors of n
    # takes lots of time for large n
    return factors
```

## Why use decorators?

- Robust design
  - separation of concerns
    - \* example: you can work the function and not the caching
  - Can easily turn behavior on/off
- Improved readability
  - Less baggage in code (Cache code is outside the function)
  - less lines of boilerplate
  - less code duplication
  - simplifies code maintenance
- Widely used in Python libraries & frameworks

## Why wouldn't use Python?

- Not built into Python 2.3 or earlier
- Can slow your code down (nested functions can sink your performance)
- Can hamper debugging when a decorator is written badly
- Documentation doesn't seem to work so well

## Decorator Syntax

- stackable
- prefixed with '@'
- can have arguments
- same for functions, methods and classes

```
@assert_inputs
@log_event
@validate_item
def itemable(x, y, z):
    a = x * y * z
    return a
```

## Typical uses

### preconditions and post-conditions

- Assert types
- check returned values
- authentication
- authorization

### Awesome ideas I need to use ASAP!

- debugging
- logging
- locking of resources (threading, io, database)
  - maybe deprecated by *with* statement?
- threads
- hardware

### Classic decorators

Properties! (A favorite of mine!)

```
class Love(object):  
  
    @property  
    def fiancée(self):  
        return 'Audrey Roy'
```

### Let's write a decorator!

- See PEP 318
- Because functions are objects you can pass them around with state and all that...

```
# remember functions are just objects, right?  
import math  
  
def trig_power(trig_func):  
    print "Storing function=", trig_func  
  
    def power(deg, n):  
        return math.pow(trig_func(deg), n)  
    return power  
  
if __name__ == "__main__":  
    sine_power = trig_power(math.sin)  
    tan_power = trig_power(math.tan)
```

Another example:

```
def report_entry(func):
    print 'Just entered a %s function' % func
    return func

@report_entry
def add2(n):
    ''' I add two '''
    return n + 2

if __name__ == "__main__":
    print add2(5)
    help(add2)
```

The problem with docstrings and wrappers:

```
def report_entry(func):
    print 'Just entered a %s function' % func

    def wrapper(*args):
        ''' our internal wrapper thingee '''
        print 'This will be our docs issue'
        return func(*args)

    return wrapper

@report_entry
def add2(n):
    ''' I add two '''
    return n + 2

if __name__ == "__main__":
    print add2(5)
    help(add2)
```

## A better version of our decorator

from functools import wraps

```
def report_entry(func):
    print 'Just entered a %s function' % func

    @wraps(func)
    def wrapper(*args):
        ''' our internal wrapper thingee '''
        print 'This will be our docs issues'
        return func(*args)
    return wrapper

@report_entry
def add2(n):
    ''' I add two '''
    return n + 2

if __name__ == "__main__":
```

```
print add2(5)
help(add2)
```

## Best practices for using decorators

- Document them well
- If you stack them, notate where stacking them can be a problem,
- Use the *functools.wraps* decorator for internal functions

## Some real-world uses

- @precondition
- @postcondition
- @assert\_range
- @assert\_type
- @stress\_data - maybe used in tests to fire off ‘random’ craziness?

## Class decorators

- Added in Python 2.6.+ and Python 3
- Singletons
- Class checks
  - must have unittests
  - must have docstrings!!!

## Some decorator thoughts by myself about Django and caching

- Why don’t we have a beaker style decorator pattern for Django projects?
- Is this just a behavioral thing?
- Can we write something that caches:
  - Key taken from name of function/method/class + args
  - Value from return object
- Again, what are we missing?
- Ask my good friend and caching master Jacob Burch why we don’t do this. . .
- Is this what Johnny Cache et al is about?

### Some advice

- beware the spaghetti!
- No hidden surprises
  - Do one thing and do it well
  - A clear name
  - No side effects
- Don't overuse decorators - TODO ask what is a good rule of thumb
- Not a one-for-one match for the classic decorator pattern!

### Further reading

- PEP 318
- PEP 3129
- Learning Python 38

### State of CPython and Python Ecosystem

By Senthil Kumaran

Overview: High level talk about CPython and other implementation

---

**Note:** Missed chunks of this so not all of it got in my notes

---

### Python Launcher for Windows (PEP 397)

- proposal by Mark Hammond and implemented by Vinay Sajip
- System to associate python files with a particular Python version
- Will be distributed with Windows installers
- Question: What is the status?
- Question: Which version of windows?
- Question: How can we accelerate development?

### Jython

- Seamless integration
- Uses a lot of Java models instead of CPython's stdlib.

## Teaching Python to the young and impressionable

by Georgina Wilcox and Katie Bell

---

**Note:** Missed most of this talk. :(

---

National Computer Science School: <http://ncss.edu.au>

### Challenges

- Laptops issued to students are so locked down the students can't install programming languages
  - They use silverlight
  - The also use a JavaScript version of Python
- *Sometimes the level of knowledge from some of the students can be both intimidating and discouraging.*

### Teaching CS in Australia

- Not enough people are teaching programming in high school
- We're doing a good job of supporting students who already know how to program
- The students who start out with nothing, have a (very) hard time catching up.

### How Python Evolves

By Nick Coghlan

---

**Note:** He uses a cool slideshow library

---

- Things have/are/will change

```
from __future__ import unicode_literals
```

### The Python Ecosystem

#### Gigantic community

- Product developers
  - Linux
  - Google
  - NASA
  - Blender
  - etc
- Tools developers



- Mercurial
  - Roundup
  - gdb
  - Reitveld
  - Bazaar
  - etc
- Distribution efforts
  - SciPy
  - ActiveState
  - Python(x, y)
  - etc
- Frameworks
  - NumPy
  - Web Frameworks
  - GUI frameworks
  - Everything on PyPI
- Documentation support
  - Online courses
  - books
  - web sites
  - <http://readthedocs.org!>
- Advocacy & Insight
  - Planet Python
  - Python User Groups
  - Pyladies!

## Alternative Implementations

Taking Python where CPython won't go

- PyPy
- IronPython
- Jython
- Stackless

## What kind of changes get accepted

Any time a change happens there is a cost. So...

- ‘do no harm’
- Want the ‘obvious way’ and the ‘right way’ to be identical
- Follow the **Zen of Python**
- Keep the whole ecosystem in mind

## Role of Core Developers

- Decide when and where to commit changes
- different standards for bug fixes and feature requests
- direct commits
- opine on direction

## Why contribute to CPython?

- educational
- fun problem solving
- terrifying and exhilarating
- CPython sprint on Monday and Tuesday!

## Panel: Python in the webs

- Moderated by Malcolm Treddennick
- Dr Russ Keith-Magee
- Dylan Jay
- Richard Jones

## Micro vs Macro frameworks

- Micro frameworks are easy
- Normal frameworks take a long time
- Macro frameworks are CMS like Plone and django-cms.

## Django thoughts

Malcolm said: *Django is a framework for perfectionists with deadlines, not beginners with deadlines.*

- Django gives you one obvious way to do things rather than a bunch of choices.

## Zookeeper

by Brianna Laughher

**Home grown conference management software**

working example: <http://linux.conf.au>

## Current System

Not a trivial syste,

- Started in 2007
- Python
- Handles call for papers
- invoicing
- Scheduling
- GPL
- social networks
- special offers
- schedule
- photo competition
- badge printing
- volunteer system
- inventory system

## Architecture

- Pylons
- SQL Alchemy usually PostGres
- Mako

## Roles

- organizer
- core team
- papers chair
- paper reviewer
- funding reviewer
- miniconf organizer
- (user + paid => 'attendee')
- (user + proposal accepted => 'speaker')

### Standout feature: Admin reports

- Amazing report system. **Can't capture them all!**
  - Status of users through the registration process
  - Funding reports
  - Volunteer things
  - A/V release checked report
  - XML metadata for video listings
  - Registrations per state/country
  - tons more

### Attention needed

- Write tests
- code refactoring
- un-LCA-fication (take the Linux conf out of the system,)
- Proposal selection
- scheduling UI
- dashboards
- mail merge
- mobile interfaces
- feedback system

### Call to action

- <https://github.com/zookeepr/zookeepr>
- <https://github.com/zookeepr/zookeepr/issues>
- <irc://irc.freednode.org/#zookeepr>

### Web Micro Framework Battle

by Richard Jones

- Richard write code for a telco
- Lots of describe system applications
- Connected via HTTP
- Been writing micro frameworks, about a dozen

Disclaimer: Talking about what is best for Richard Jones

### What he needs for standard library

- Easy to understand
- docs (especially downloadable)
- minimal magic
- no surprises
- terse
- HTTP request/response
- URL routing (“restful”)
- WSSGI
- PyPy and Python 3
- size of the framework (not too many lines of code)

### What is out

- No ORM or any DB wrapper
- Template engine
- Mega frameworks
  - Django, grok, pyramid, web2py, zope
  - routes + webob
  - et al

### Sample app: wiki

- page view (GET /PageName)
- page creation (GET & POST /edit)
- redirect (GET & POST /edit)

### Contenders

Here we go!

#### cgi + wsgiref

- Standard library
- He’s done it a ton of times
- Manually invoking of the cgi module
- very straightforward

## bobo

- Started as one of the first object parsing libraries.
- Became Zope “Bobo became tainted on the way”
- Form objects are pulled into the request

```
@bobo.query('/')
def index():
    return bobo.redirect("/FrontPage")

@bobo.subroute('/:name?', scan=True) # scan is required to be there for an arcane_
↪reason
class Page(object):
    def __init__(self, request, name=None):
        if not storage.wikiname_re:
            # more code here not included

if __name__ == '__main__':
    import boboserver
    boboserver.server(['f', __file__])
```

### Cons

- Docs were not clear about display of index
- Strange behavior like weird page not found issues
- Docs focus too much on the Bobo way and not how to make it works

## cherry.py

- docs in sphinx!
- easy to get into WSGI

```
@cherry.py.popargs('name')
class Wiki(object):
    exposed = True

    def GET(self, name=None):
        return wiki.render_edit_form(name)

    def POST(self, name, content='', submit=None, cancel=None):
        if submit:
            # not finished here
```

The funky bit:

```
conf = {
    '/': {

    }
}
```

### Cons

- Richard Jones had to guess to make things work
- Missing/funky bits

## web.py

My first web framework in Python!

```
class edit:
    def GET(self, name):
        return wiki.render_edit_form(name)

    def POST(self, name):
        f = web.input('content') # This is how you get content from form post data!

urls = (
    '/', 'index',
)
app = web.application(urls, globals())

if __name__ == '__main__':
    app.run()
```

### Cons

- Weird way of handling form post data
- urls are not a list of tuples

## bottle

- good docs
- straightforward
- simple template language and lots of wrappers
- Seems elegant!
- pretty cool!

```
@get('/')
def index():
    redirect('/FrontPage')

@post('/:name/edit')
def edit(name):
    if request.POST.get('submit')

run(host='127.0.0.1', port=8080)
```

## itty

- Very similar to bottle
- more explicit

- By Daniel Lindsley methinks
- much smaller docs
- Much smaller than bottle

#### Cons

- Redirects throw an error in the stacktrace

#### flask

- Relies on werkzeug on jinja2
- Downloadable docs
- utilities for testing
- shell/repl
- awesome debugger *app.run(debug=True)*
- Methinks it rocks. My own personal favorite microframework. :)

#### cons

- redirects require a download from werkzeug, not flask?!?

#### pesto

- Kind of like Flask but more explicit
- utilities for testing
- More verbose

#### werkzeug

- Sample docs are kind of laid out in a funny way
- Sample project does more than Flask sample app - how weird is that?
- Very clear code
- Url mapping done at the end
- No top-level application. So you have wire it together urls to views yourself. Maybe just use Flask? :D

#### aspen.io

- Very different than everything else
- So neat/odd that he had to include it
- Requires weird templates that make you stick in page breaks:

```
form aspen import Response
^L
raise Response(301, headers={'Location': '/FrontPage'})
^L
```



Structure of an app:

```
.aspen
index.html
%name/index.html
%name/edit
```

### Cons

- Odd boilerplate
- Not sure how to escape certain things
- Very thin documentation

### How do they rank?

**Warning:** this is for sites without sessions or user tracking! His criteria is not for large websites!!

Framework	Total
<b>bottle</b>	<b>7</b>
pesto	6
itty	4
cgi + wsgiref	3
flask	3
werkzeug	2
cherrypy	1
web.py	1
aspen.io	-5
bobo	-7

## Pyramid

by Dylan Jay

### Heritage of Pyramid

- Zope => Zope2
- Zope2 => Zope3 begat BlueBream
- BlueBream => ZTK
- ZTK => Repoze.BFG (inspired partially by Django)
- Repoze.BFG => Pyramid
- RoR possibly begat:
  - Django
  - Pylons
  - TurboGears

## Who uses it?

- Karl Project
- juice.to

## Differences between Pyramid and others

Here we go:

Pyramid	Django	Plone
Not opinionated	Opinionated	Structured CMS
Very simple	Simple	Complex
No DB	ORM	Custom DB

## Sample view

```
from paste.httpserver import serve
from pyramid.configuration import Configuration
from pyramid.response import Response

def hello_world(context, request):
    return Response('Hello world')
```

## Routes

```
config.add_route('myroute', 'me/{one}')
config.add_view(hello_world, route_name='myroute')
```

## View Callables

```
class MyView(object):
    """ So much more obvious than Django CBVs! """

    def __init__(self, request):
        # something here I didn't write down

    def __call__(self):
        # why doesn't Django do this?
        raise HttpFound('stuff')
```

## Renderers

```
from pyramid.view import view_config

@view_config(renderer="json")
def hello_worldrequest():
    return dict(content="hello")
```

- One of the renderers is a **jinja2** package.

## Templates

- Default: Chameleon (latest version ZPT system)
- Can use Jinja2 trivially

## Traversal

- I don't really care but the Zope fans seem to love it.
- Works in that you can get things within things
- Some claim you can't do this in Django - but they are wrong. See MPTT.

## View Predicates

All described here: [http://docs.pylonsproject.org/projects/pyramid\\_zcml/en/latest/zcml/view.html](http://docs.pylonsproject.org/projects/pyramid_zcml/en/latest/zcml/view.html)

## Security

- Separation of authentication and authorization.
- You can easily control the security context of a request
- Seems to rely natively on ACL, but you can replace that if you want

```
# TODO - add example
config.add_view(myview, name='my-view.html', )
```

See the fun of:

```
class Blog(object):
    pass

blog = Blog()

blog.__acl__ = [
    (Allow, Everyone, 'view'),
    (Allow, Editors, 'edit'),
    (Allow, Editors, 'delete'),

]
```

## Scaffolding

- Provides a default best practices project layout. **Why doesn't Django do this?!?**
- Very obvious static directory

## How Django does it better

- Django’s consistent persistence system means easier to cook up reusable apps

## Zen of Python

by Richard Jones

aka 19 Pythonic Theses

## Guido’s Original Design Philosophy

### Timesaving concepts

- Borrow ideas from elsewhere whenever it makes sense.
- “Things should be as simple as possible, but no simpler.” (Einstein)
- Do one thing well (The “UNIX philosophy”).
- Don’t fret too much about performance—plan to optimize later when needed.
- Don’t fight the environment and go with the flow.
- Don’t try for perfection because “good enough” is often just that.
- (Hence) it’s okay to cut corners sometimes, especially if you can do it right later.

### Clarity concepts

- The Python implementation should not be tied to a particular platform. It’s okay if some functionality is not always available, but the core should work everywhere.
- Don’t bother users with details that the machine can handle (I didn’t always follow this rule and some of the of the disastrous consequences are described in later sections).
- Support and encourage platform-independent user code, but don’t cut off access to platform capabilities or properties (This is in sharp contrast to Java.)
- A large complex system should have multiple levels of extensibility. This maximizes the opportunities for users, sophisticated or not, to help themselves.
- Errors should not be fatal. That is, user code should be able to recover from error conditions as long as the virtual machine is still functional.
- At the same time, errors should not pass silently (These last two items naturally led to the decision to use exceptions throughout the implementation.)
- A bug in the user’s Python code should not be allowed to lead to undefined behavior of the Python interpreter; a core dump is never the user’s fault.

## Beautiful is better than ugly

See [http://en.wikipedia.org/wiki/Euclidean\\_algorithm](http://en.wikipedia.org/wiki/Euclidean_algorithm):

```
function gcd(a, b)
    while b != 0
        t := b
        b := a mod b
        a := t
    return
```

Wikipedia's version is not as pretty as Python:

```
def gcd(a,b):
    while b != 0:
        a, b = b, a % b
    return a
```

### Explicit is better than implicit

File openings are not that explicit

The `:` in Python is lovely and explicit.

```
class Circle(object):

    def __init__(self, radius):
        self.radius = radius

    def area(self):
        """ The 'tau' value is from outside the class """
        return tau * self.radius
```

### Simple is better than complex

- Something simple is easily knowable
- Something complex is not
- Automatic memory management means code is simpler
- That we can define getter/setters and override existing ones in Python is awesome.

Getting length of objects is simple in python:

```
l = [1,2,3,4,5,6,7,8]
len(l)
```

Try to keep your try/except blocks as small as possible. You'll thank yourself later.

### Complex is better than complicated

---

**Note:** I never actually think about this koan.

---

```
for line in open('document.txt'):
    print(len(line), line, end=' ')

# how about opening up things
for file in glob.glob('*.txt.gz'):
    for line in gzip.
```

## Flat is better than nested

### Inheritance flattening

- Keep object inheritance shallow
- Multiple inheritance keeps things shallow but things get more complex
  - Richard Jones worries about this
  - I don't worry that much. Never bites me the way Java did.

### Break up complex structure

- Keep your *if/elif/else* use as light as possible
- Smaller code == Better code

### Sparse is better than Dense

- Is this a style guide thing?
  - whitespace?
  - naming standards
- I (pydanny) think it is about spartan programming
  - <http://www.codinghorror.com/blog/2008/07/spartan-programming.html>
  - [http://ssdl-wiki.cs.technion.ac.il/wiki/index.php/Spartan\\_programming](http://ssdl-wiki.cs.technion.ac.il/wiki/index.php/Spartan_programming)
- Koans by Tim Ansell
  - 14 arguments for a method is too much
  - Don't compromise on complexity by adding more complexity

### Readability counts.

- Koan: Readability is the number 1 reason why organizations select Python

```
if (x == y);
{
    // logic
};

// a day wasted
```

## Special cases aren't special enough to break the rules

- Everything is an object

## Although practicality beats purity

Sometimes the rules need to be broken:

```
>>> class Two(int):
...     pass
...
>>> print(Two(1))
1
>>> Two.__str__ = lambda x: '2'
>>> print(Two(1))
2
```

A better example is circular imports.

- <http://stackoverflow.com/questions/3955790/python-circular-imports-once-again-aka-whats-wrong-with-this-design#3956038>

## Errors should never pass silently

- Errors should not be fatal
- Don't blame the user for bugs in Python
  - Either the core devs fault
  - Or the user added in ctypes

## Check out except Exception at the bottom!

*logging.exception(error) captures the entire error to the logs!*

```
try:
    handle_a_client()
except socket.error, e:
    log.warning('client went away: %s', e)
except Exception, e:
    logging.exception(e) # This captures the whole traceback!!!
```

## In the face of ambiguity, refuse the temptation to guess.

```
1 + '1'
# blows up in Python, not in other languages
# We like this behavior!
```

Also, remove the ambiguity and whack some parenthesis

**There should be one - and preferably only one- obvious way to do it**

Protocols:

- File API
- DB API
- WSGI
- etc

**Although that way may not be obvious at first unless you're Dutch**

- Guido can be quirky
- Community feedback keeps BDFL in check

*You try to shoot yourself in the foot, only to realize there's no need, since Guido thoughtfully shot you in the foot years ago.* - TODO: find who said that

**Now is better than never**

- Fix the problem now
- Try it in your shell, and your tests
- *Perfection is the enemy of the good*
- Python 3 was years in the making, but much less than Perl 6.

**Although never is often better than *right* now.**

Things that ain't gonna happen

- Adding '?' to identifiers in Python

**If the implementation is hard to explain, it's a bad idea.**

If you can't say it, don't do it

**If the implementation is easy to explain, it may be a good idea.**

Just because you can explain your idea, if it has no point then it shouldn't be included.

**Namespaces are one honking great idea – let's do more of those!**

- locals > nonlocals > globals > builtins
- Me (pydanny) loves this about Python



## Reference: Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

## 1.16 Django 1.3 Webinar

by Jacob Kaplan-Moss

### 1.16.1 New Features

#### Templatetags

##### with

New syntax:

```
{% with total= objects.count name=request.user.name... %}
...
{% endwith %}
```

##### include

New syntax:

```
{% include sub/template.html with total= objects.count name=request.user.name... %}
```

Where only specified arguments are included

```
{% include sub/template.html with total= objects.count only %}
```

## load

New syntax:

```
{% load my_tag from my_tag_lib %}
```

Also handy bits for future work for making url generation easier:

```
{% load url from future %}
{% url "path.to.view" %}
```

## Class Based Generic Views

<http://django.me/generic-views>

One of the problems with using class based views is that if you store data in it, then its not thread-safe. Which is why you have to do **MyClass.as\_view()** because it makes things thread-safe so always build from something that inherits from the base **View** class.

basic usage in urls.py for direct\_to\_template:

```
TemplateView.as_view(template_name='blah.html')
```

Detail views:

```
class AuthorDetailView(DetailView):
    queryset = Athor.objects.all()
```

## Sample JSON view

Here we go:

```
from django import http
from django.utils import simplejson as json

class JSONResponseMixin(object):

    def render_to_response(self, context):
        "Returns a JSON response containing 'context' as payload"
        return self.get_json_response(self.convert_context_to_json(context))

    def get_json_response(self, content, **httpresponse_kwargs):
        "Construct an `HttpResponse` object."
        return http.HttpResponse(content, content_type='application/json',
                                   **httpresponse_kwargs)

    def convert_context_to_json(self, context):
        "Convert the context dictionary into a JSON object"
        return json.dumps(context)

class JSONDetailView(JSONResponseMixin, BaseDetailView):
    pass
```

## Should you use CBVs?

- With caution
- Use it **ONLY** when you need it.
- **Wait** until some best practices before embracing it wholesale.

## Model “on delete” options

[http://django.me/on\\_delete](http://django.me/on_delete)

The basics:

```
author = models.ForeignKey(Person, on_delete=models.PROTECT)
```

On your models if you want to protect the associated model:

```
author = models.ForeignKey(Person, on_delete=models.DO_NOTHING)
```

Only on database fields that are defined to accept null:

```
author = models.ForeignKey(Person, on_delete=models.SET_NULL)
```

For setting a new value via a handy sentinel object:

```
def get_dummy_person():
    p, created = Person.objects.get_or_create(name='DELETED')
    return p

author = models.ForeignKey(Person, on_delete=models.SET(get_dummy_person))
```

## Testing

Resources:

- <http://django.me/testing>
- <http://docs.python.org/library/unittest.html>

Lots of enhancements to make testing more powerful and more fun. Django 1.3’s test framework is built on **Unittest2**.

- Vastly improved failure messages
- Easier to skip tests:

```
class SomeTests(unittest.TestCase)
    @unittest.skip("is always skipped")
    @unittest.skipIf(*conditional*)
    @unittest.skipUnless(*conditional*)
    @unittest.skipIfDBFeature(*conditional*)
    @unittest.skipUnlessDBFeature(*conditional*)
    def test_my_stuff(self):
```

- TestCase.addCleanup
- assertRaises context manager
- Class and module-level setup/teardown

- Backwards compatible
- `assertNumQueries`:
  - assert number of queries in a views
  - Good for confirming possible performance issues
  - <http://django.me/assertNumQueries>
- How to make it work:

```
from django.core import unittest
class MyTest(unittest.TestCase)
```

## RequestFactory

<http://django.me/RequestFactory>

So you can test without going through urls or middleware:

```
def crazy_test(self):

    rf = django.test.client.RequestFactory()
    request = rf.get('/url')
    response = some_view(request)
    self.assertEqual(response.status_code, 200)
```

## Caching backend

- Now supports multiple caches (`settings.CACHES`)
- Old syntax works but you'll need to upgrade at some point
- Features
  - Versioning
  - site-wide prefixing
  - transformations
  - pylibmc support - new and faster memcached library

Jacob suggests:

- switch to pylibmc
- can be tricky to install on some operating systems

## Static Files

- `django.contrib.staticfiles` - collects static files from multiple apps into a central location.
- [django.me/static-files](http://django.me/static-files)
- **media** Files uploaded by users, probably stored in a `FileField` or `ImageField`
- **static** Static assets that are part of your site - CSS, JavaScript,
- **files** Either of the above

In production you have a couple extra steps:

```
* Use a dedicated media server (nginx preferred)
* run `STATIC_ROOT = '/var/www/static'`
* run `STATIC_URL = 'http://media.example.com/'`
* run `./manage.py collectstatic`
```

Notes:

```
switch existing sites only if you are unhappy
```

## Everything else

- Built in support for Logging (<http://django.me/logging>)
- TemplateResponse (<http://django.me/TemplateReponse>)
  - Good for writing decorators that adds in stuff after a view template has been run.
  - Good for not writing so much page specific middleware
- `django.shortcuts.render()` (<http://django.me/render>)
- Transactions as context managers (kinda neat)
- “pretty” error emails (???)
- context-aware simple tags ([http://django.me/simple\\_tag](http://django.me/simple_tag))

## 1.16.2 Upgrading Django

Jacob’s process

1. Use virtualenv + pip
2. Upgrade just Django. Don’t upgrade other things at the same time.
3. Test and pray you have real unit tests.
4. Upgrade 3rd Party Python Apps
5. Deploy
6. Start adding new features

## Upgrade and Test

The process:

```
pip install -- upgrade Django
./manage.py test
```

This is why having unit tests is a **good** thing. But also click around and test things manually.

## Things to watch for

- Accessing internals: `models._meta`, `queryset.query` and other internals are not guaranteed by Django to not change.
  - So if your code depends on these things they can break between versions
  - Mark this code in comments as using internals
  - write tests against this code!
- Arbitrary keyword arguments: `model.save()`, `model.delete()`, signal handlers, etc can change.
  - Use `**kwargs` instead of keyword arguments!
- Monkeypatches
  - Just say no
- Deprecated APIs and backwards incompatible changes (coming up next)

## What about 3rd party apps?

- The big ones should all work: South, Celery, Haystack, Tastypie, etc
- Maybe make a sub-app or fork for those that do not.

## Deploy!

Get requirements:

```
localhost$ pip freeze > requirements.txt
remote$ pip install -U -r requirements.txt
```

Safety path:

1. Build new virtualenv
2. Grab code
3. Build reqs there
4. Point apache at new version of site.

## 1.16.3 Deprecated Features

check out <http://django.me/deprecation>

## Admin

Don't use `AdminSite.root()`!

Follow this pattern:

```
urlpatterns = patterns('',
    (r'^admin/', include(admin.site.urls))
)
```

## Custom auth backends

New rules for anonymous users that must be in place:

```
class MyAuthBackend(object):
    supports_object_permissions = False
    supports_anonymous_user = False
    supports_inactive_user = False
```

### 1.16.4 Backward Incompatible Changes

Usually because of security or database bugs. Sometimes the docs were wrong.

#### Security Fixes

- Added CSRF validation for AJAX requests - Done to protect FLASH, not browsers.
  - Symptom: Posts request will fail .
- Placed restrictions on filters in the admin
- Stopped rendering passwords in PasswordInput
- Users that are inactive can't reset their passwords anymore

#### AJAX specifics

<http://django.me/csrf-ajax>:

```
$.ajaxSetup({
    beforeSend: function(xhr, settings) {
        if (!(/^http:.*/.test(settings.url) || /^.test(settings.url))) {
            // Only send the token to relative URLs i.e. locally.
            xhr.setRequestHeader("X-CSRFToken",
                $("csrfmiddlewaretoken").val());
        }
    }
});
```

#### Data-loss bug

File field deletion issue (Look up in Jacob's slides!)

#### Optimizations

- manually managed transactions (via `@transaction.commit_manually`) needs to be explicitly closed
- New index on session table:

```
python manage.py sqlindexes sessions

    * But Jacob recommends using memcached or redis sessions for performance on
    ↳ sites with huge numbers of frequent users.
    * Google on django-redis-session
```

## The rest

- Clearable FileField widget is the default
- No more PROFANITIES\_LIST (re-set to get the old behavior)
- Localflavor corrections for Canada, Indonesia, and the USA
- FormSets can no longer take empty data
- Initial SQL no longer works in tests. Use fixtures instead.

## 1.16.5 Predictions for future Django versions

### Predictions for 1.4

#### To be removed

- Python 2.4 support
- Remaining single-db support
- Old style messages are going away
  - Don't use `django.contrib.auth.messages!`
  - Use new messages (`django.contrib.messages`)
- Legacy auth backends are going away

See <http://django.me/depreciation>

### Rampant Speculation

- Basic schema migration support
- Refactor and formalization of “apps”
- Improved ideas of what a “user” is
- Better hooks for monitoring, debugging, and metrics
- template compilation
- Better time zone support
- Python 3

### Predictions for 1.5

#### Possible removals

- Python 2.5 support (2.6 is just plain faster)
- `mod_python` support (no more support, no way to download, just say no!!!)
- Removal of function based generic views
- Old-style url and ssi template tags



## 1.17 Pycon 2011

### 1.17.1 Biased Survey of the Python Web

By Mark Ramm

**The state of Python web things is that right now it is the best of times and the worst of times**

#### **What is good right now**

- Deployment - WSGI
- Frameworks - more pythonic
- Community - bigger (django)
- Lots more great libraries

#### **What is bad right now**

- The choice of frameworks is too big
  - Just in the web templating space the options to choose from is too big
- Confusing
- Leaderless
- Complicated
- Frustrating

#### **Django specifically**

- huge\_community
- new\_python\_devs
- template\_language\_controversy
- model\_layer
- isolated\_community

#### **From mike\_bayer import awesome**

- SQLAlchemy
- mako
- beaker

## The state of the python web

- Confusing
- Leaderless
- Complicated
- Frustrating
- Fertile
- Innovative
- Amazing

### 1.17.2 Data Structures in Python

by Alex Gaynor

Python is awesome because it implements basic data structures as described by Knuth.

Paraphrase: *The Python core has read Knuth so we don't have to!*

Alex Commandments

1. Use types idiomatically
2. Sometimes you don't get a choice
3. Be efficient, when it doesn't cost you anything
4. sometimes you have more than one concern to deal with. The standard lib can help!
5. Don't do more than you have to: collections.abc are there to help.

## Builtins

- list vs tuple
- list vs set
- set vs frozenset

### list vs tuple

“I only use tuples if using a namedtuple would be equally appropriate”

Not a performance or mutability issue, but use them idiomatically

### sets vs lists

- lists have an order, sets don't
- sets must be hashable
- sets let you check for uniqueness super-fast

## sets vs frozenset

blah blah

## The stdlib

### collections.OrderedDict

- new in 2.7
- For when you have a dict that needs an order
- Syntax:

```
OrderedDict ([
    ("name":Field),
    ("type":Field),
    ("state": Field),
])
```

### collections.deque

- Fact: list.pop(0) and list.insert(0) are slow
- Good for in memory logs and such

## Other

- Array: Good for a bunch of types of the same sort
- heapq: Look into it.

## Do It Yourself

When you have to do it yourself use collections.abc!

- abstract base classes for extending collections
- Because you **don't subclass dict ever!!!!**
  - Subclassing Python's builtin containers tends not to behave as we want
  - Subclassing the ABCs does
- OrderedSet example: <http://code.activestate.com/recipes/576694/>

## 1.17.3 Fun with Python's new features

by Raymond Hettiger

**question:** For Collections.counter, I thought you weren't supposed to subclass dicts directly

I think this is all Python 3.x material

## Collections.Counter

Can do anything you can do with a dict but you can use counter notation:

```
c = Counter()
c['x'] += 1
c['y'] += 1
del c[x]
```

Convenience methods:

- `most_common(n)` returns a sorted list of the `n` highest counts
- `elements()` lists all of the contents individually. Differs from `__init__` which returns pairs: (elem, count)

Counter Math:

```
>>> # simple
>>> x = Counter(a=1, b=1)
>>> x.subtract(a=1, c=1)
>>> x
Counter({'b':1, 'a':0, 'c':-1})
>>> #advanced
>>> {'a','b'} - {'a', 'c'}
{'b'}
>>> Counter(a=1,b=1) - Counter(a=1, c=1)
Counter({'b': 1})
```

## collections.namedtuple()

Works like a regular tuple but lets you assign names to each field:

- makes the code self-documenting
- Makes the printed `__repr__` intelligible
- Lets you change tuple order without affecting client code

One of the single best changes you can make to existing code. The additional space cost for namedtuples is zero

Convenience methods:

- `_asdict()` turns a named tuple into a dictionary
- `_replace()` creates a new named tuple with altered values:

```
>>> result._replace(failed=2)
TestResult(failed=2, attempted=7)
```

Pro tip: Using the Field Structure:

```
>>> LabeledResult = namedtuple(
    LabeledResult,
    TestResult._fields + ('blah'))
```

Pro tip: Every time you make one reset the slots to nothing:

```
>>>> class Point(namedtuple):
...     __slots__ = ()
```

## caching

LRU Cache:

```

from functools import lru_cache

@lru_cache(maxsize=100)
def big_computation(*args):
    ...

```

### 1.17.4 Greasing the wheels of exploration

---

**Note:** Pycon politics kept me out of most of this talk. Which sucked!

---

by Michael Sims

“The last time we explored a new landmass was Australia over 3000 years ago”.

**Question:** How do you keep the camera lenses on the rover clean of dust

### 1.17.5 Obfuscated Python

by Johnny Healy

Really funny!

#### Assignment Operators

Fun things you can do:

```

>>> object = str
>>> tuple = lambda *x: x
>>> __builtins__.object = str

```

#### Comparisons

What?!?:

```

>>> object() == object()
False
>>> object() == object()
False
>>> object() == object()
False
>>> help((lambda x:x*x).func_code)
... blah blah blah
>>> code = type((lambda x:x*x).func_code)
>>> code
<type 'code'>

```

## 1.17.6 Opening the Flask

by Armin Ronacher

### April Fool's Joke

Decided to mock the various microframeworks:

- web.py
- bottle
- web2py

### Features

- Eirik Lahavre
- Entirely made up
- “Impressive scaling capabilities”
- one file framework

### What he learned

- people took him seriously
- marketing beats quality
- features don't matter
- people aren't looking at source code
- Does not have to be new

### Inspiration

- be honest
- don't reinvent things
- stay in touch with others

### Details

- jinja2
- wuerkzueg
- optionally blinker for signaling
- tons of documentation
- “best of breed” code

## Some numbers

- 800 LOC
- 1500 LOC tests
- 200 A4 Pages of Documentation

## Ecosystem

- Over 30 extensions
- very active mailinglist
- over 700 followers and 100 forks on github - yay

## Design Decisions

- use of Context Locals as globals
  - Rather than pass around the request object you can see it everywhere
- No import time side effects
- Explicit application setup
  - Applying WSGI middlewares
  - more than one app
  - testing
  - create app in function
- circular imports
- cached imports
- keeps things simple

## Extensions

- Formal extension system
- Approval process
- Core must stay small so extensions must work with core

## Lessons Learned

- Documentation matters
  - Use a clean documentation style and people will help document more
  - Documentation style for extensions
  - Simple visual design
- Communication matters

- Heartbeat signals
- Consistency

### 1.17.7 Outreach

by Asheesh Loroia

#### How the Fedora design group does it

- Provide contributor bounties
- Tries to get in a new contributor every two weeks
  - offers help for new people
  - tries to get people of all levels
  - if you get three contributors you get a free t-shirt

#### How the SF rails group grew

- Was at 2% woman
- Ran women only or woman + 1 events
- Huge turnouts!

#### Dos and Don'ts

- Do: Decrease latency (and terror)
  - respond to mailing lists!
  - Set a deadline for responses
- Do: Set goals
- Do: Tell people what to do
  - assign bugs
  - Point them at existing issues
- Do: Attach people to strong communities
  - Don't just run single events
  - Keep things moving forward forever
- Do: One-on-one interviews
- Don't: Talk about how much you hate doing X

#### Conclusions

- Demographics reinforce themselves
- Exploit the biases in your outreach



## Next steps

- All it takes is effort
- Run a meetup outreach effort
- Run a starling bounty for your project
- Put your project on OpenHash
- Ask for help: <http://asheesh.org/pycon11>

## 1.17.8 Pluggable Django Patterns

**Warning:** Corey is a great guy and really smart. I've learned from him on other topics. But I strongly disagree with what he presented in this talk. My notes are extremely incomplete.

by Corey Oordt

- An app should not be a monolithic pile of code
- A pluggable app is focused
  - It does what it needs to do and nothing more
  - Should have all its dependencies easily described
  - Should be adaptable
  - Should be easily installed

### How do you make an app sure is pluggable?

- Use small pieces
- types of apps
  - data (manages apps)
  - utility (single functions like pagination)
  - decorator (add functionality to other apps django-mptt, django-taggit)

## 1.17.9 Re-Introduction to C

- By Noah Kantrowitz
- Good speaker, knows topic
- Should have gone over fundamentals in second half before having us code
- My proposed change
  - # Insist people type out working code example presented (muscle memory) # Brain dump (what was second half) # Another insist type out of working code example based on brain dump material # Go to attendee excercises where they have to figure things out themselves

## Talk Notes

### Stack vs Heap

- Stack is very specific memory buffers
- Heap is everything else

### Pointer Arithmetic

C knows how many bytes in each variable:

```
int arr[10];
arr = 1000
arr + 1 = 1004
char arr2[10];
arr2 = 1000
arr2+1 == 1001
```

### Strings

```
char*s = "abc";
*s == 'a';
char s[4] = "abc";
*(s+1)=='b';
```

### Structures


Structures are the closest in C to having OO style classes. Use **typedef** to ensure that you can more easily construct the structure.

- Named+typed offsets
- **Syntax:**

```
`typedef struct Foo {int x; char y;} Foo;`
```

- Inside the curly braces you can stick in variables to be called on instantiation

```
Foo f = {1,2};
f.x==1;
Foo f = {0};
```

\* Useful in that everything is set to Zero even **if** there is more than one variable.  Even works with chars! Yay

### Unions

Rare thing used in C, and then specifically for high performance C.

- Also named+typed offsets

- Overlapping (?)
- **syntax:**

```
typed union Foo {int x; char y[4];} Foo;
f.y[3] = 1;
f.x == 0x01000000;
```

## Enumerations

Enumerations are symbolic references to numbers. While numbers you should not do math on them. Nice syntax sugar.

- **syntax:**

```
typedef enum Foo {BAR, BAZ} Foo;
```

- BAR is equal to 0
- BAZ is equal to 1

```
Foo f = BAR;
f = 1;
BAR + 1;
BAZ == BAR + 1;
```

## Comments

Same as in JavaScript.

## Function Declarations

- *int foo(int x, char y);*
  - returns int
  - accepts int x and char y.
- *void foo(void);*
  - Don't return anything
  - Don't accept any arguments
- *void foo();*
  - Don't return anything
  - Accept any number of arguments

If C cannot find something it will report an Int error

## Main

This is why python has “\_\_name\_\_” == “\_\_main\_\_”!

```
int main(int argc, char **argv);
./prog foo bar
argc == 3
argv == {"/prog", "foo", "bar"}
```

## printf

How to do a print in C:

```
#include <stdio.h>
def printf(fmt, *args) return fmt%args
printf("%s %u\n", "foo", 42);
```

- Coming from a user do this to make sure that their percent signs (%) are not accidentally made part of the format strings:

– `printf("%s", s);`

## blocks

Blocks are curly braces and then statements. **Variable statements must happen at the top of a block.**

```
{ stmt; stmt; }
if (expr) stmt; else stmt;
if () {} else {};
if () {int x=0; foo(x);}
if (x==1) {y=1;} else if (x==2) {y=2};;
```

## while

Same as python

```
while (x==0) {y++;}
```

## do while

Same as while but runs it once first

## Switch

Basically a structured GOTO system that jumps to each case as in other languages. How I think it works if expr evaluates to a number (confirm later):

```
switch (expr)
{
    case 1: {
        y = 1;
        break;
    };
    case 2:
```

```

        y = 2;
        break;
    default:
        y = 3;
}

```

## Preprocessor

Transforms your code before it hits the compiler. Don't use '#' to start any lines except for directives!

- `#include`
  - Takes the entire contents of this file and pastes it in. Not quite import!
  - `#include "file.h"` looks in the local path
  - `#include <stdio.h>` looks in the system libraries
- `#define`
  - Values that the preprocessor replaces (simple macros)
  - `#define Y 1.0` now works in the rest of the file. Think of it as a global. Can't do C expressions but can define text based replacements.
  - Don't put semi-colons at the end of a `#define` macro.
- `#define` can take arguments!

```

#define Z(a,b) foo (A * 2, b, 0)
Z(1,2);
foo(x +1 * 2, 2, 0);
`#define Z(a,b) foo((a) * 2, (b), 0)

```

- `#if` include other preprocessor bits:

```

#if X
    #define Y 1.0
    #include "file.h"
#endif

```

- `#ifdef` is used in older code and is simply `#ifdefined(X)`.
- `#pragma once`
  - Include guard
  - Makes sure you include something only once since you might have multiple files including the same thing and that can be bad.
  - Don't do `#ifdef __FILE_H__`!

## Headers

Headers are files that end in '.h' and contain function declarations. This way the compiler knows what functions are going to be used:

```
#pragma once

void handle_request(int sockfd, const char *request);
```

Sometimes you see *typedef struct Foo Foo* and this is to just let the compiler know there will be a struct called Foo.

## Useful functions

- string.h
  - length: *size\_t strlen(const char \*s)*
  - compare: *strcmp(char \*s1, char \*s2)*
  - copy: *\*strncpy(char \*s1, char \*s2, size\_t n);*
  - *memcpy(void \*s1, void \*s2, size\_t n)*
- malloc (buffer management)
  - *#include <stdlib.h>*
  - *void \*malloc(size\_t)*
  - *void free(void \*ptr)*
  - *void \*calloc(size\_t count, size\_t size)*
- stdio.h (I/O handling - files writing and reading)

## Runtimes

Check out: <http://docs.python.org/c-api/>

## Convore

<https://convore.com/pycon-2011/reintro-to-c-tutorial/>

### 1.17.10 Why I went to each event

I have my reasons!

## Re-Introduction to C

I've wanted to learn C for a long time and a lot of other Pythonistas feel the same way. When Noah proposed this I jumped right behind it and push for its inclusion in the Pycon-PC process (even though I wasn't part of the tutorial selection team I made my opinion known).

## 1.18 Scale 9x

### 1.18.1 Big asterisk talk

- By some guy at Digium
- Digium is what started the tech

#### The community

9800 contributors to Asterisk

#### Asterisk Design Issues

- Couldn't predict how it was going to be used
- Wasn't planned to be used in the cloud
- Asterisk was not planned to be used in some of the directions it is being used
- issues
- performance
- scalability
- extendability
- fault tolerance
- too many implementations to get new design architecture in!

#### Asterisk SCF

- Asterisk Scalability Framework
- Asterisk SCF is a collection of parts rather than a monolithic whole
- Allows distribution across machines

#### Get started

- [asterisk.org](http://asterisk.org)

### 1.18.2 asterisk intro

SIP = almost universal VOIP protocol IAX2 (don't use it!) Codec used to transfer voice-to-data is ULAW

Channels = how many translations from voice-to-data you need to handle. Generally about 1 channel unless you have scaling issues.

Registration = how your box is registered to accept calls for a particular number

Asterisk can do:

- conference calls.

- caller ID
- special prompts
- voice mail
- asterisk-to-asterisk can be free because there is no middle parties
- telemarketer torture

## **AEL = Asterisk Extension language**

A C-like language used to configure the behavior of an asterisk installation

## **Warnings!**

- People can hack into your box and call Siberia.
- SIPVicious is a tool originally written to check for hacks against Asterisk but is used to hack.
- Can it be placed in the cloud or do we need a datacenter?
- The internet goes down
- 64k bandwidth each way
- Echos are troublesome (Open Source Echo Canceller)
- Huge amounts of servers to run this thing

## **Changes in install screen**

- include the sounds in the codec (last three items)

## **Thoughts**

- check asterisk hosting

## **Resources**

- O'Reilly book: Asterisk: The future of telephony

# **1.19 MongoDB LA 2011**

## **1.19.1 LA MongoDB meetup**

- In February of 2011

## **What OS for production?**

- Linux
- Ubuntu



## For production

- Always use 64 bit machines
  - Use the 10gen binaries, not the Ubuntu ones
1. Register 10gen signing key
  2. Set up 10gen package repo

TODO - look this up on the mongo wiki

## Python Language Driver

- python-longodb
- On launchpad
- Check on pypi

## Backups

- Loops of hoopla in community about need for backups.replication
- Single server durability big part of 1.8 release coming up

## Question

- If you introspect on a document in order to list its fields, how do you manually order the fields?

## Features, ease of use

- learnmongo.com
- 

## 1.19.2 Database Design

### Pure SQL Implementation

All tables are assumed to have an ID.

Pros:

- All data in one database means simpler architecture
- All staff familiar with SQL
- SQL is proven

Cons:

- Django Admin becomes unwieldy with complex inlines.
- Requires sophisticated Python logic to handle Survey display and controls.
- Saving survey state is rigid and will make changing the survey model much harder.

- Survey and survey state make saving and fetching historical surveys very rigid. Post-launch might need to make new tables after any change to ensure no historical data is lost.
- Adding sub-surveys adds in a huge amount of complexity

## **client**

Represents customer for a particular survey

- name
- point\_of\_contact

## **user**

- name
- email
- password

## **profile**

- address\_1
- address\_2
- city
- state
- zip
- dob
- gender
- occupation
- employer
- address\_verified
- dob\_verified
- employment\_verified

## **survey**

- title
- description
- start\_date
- end\_date
- max\_responses
- value\_per\_response

- verification\_level
- client\_id

### media

- id
- url
- **Optional:** survey\_id

### media\_survey

**Question:** Can a survey have multiple media elements that are shared across other surveys? If not, then we probably can remove this table.

- survey\_id
- media\_id

### question

- label
- order (within the survey)
- type (radio, select, checkbox, text, textarea)
- survey\_id

### response

- value (integer, boolean, or text)
- question\_id
- user\_id

### survey\_state

Saves the state of a survey for a user.

- completed
  - start\_date
  - end\_date
  - user\_paid
  - survey\_id
  - user\_id
-

## SQL + Mongo Implementation

All tables are assumed to have an ID.

Pros:

- User/Financial data separate from survey data. Faster results for both.
- Financial data in SQL, which was designed for that task.
- Doesn't replace critical components of Django (Session, Auth, User with unproven MongoDB components)
- Survey logic changes much easier to implement in Mongo, just new app for each alteration.
- Survey and Response documents store historical survey app so continue to work over time.
- Adding sub-surveys is of moderate complexity
- Simpler display and controls of Surveys
- BSON/JavaScript interface

Cons:

- Need to construct survey forms from scratch (will have to do that in any implementation)
- New technology to most of the staff
- MongoDB adds complexity to the architecture
- Local deployments harder

### client (SQL)

Represents customer for a particular survey

- name
- point\_of\_contact

### user (SQL)

- name
- email
- password

### profile (SQL)

- address\_1
- address\_2
- city
- state
- zip
- dob
- gender

- occupation
- employer
- address\_verified
- dob\_verified
- employment\_verified

### survey (SQL)

- title
- description
- start\_date
- end\_date
- max\_responses
- value\_per\_response
- verification\_level
- client\_id

### media (SQL)

- id
- url
- **Optional:** survey\_id

### media\_survey (SQL)

**Question:** Can a survey have multiple media elements that are shared across other surveys? If not, then we probably can remove this table.

- survey\_id
- media\_id

### survey\_state (SQL)

Saves the state of a survey for a user.

- completed
- start\_date
- end\_date
- user\_paid
- survey\_id
- user\_id

### survey\_document (MongoDB)

- survey\_id (used to relate to SQL)
- survey\_app (we create new Django app for new versions of surveys)
- questions (list/array)
  - label
  - type (radio, select, checkbox, text, textarea)

### response\_document (MongoDB)

- survey\_document (copy of the survey\_document this is responding to)
- responses (list/array)
  - value (integer, boolean, or text)
  - survey\_document.question
- user\_id (used to relate to SQL)
- survey\_id (used to relate to SQL)
- survey\_state\_id (used to relate to SQL)

## 1.20 Django Master Class 2009

by Jacob Kaplan-Moss in Springfield, Virginia

---

**Note:** I had to deal with a work emergency so didn't get to experience the second half of the class.

- TODO - link to PDF slides in repo
- 

### 1.20.1 Caching

The difference between a site that scales well versus what doesn't comes down to caching.

- tomoayko/writings/things-caches-do

#### Cache backends to use

- memcached is the way to go
- tokyocabinet is an alternative to memcached. But Jacob seems more familiar memcached
- for testing you can explore
  - filesystem
  - database
  - local memory

## Cache setup

settings.py:

```
CACHE_BACKEND = 'MEMCACHED://10.0.0.100:11211/'
CACHE_BACKEND = 'file:///tmp/cache'
```

## Per Site Cache

- Django has a built-in cache.
  - This is good for read-heavy sites.
  - Not good for write-heavy sites
  - All or none solution

Code:

```
MIDDLEWARE_CLASSES = (
    'django...UpdateCacheMiddleware',
    ...
    'django...FetchFromCacheMiddleware',
)

CACHE_MIDDLEWARE_SECONDS = 600
CACHE_MIDDLEWARE_KEY_PREFIX = 'mysite'
CACHE_MIDDLEWARE_ANONYMOUS_ONLY = True
```

Details:

- Only GET requests are cached
- URL captures are added to the cache key
- The cache middleware and per-view share the same logic
- Cache keys are opaque, so cache invalidation is difficult

## Template fragment caching

sample:

```
{% load cache %}
{% cache 600 author_info author.id %}
    {{ author.render_something_expensive }}
{% endcache %}
```

## Low-level cache API

various Cache methods:

```
* set(key, value, timeout)
* get(key, default)
* add(key, value, timeout)
* get_many([key1, key2, ...])
* delete(key)
```

```
* incr(key, amt=1)
* decr(key, amt=1)
```

## Documentation

- <http://djangobook.com/en/2.0/chapter15/>
- <http://jacobian/r/django-cache>

## Cache decorators

Ack, focusing on work issues so not taking notes.

## Conditional view processes

Doesn't save performance but does save bandwidth

## 1.20.2 REST

### Gem

- `request.raw_post_data`

REST has an absence of bureaucracy

## Concepts

- Resources
- Names (URIs)
- Representations
- Connections (links)

## REST properties

- Addressable
  - Every resource is addressable
- Statelessness
  - Can make requests in any order
  - Can make doing transactions a bit challenging (need to treat state as a resource)
- Connectedness
- A uniform resource interface



## IOW: “Respect the web”

- RFC 2616

HTTP	API	SQL
GET	select	select
POST	create	insert
PUT	update	update
DELETE	delete	delete

## Adding an API

- Odds are you already have one! Your website!
- But cool kids like me (Danny) use JSON
- use Python 2.6+ version of json over simplejson to get better performance. Same library, just implemented in C.
- Jacob likes to set mime type during DEBUG to text/javascript to make debugging easier

## Plain Django

- Jacob likes to construct RESTful resources so he can encapsulate all the RESTy bits in external functions and classes. Keeps his API methods really short.
- In POSTS he remembers to set the right status

## Piston

### 1.20.3 Schema Migration

Keeping your sanity when changing models

## Gems

- You don’t need to use Django tools to do it. Consider sqlalchemy-migrate
- Question: Will South get brought into core? Not until more thought has been given and products mature nicely.

## The problem with the Django ORM

- manage.py syncdb doesn’t alter tables
- manage.py reset just deletes tables
- SQL Shell doesn’t record any changes.
- SQL scripts are problematic
- Forgetting migrations!
- Track migrations
- Need to be able to go forwards and backwards
- Keep multiple developers in sync, just like good source control

## The main lesson

- You have to have a tool
- Once started, you always have to use the SQL tool

## Django Migrations Tools?

- Nothing in core
- options Jacob knows about
  - desebe
  - django-evolutions
  - dmigrations
  - migratory
  - south
  - yadssel
  - sqlalchemy-migrate looks good.

## Django-Evolutions

- About 2 years old
- Works with **syncdb**
- Applies (some) evolutions automatically. Kind of terrifying!
- Write evolutions in Python or SQL
- Requires some care in workflow especially in a team
- Lead by Django core contributor (Russ Keith-McGee)

## South

- 1 year old
- disables syncdb; migrations from the start
- supports auto-generation
- Python only
- works for data migrations
- good for teams
- At this time is the recommended system.
- South lets you control dependencies
  - <http://south.aeracode.org/wiki/Dependencies>

## 1.20.4 Testing

### Gem

- fixtures = ['authtestdata']

### Quotations

Kent Beck:

```
Tests are the programmer's stone, transmuting fear into boredom.
```

Microsoft Research:

```
40-90% reduction in bugs takes 15-35% increase in time if you use Test Driven_
↪Development.
```

Jacob Kaplan-Moss:

```
Whatever happens, don't let your test suite break thinging, "I'll go back and fix_
↪this later"
```

### Django Tools

- Unit tests (unittest)
- Doctests (doctest)
- Fixtures
- Test client
- Email capture

### Unit Tests

- Python 2.7 should support it much better

### django.test.TestCase

- Extends Python unittest
- fixtures
- Email capture
- Database management
- Slower than unittest.TestCase

## doctests

- Neat
- Eat to read
- Hard to maintain

## Functional Tests

- a.k.a Behavior driven development
- “Blackbox”, holistic testing
- Hardcore TDD folks look down on functional tests
- But they keep your boss happy
- Easy to find problems; harder to find the actual bug.

## Fixtures

Jacob like YAML over JSON because of readability. Don’t bother with XML fixtures.

```
fixtures = ['authtestdata']
```

This is some basic Auth data from core that you can use to have groups and users. Hooray!

Test running:

```
./manage.py test
./manage.py test app
./manage.py test app.SomeTestCase
./manage.py test app.SomeTestCase.test_method
```

## New fixture tool

- <http://farmdev.com/projects/fixture>
- Content Types in fixtures

## Coverage tool

- Ned Batchelder’s Coverage
- <http://nedbatchelder.com/code/coverage/>
- Coming in 1.2!

## Mocking tool

- <http://www.voidspace.org.uk/python/mock/>

## Browser Testing

- Selenium (ya ya)
- Windmill (has Django support now!)

## Exotic Testing

- Static source analysis (pylint is my favorite game)
- Smoke testing via web crawlers (webcheck is my favorite)
- Monkey testing aka fuzz testing
  - <http://agiletesting.blogspot.com/2006/11/python-fuzz-testing-tools.html>
- load testing:
  - <http://www.portswigger.net/suite/>

## Further resources

- <http://bit.ly/py-testing-tools>
- <http://us.pycon.org>

## 1.21 DjangoCon 2010

### 1.21.1 Keynote: Why Django sucks and how we can fix it

By Eric Florenzano

#### Apps

Making custom changes to an app and now staying in trunk fails. Apps that provide models are inherently inflexible. primary key assumptions

#### Class based views to the rescue?

- No consensus on how to implement it?
- Where do you put customized view subclasses?
- urls.py is already overloaded

#### Generic Foreign Keys

- good for flexibility
- Bad for configuration

## Performance

- Things are slowing down
- Memory usage is going up in each version since 1.0
- performance is going down since 1.0

## Django Core

- Closed and very private
- Why isn't Alex Gaynor not a core developer?
- Why isn't there a truncatechars filter?

## Badteries

- DataBrowse is a joke
- lorem ipsum doesn't belong
- Need to move to a DVCS

## django.contrib.auth

- django.contrib.auth is inflexible
- first\_name, last\_name is culturally limited
- Admin is couple to user
- Integer primary key
- get\_profile is inelegant
- no way to use secure key

## 1.21.2 Reusable apps

By Alex Gaynor

Note: Alex talked FAST but it was one of the best talks at DjangoCon.

Basics

- similar patterns
- write once, use everywhere
- no need to reinvent this wheel

## But Standard apps patterns are not truly reusable

Too many differences in business logic between systems

- ex comments:
- comments apps have to cover many different business cases

- differing storage (comments in a group, against a logic)

### What about class based views?

- Better than function based views because you have inheritance
- `django.contrib.admin` has good examples of class based views

### Reusable frameworks

- an example would be badges
- brabeion
- You can't predict all the ways people want to use your software
- You know the common ways (80/20)
- Provide default behavior and let people swap it out
- `django-taggit` is his example
  - non-integer primary keys
  - per-user tags
  - official tags
  - custom caching managers, or anything else

### Custom Backends

Many different backend systems

- `django-registration`
- `django.core.cache`
- `django.contrib.auth`
- `django.core.files`
- `django.contrib.sessions`

### Libraries approach

- not everything is business logic
- Good API design
- Well defined problem spaces

### 1.21.3 Class based views

by Ben Firshman

- Views have not been updated since release of Django
- Generic views are very inflexible
- `newforms-admin` use class based views

## Ben's implementation

- In the request – views – response stream a view is just a callable (`__call()`)
- Class based views let you do mixins such as rendering Jinja or JSON values. This beats Piston, TastyPie, etc by letting you not have to create a separate application framework.

## 1.21.4 Customizing Django Admin

Peter Baumgartner and Michael Trythall

### User Experience

#### Problems with the Admin as a User Experience tool

- UI is the gateway to application log
- Users remember bad experiences, associate them with **you** and the admin
- Good experiences - Happy Customers - Profit!
- No dashboard, statistics, or recent activity
- no actions or modules highlighted or given priority
- No assistance/help for beginner users
- Impact from changes is not always clear
- Disconnect from external systems.
- Doesn't fit mental models
- Apps are not organized by context
- little or no navigation outside of breadcrumbs
- Doesn't follow workflow

#### Missing features of the admin tool

- Content and asset management tools, e.g. WYSIWYG, image manipulation
- Error recovery (undo)
- Export/Import with certain types
- Inline help systems
- Cross model search

#### Poor display for complex models

- Way too many fields on complex models
- Try to limit your fields



## Customizing the Experience

### ModelAdminMedia

- js
  - JQuery
  - AJAX
  - Fancy inlines
  - Inject HTML
- CSS
  - colors
  - layout

### Custom templates

(base.html, change\_form.html, etc)

### Model admin / Model Form hacking

- list\_editable
- row level permissions

### ModelForms

Use a model form and tell the Admin class to use ModelForms. Not well documented - UNNACCEPTABLE!

## 1.21.5 Lightning talks

### Hashing by david gouldin

Neat ways to do AJAX and fast view pages

## 1.21.6 Maintaining an old Django project

By Shawn Rider

### Background

PBS TeacherLine dates back to 2006. Catalogs, CMS, brochures, and classes lets users added tons of content every year. Hundreds of classes and tens of thousands of users.

Before 2006 TeacherLine was written in ColdFusion. Run on Windows, separated from the PBS architecture. PBS has a long history of development with open source. So a complete rebuild was necessary. Some technologies considered:

- Ruby / Rails

- PHP / Some PHP framework
- Python / Django

### What PBS likes about Django

- Speed of development
- Code Quality
- Modularity of framework
- Django Admin
- Active community
- Python!

### Things that worked for PBS

- Django is opinionated in a generally good way
- A culture of self-criticism
- Isolate functionality into reusable components

### Lessons learned

- Never override the User model
- Make tests right away
- Make the most of your VCS
- use tests
- take your time

### Sell the upgrade to the Uppers

- it will lower the cost of future development
- it will alleviate a pain point felt by staff processes

### What he wants

- Multi-configuration support out-of-the-box
- A better way to know when Django's modules are completely loaded into memory
- More robust handling (Signals++)

## 1.21.7 Maps of imaginary lands

By Malcom Tredinnick

## GeoDjango Admin

Nice experience out of the box.

- How does it work?
- What can I do next?
- Is this all there is?

## What is GeoDjango?

An interface for Django to provide mapping bits easy to work with for Django.

## OpenLayers

- Client side JavaScript framework
- Combines data from multiple data fields
- Provides neat looking UI around it
- Day to learn, lifetime to customize

### 1.21.8 pony pwning

By Adam Baldwin

He breaks stuff. He hack and cracks. And gave a really useful talk!

## Notes

**django** = pile of *awesome*

*Django has good security*

## Developers aren't perfect

Don't rely on QA to find your bug

## Security Failures minority issues

30% based on incompetence or ignorance 9% based are needle in haystack code issues (XSS) 1% are 0 days

## 90% of the problems

## XSS issues

- Template system issues
  - autoescape off, safe, mark\_safe
  - context HTML

- IE has security holes.

Avoid getting burned

- Consider OWASP ESAPI
- Audit templates
- Audit reusable and snippets
- Educate designers

## File uploads

- images can contain PHO
- ImageField does not case
- Image field does not check extensions
- File uploads often are put in unprotected directories

## Direct Access is bad

Return a Not Found error if people can't find something and log the exception

## Doing stupid things

Privileged operations with HTTP GET

eg /object/delete/2 is bad!

Also, don't expose IDs

## Click Jacking

/admin/ **was** vulnerable. This has been fixed as of Django 1.3

Mitigations:

- Set X-FRAME-OPTIONS DENY header
- Use django-xframeoptions middleware
- Implement frame breakout code

## admin

[Redacted]

## Communicate with security guys

- They are impatient
- Will publish it if you do not respond/fix

### 1.21.9 State of Pinax

Brian Rosner

#### Inspiration - Ole Kirk Christianson

Legos

- blocks
- sets
- themes

#### Lego sets - Starter projects

Pinax is **NOT** just for social networks

- zero project = directory layout, static files, deployment
- static project = zero project + static serve
- account project = zero project + account + mailer + uniform
- company project = zero project + blog
- basic project = account project + profiles + notification + announcements
- private beta project = account project + restricted access + waiting list + signup codes
- intranet project = basic project + restricted access + top level content apps (wiki, attachments, tasks, etc)
- code project = basic project + projects + group-based content apps (wiki, attachments, tasks, etc)
- friends project = basic project + friends + invitations
- social project = friends project, blogs, microblogging, user-to-user messages + LOTS

#### The Trade off

- How much to fix
- how much to make configurable

#### Pinax IS Django

- Django doesn't hide python and Pinax doesn't hide Django.
- Pinax is even more opinionated about Django

#### Questions

- tested on zc.buildout?
- Pinax Project site refactor?
- IE5 CSS corrections
- Pointing new users to Python, Django

### 1.21.10 Treehugging

by Brian Luft

#### Introduction

Where do we find structured data:

- Hierarchical data
- graphs
- organizations

#### Trees and relational database

- A table is not a tree. We need to do a few tricks

#### Adjacency List

Self referential foreign key. This is how usually I've done it in the past. Can be very cumbersome, requires a lot of joins, not obvious or easy. My own note: I believe Oracle has a tool to deal with Adjacency Lists but its database specific

- fast writes but slow reads
- fragile update operations
- You need to know what level in the tree your item is at when you look it up
- Easy to create orphans by accident
- Easy to start, hard to maintain

#### Nested sets

Not easy to set up and hard to modify. Query works regardless of depth.

- Efficient reads
- high maintenance cost

#### Materialized Paths

Every node in the tree has a "path" attribute. I did this once.

- Queries simple and fast
- Effectively de normalized
- Writes slow
- Requires some wackiness when making adjustments

## Django Apps

- django-mptt (we used this on <http://science.nasa.gov/>)
- django-treebead

Comparison chart

- <http://www.qompr.com/charts/63;django-hierarchical-tree-data/>

Versus:

- Treebeard supports all three models, MPTT only handles Nest Sets
- Both provide Move Node forms, MPTT provides a TreeNodeChoiceField(ModelChoiceField)
- MPTT is being maintained, treebead has slightly more active development
- Front end: Treebead has a get\_annotated\_list function, MPTT has some handy template tags / filters

Overall Impression:

- Treebeard: Overall impression: treebead model creation methods on model is weird and uncomfortable
- MPTT: Related items methods

## Miscellaneous

- django-treemenus - possibly solves some outstanding treebeard issues
- neo4j - Designed for large graph systems but Java based
- Suckerfish/Superfish - Some view related items
- protovis

### 1.21.11 Why Django Discussion

Open spaces thing:

```
@jdunck - I want to see features of how Django has changed lives.

@andymckay - You don't need 400 case studies, you need 6 great ones. Writing case_
↪studies is really hard.

-- Idea: DjangoCompanies.com

@natea - Plone allowed sponsorship to get higher rankings in company listings.

@audreyr - Feature the community on whydjango.com since other advocacy sites do that.

@jdunk
```

## 1.22 Pycon 2010

---

**Note:** I met Audrey at this conference. So my notes are even more incomplete than usual. Ha ha ha.

---

### 1.22.1 0.9 Pinax Roadmap

#### Must

- Implement per object permission system
- Finish the django-friends split
- Notifications redo
- Clean up profiles
- Finish off the improved install branch
- Test runner (I've got this)
- Tagging (I've got this)

#### Should

- Alternative themes (we want 3 or 4 themes)
- Settings refactor
- Externalizing some of the apps
- Better i18n support
- Add more external authentication

### 1.22.2 Leafy Chat

by Alex Gaynor

*TODO: Get the slides later*

#### Intro

Using AJAX, comet, and lessons learned

#### LeafyChat

- Used in Django Dash
- Leah Culver, Chris Wainstroth
- Orbited to handle proxying
- Orbited for Comet
- Twisted to handle IRC/Orbited connection
- Used JSON as message packet format for **EVERYTHING**



## Conclusiuons

- Kinda works
- Messy
  - IRC in same process as comment so it doesn't scale
  - Any changes goes on the server, the client and the UI lib.

## DjangoDose

- Built in a week before than DjangoCon
- Built on twisted, orbited, Django, and StompMQ
- Brian Rosner, Eric Florenzano
- Instead of individual channels just one channel on a message queue.
- Works really well for larger scale
- Initial users had repeats of early data

## Hurricane

- An attempt to take lessons learned from LeafyChat and DjangoDose and build a framework
- Uses producers and consumers in a planned, managed method.
- But done on no sleep over 40 hours - had abstraction issues
- Tornado
- Multiprocesses

## Pycon 2010

- Do the same thing as Hurricane but better. Include a lot more data (flickr, github) and users (Pycon)
- Uses redis
- Orbited proxies to a twisted Daeon
- One backend process for each item (one for RSS, twitter, etc)
- Don't reinvent Orbited. Its really good at what it does

## Conclusions

- Use orbited
- Asynchronous programming is hard

### 1.22.3 Django in Depth

by James Bennet

*I was trying to fix some problems at work so my notes here are amazingly incomplete.*

## Model Inheritance

- Abstract/concrete
  - `ye aulde abstract = True`
  - Use cases: common field sets and/or methods and/or META declarations
- DB level
  - No special mechanism. Just subclass a model
  - You can't directly subclass
  - Always implemented as multi-table
  - Has OneToOne key to parent
  - Good for modeling the “is-a” relationship
  - I don't like it and neither does James Bennet
- Python level inheritance
  - `proxy = True`
  - Will use the parent's table
  - Must have one abstract parent
  - use cases: Adding methods to existing models, adding managers or changing Meta behavior

## Django Views

- `SystemExit` is not caught by Django, otherwise can be caught
- Exceptions raised by exception middleware or 404 pages
- Deliberately uses empty `Context` because nothing about the `Context` can be trusted.

## AdminSite

Here we go!

## Overriding templates

- 

### 1.22.4 Dude, where's my database?

by Eric Florenzano

## Relational

- Highly structured
- Strong type system
- Powerful query language

- Python talks to all of them
- In common use

## Problems

- Hard to scale
- 

## Key/Value

- Tracking HTTP sessions
- User references
- URL shorteners
- simple and fast
- Examples:
  - gdbm
  - Kyoto Cabinet
  - Berkely DB
  - MemcacheDB

## Problems

- No interesting queries
- 

## Data Structure

- Super fast
- Maps to certain problems very well.
- Modification of key/value
- Structured values
- Atomic operations
- Example:
  - Redis

## Problems

- Lack of alternative implementations
-

## Graph

- Store data as nodes and edges in a graph
- Fits logically to many problem spaces
- Programmatic queries
- Examples:
  - Neo4j
  - VertexDB

## Problems

- Scale ceilings
- Lack of alternative implementations

## Document-Oriented Database

- Unstructured
- Formatted (JSON, Python object)
- Programmable Query API
- Examples:
  - CouchDB
  - MongoDB
  - ZODB
- Use Cases:
  - Activity streams
  - User data
  - CMS

## Problems

- Scale ceiling
- Implementation-specific weaknesses

---

## Highly Distributed Databases

- Optimized for multi-node
- Add and remove nodes on the fly
- Hard to do ad-hoc queries
- Sacrifice consistency

- Examples:
- Cassandra
- Riak
- HBase
- Hypertable

## Problems

- Eventual consistency
- Can't do efficient ad-hoc queries

### 1.22.5 Eventlets

by ???

#### Problem

Threads and stuff aren't so good right?

#### Solution: Coroutines

Using Generators are an easy way to do co-routines in Python. But these are limited.

#### Eventlet

Green threads on top of greenlet

### 1.22.6 Form Panel

by all the framework builders!

#### Who is who?

- Johnathon Ellis - FormAlchemy
- Christoner McDonough - Formish
- JKM - Django Forms
- Chris Perkins -

#### FormAlchemy

Looks good to me. I could see using it instead of DjangoForms

## Sprox

Ugh. Obvious but code not pretty/elegant.

## Django

Django forms are declarative and awesome.

## Formish

Not tied to models in any way. I guess you could roll your own.

### 1.22.7 Neo4j

by Tobias Ivarsson

## Data Model

Relationships are what are normally called Edges

- Relations have types and a direction, can be traversed in either direction
- Both nodes and relationships have properties which are key value pairs

## Questions

- Can a node have a relationship with itself?
- Yes
- Can nodes have types or do you just rely on properties??
- Nope

## Support for Django out of the box

Very similiar format to Django models.

### 1.22.8 Underwater robots

University of Maryland

## Details

- Python
  - 15,000 SLOC
  - AI, GUI
- C++
  - 50,000 SLOC

30 member team

What does Tortuga do?

- Competes in competitions
- Does not leak
- Drives under water
- see and move around obstacles
- home on sounds

Construction

- 6 thrusters
- Pressure vessel with a Mac Mini, batteries, custom electronics
- Runs Gentoo Linux
- 4 hydrophones
- Grabber used as a claw

## Python

- Great flexibility and unit testing support
- compact code
- easy to learn - easy to get new members up to speed
- stdlib and 3rd party support helped a lot

## Unit Testing

- No 3rd party library to install and manage

## Woes

- C++ integration
- Boost.Python and Py++ are powerful, but complex
- Overhead for wrappers is large in terms of dependencies, disk space, and compile time
- Small bugs and compiler incompatibilities lead to fragile bindings

## The GIL

- Inflexible nature greatly constrains concurrent system design
- Forced the core of our software into C++
- C++ calling back into python is especially likely to run afoul of the GIL

## GUI & Simulator

- Done in wxPython & Python-Ogre

## Dependency Management

- Build things with a custom script

## Conclusion

- Dynamic languages are great fit for dynamics problems
- Python is great for robots because of their dynamic nature
- State.py is their AI library

### 1.22.9 Using Django in Non-Standard Ways

by Eric Florenzano

- Categories
- Choosing alternatives to what Django offers
- Using bits of Django in other contexts

#### The main thing

Its not as hard as you think its going to be.

#### Choosing alternatives

- Use Jinja2 with Django.
- More performant
- Some people like it more

Check out his slides for how it was done. In Django 1.2 you can just write a custom template loader.

#### What about my apps?

- Sometimes you just have to rewrite your template code.

#### Not using `django.contrib.auth`

- Sometimes it will be more difficult than not using it
- When using it will make your code less straightforward than not using it
- ie a facebook app

How to do it:

1. Create a tiny app with one model whose PK is the Facebook User ID



2. Wrote one decorator function redirect to an authorization page if not auth'd
3. Convert a few apps to use the tiny app's key instead of `django.contrib.auth.model.User`

## Not using the ORM?

If against a database, why not write a pluggable Database backend? Cause writing that is non-trivial.

*Too much detail to notate but check this out below. Its a way to use settings without a settings file!*

Gem:

```
from django.conf import settings
settings.configure(USE_I18N=False)
```

## Using the ORM in stand-alone mode

1. Make sure the app with models is on your python path
2. Call `settings.configure` with your DB info
3. Optionally copy `manage.py` to your project
4. import your models and use them

## WSGI Middleware

- Start looking at repoze
- `repoze.bitbt` - Scales images
- `repoze.squeeze` - Merges JS/CSS based off statistical analysis
- `repoze.profile` - Aggregates Python profiling data across all requests, and provides an HTML for viewing the data

## Other

- Yardbird - IRC using Django's URL mapping to match messages and views into handling the callbacks
- Djng - Django based microframeworks
- Jng - Single file CMS

## 1.22.10 Django Internationalization

by Matt Croyden

*I already know all this thanks to most of my other Django work but I may learn something new*

## Code

Sample code:

```
from django.utils.translation import ugettext as _  
  
output = _("Hello, world!")
```

## Translating models

- Don't forget the Meta attributes!
- All titles and help\_text items on model attributes should be translated

## Translating templates

sample:

```
{% load i18n %}  
{% block trans %}{% endblock trans %}  
{% trans "blah"%}
```

## Its just Python

- gettext, ugettext, ungettext are just simple wrappers around Python behavior
- Django provides a lazy translation layer on top, useful in models and template tags/filters

# 1.23 DjangoCon US 2009

## 1.23.1 Keynote: On politics

by Ian Bicking

Django community resists the politics of code.

## 1.23.2 Keynote by Ted Leung

- Sun Microsystems VP on new tools
- Cloud computing guy at Sun

## Status of web frameworks

- The status of web frameworks
- Django is **the** python web framework
- Growth of django jobs is astronomical, but our numbers are still low
- Django jobs increased 692% in 2009!

## RIAs

- Flex/Flash
- Silverlight
- JavaFX
- Open Web

Latency is a big deal. If your app does not respond in 0.1 seconds, then you have a problem.

## Cloud Computing

- Began as deploy / operational play
- Will impact development

## Competition

- Rails have solved the deployment issue
- Lift (Scala) has a lot of neat features like easy Comet support
- Nitrogen (Erlang) has a strong focus on UI.

## JavaScript

- Bleah, slow
- But getting optimized for speed

## What can Django do better

- What about an AJAX version of the admin module!
- Django + Comet is a pain in the butt!
- Deployment standards
- Monitoring and building that into the admin!!!
- Suggested stuff that Pinax is doing for best practices
- REST needs to go into core

### 1.23.3 Confessions of a Perl bigot

---

**Note:** In early 2010 Frank and Jacob Kaplan-Moss hired me to consult at RevSys.

---

## by Frank Wiles

Old Django dude who does a lot of Perl

- Don't flame him
- We want dynamically typed, garbage collected languages

## Afraid to give up his addiction to Perl

- Didn't want to give up his expertise
- Lazy and did not want to learn

## Outside observations

- urlconf uses regex
- Quality of his code supply is better
- Good documentation
- Lots of modules that don't install right
- Hands don't hurt as much because pinkies aren't used to grab the shift and obscure things
- Seeing lots of startups using Rails and Django

## Perl thoughts

- Perl is a very insular community. Not as many blogs and articles in the modern methods
- Django is moving upwards. Not yet stuck with legacy issues

## What is Django missing?

- Centralized repo a'la CPAN. *We do have pypi.*
- Perl has more HOWTO type docs and blog posts on various topics. *PyMOTW is a good example of what we need.*
- Dedicated Q&A website. Such as Perlmonks.org
- Centralized user groups structure
- Shared community hosting/infrastructure. *Don't we have webfaction?*

---

**Note:** Back in the day I said *webfaction* but compared to djangozoom.com, ep.io, dotCloud.com, gondor.io and now Heroku, it is nothing!

---

### Advice on having a Django addiction

- Just say yes.
- Best of breed.
- Development speed versus app performance
- Right level of coupling

### 1.23.4 Continuous Integration

- A way to make deployments and releases stronger

#### Assumptions and Constraints

- CI includes running tests
- VCS must be part of things
- Compiling

#### What is CI?

- As a developer saves to the VCS it pulls the code, runs the tests, and reports.
- Can be done nightly, daily, or per VCS commit.
- Build artifacts need to be handled. A build artifact is data and other non-code pieces.

#### Tests

- Integration
- Functional
- Code inspection

### 1.23.5 Deploying Django

#### by Brian Rosner

1. relative paths in settings.py
2. copy from Pinax style of setting things relatively
3. use PROJECT\_ROOT everywhere

### 1.23.6 GeoDjango

#### by Adam Fast

- Ability to CRUD geodata
- Querying based on geographic criteria

- Display of this data (slippy maps)

## GeoDjango

- installation is easier
- PostGreSQL: GEOS, PROJ.4
- SQLite: GEOS, PROJ.4, GDAL
- geodjango-basic-apps
- Lots of neat and easy to use methods thanks to the ORM and GeoTools

## Grue

- Geometries are big!
- PostGIS DBs aren't cross platform. Migration is HARD.
- "Search" isn't straightforward
- You can use South for migrations!

## 1.23.7 High-Performance JavaScript

---

**Note:** This was an awful talk. It could have been better, but the phrase, "Hey I travelled 1000 miles to get you guys to contribute to my unknown framework"" rankled badly with people like me who traveled 3000+ miles to attend.

- Plan for 200 millisecond latency
- 

## by Erich Ocean

- JavaScript can be used on mobile devices
- Much, much faster
- Flash, etc has limitations

## About sproutcore

- The only HTML 5 application framework adopted by and now developed at Apple
- Open source
- Developers write standard HTML, CSS, and JavaScript
- Is seen as successor to NeXTStep/Cocoa
- First cloud application framework

## The HTML 5 Stack

- Webkit (FF, Chrome)
- fast JS engine
- SproutCore application framework
- Favorite DOM library (jQuery, YUI, Dojo, Extjs)

## Django's opportunity

- Client-server interaction in the cloud is hard
- Django has a very well designed ORM that exposes meta-informantion
- Sproutcore has a well designed ORM that makes use of meta-information

### 1.23.8 Lightning Talks

#### Dive into cpython by Alex Gaynor

Um... yeah. Doesn't apply to me!

#### pywatch by Chris Heisel

Done with buildout. Not touching it.

#### Web Cube. by aron and Nickolai

- Sales guy will say anything
- Designer takes too long and has programming challenges
- Planning and design process takes too long
- CEO needs day-and-night effort to make things work
- Commercial

#### Surlex by Cody

- Regex expression can be gnarly
- Surlex tries to make things easier

### 1.23.9 No Bad Pony

by Dr. Russell Keith-McKee

## What he does

- wotnows.com.au - blogs
- wearehunter.com- music

## How is pony formed?

- Russell said pony about a silly requirement request
- Cal Henderson said he wanted a magic pony

## Bad pony is:

- ideas that are just plain wrong (not following pep 8, stupidity)
- impractical ideas (problems that don't exist, changes design contracts, architecture astronauting)
- Design mismatch (DTL to Jinja, ORM with SQLAlchemy, test framework with Nose)
  - Ignores the philosophy (Add GROUP BY HAVING to ORM, Add variables/callables to template language, Add AJAX to forms)
  - Just a setting (less is good)
  - Wrong direction (feature creep, make dev server multithreaded, add connection pooling)
  - Add a backend (more work, hard to test, they provide APIs anyway)
- The community
  - The core doesn't have to do everything
  - In fact, the core can't do anything
  - Community has an essential role
  - Blessing by core doesn't make code better
- Add X to contrib
  - django-comments
  - django-tagging
- Process ponies
  - write more blog posts
  - have a weekly news summary
  - have a nightly tarball download
  - Have a continuous build
  - Have a precompiled PDF documentation

## Massive Features

- Support for multiple database connections
- schema evolution
- Support for non-SQL data stores



### Some popular tickets

1. Better related objects admin UI
2. Multiple database support
3. ModelValidation
4. FOR UPDATE in querysets
5. Extendable user auth module
6. Custom app\_label/verbose\_name
7. Support clearing FileFields
8. Identity mapper
9. Binary DB fields
10. Session-based messages

### How do you get your Pony?

1. Don't be an ass.
2. Offer to help out
3. Help out!
4. Advocate
5. Document
6. Write some code
7. But follow the process

Earn some trust from the core team!

### If you must write the code...

1. Do the research
2. Demonstrate you understand the problem
3. Implementation trumps idle discussion
4. Maximize utility for the core team

### No Pony != End of the world

## 1.23.10 Pluggable Applications

by Nowell Strite and Shawn Rider

## Background

- Many sites
- Built on a diverse technical architecture
- Many repetitive components
- Ease of implementation
- Highly specific details

## Reusable App design pattern

- Easy to share functionality with other projects
- Quick to get features up and running
- Possible to expand and enhance functionality

## Reusable App design pattern - sort of

- Apps are expected to live at one URL
- Convention of adding 'extra\_content' params are not satisfactory
- Conventional template replacement is not always flexible enough

## Template tags to the rescue

- Kind of fixes the problem
- but form postings become complicated
- but performance issues

## The Use Case

1. Access to a robust database of curated resources
2. Community features to allow educators to share and discuss those resources

Because discussion is so important in this setting, many objects in Peer Connection support discussions. Given that the purpose of the site is partially to learn to use technology, a clean user experience is essential.

## Their reqs

- Sensible URLs that follow our patterns elsewhere on the site
- Allow discussions app to interact with other apps and objects in expected ways
- Allow posting of data and error handling in the same location - no redirects to stand-alone pages to fix a message post
- Flexibility to enforce permissions and other variations of display in an ad-hoc way

### **Solution: Pluggable Resusable App Pluggable**

- Apps can be presented at various URIs through the site
- Better architecture
- Content generated by the pluggable app is in the main content well, but can be farmed out to other places

### **Make it so**

- pluggable urls
- pluggable views
- Subclassing / instantiation of pluggable application

### **Very nice!**

- They do control of request stuff very nicely!

## **1.23.11 Scaling Django web apps**

---

**Note:** Best Django scaling talk I ever attended!

---

### **By Mike Malone**

- Common bottlenecks
- Django gave stuff for free
- Scaling is not speed or performance
- Not affected by performance
- No silver bullet

### **A Scalable Application**

- Writing to local disk will kill performance
- A scalable system doesn't need to change when the size of the problem changes

### **Caching**

- Per site cache (except for forms)
- per view cache
- heavily personalized sites don't work this way

### Django has a nice low-level Cache API

- from django.core.cache import cache
- Use signals.post\_save so you can invalidate caches
- They would do a hack by using cache.set to None for certain cases

Monkey patching the cache backend lets you tie the django.cache system to memcache

### Upping your appserver

- Load balancing

### Need to see the slides

- Tons of details dumped really fast
- Lots of good sys admin information

## 1.23.12 Ur doing it wrong

by James Bennett

Our WVA friend

### To start off: RTFM

1. People starting Django often don't have a background in Python.
2. Perl maxim #11924 "Well, if you don't know what it does, why did you put it in your program?"
3. Get the basic python bits down

### Idea: Django module of the week

1. Just like Doug Hellman's Python module of the week
2. Would take 2 years
3. The problem is that I need a better blog engine

### Django is just Python

1. Django applications are just python modules. Did not reinvent the wheel.
2. Django Views could be callable classes. Hmmmm...
3. Django admin used callable classes for views.
4. Yet people never grok that Django is really just python.

## The biggest problem with the perception of Django

People think that Django is too glued together. But that isn't true:

1. Just do *import SQLAlchemy*
2. You don't need to use Django's default components

## Testing new Django apps

What James does to check out your app:

1. Need to have your stuff working with standard Django install tools.
2. top level stuff needs to be there in your package.
3. Stay away from setuptools. Go to *distribute* by Tarek Ziade
4. Wants to see a concise one-sentence explanation from the README describing what the app does
5. You better have a good explanation as to why you are duplicating other person's work
6. Wants to see documentation or will file a bug. Will jump away from your work if you don't have it unless you are lucky.
7. Pick a license. Choosing one causes a flame. Django prefers the BSD.
8. Tests are handy. If there aren't unit tests he doesn't trust your code.

What turns off James:

1. Put this module in `django.contrib`
2. Ignoring standard features of Python or Django is silly

Something cool:

1. `django-lint` looks awesome
2. `pypants` is awesome
3. Assigns a score represented by pants on Python packages

### 1.23.13 DjangoCon Report

---

**Note:** This was the draft of something I was going to report back on to NASA HQ contract management to show why us developers and NASA staff wanted us to attend this sort of stuff. Unfortunately, Indyne/HITSS was staffed by amateur level management.

---

- Outreach to Ames Research Center
- Django Education Foundation
- Scaling Django
- Pinax Tutorial
- Pinax 0.7 release
- Discovering Django resources

## 1.24 Pycon 2009

TODO - Dive into the folders for django, excell, internet, and scrape.

### 1.24.1 Behind the scenes at Everyblock.com

- Adrian Holovoty

#### Model designs

- Looks straight-forward
- But can't do models per type
- **So perhaps a bit of abstraction?**
  - **Entry Attribute Value method?**
    - \* Needs multiple joins
  - **Bitmap method uses limited columns**
    - \* varchar01, bool01
    - \* Needs an in-database schema
    - \* Sorting can be an issue. Everyblock wrote a custom sort method
- GeoDjango lets you do geographic queries against your models
- Look up select\_template method in Django

#### Data Scraping

- Regex for addresses
- <http://code.google.com/p/templatemaker/> (built off C + python)
- Getting public info can be hard

#### IMPORTANT

- June 30th of 2009 Everyblock gets open sourced!

### 1.24.2 Building test frameworks with twill

Get the docs for this. I think this is eclipsed by Windmill

#### What is a test engineer?

- QA people write a lot of test code
- Doesn't do much manual testing

### Some options they don't use

- Expensive options like Rational Test Suite
- Windows-only

### Cases

- You have tons of tasks to tests and how do you handle mutable tasks?

### Test framework stack

- Nose
- Scenario objects
- Order Factory (Abstract State Machine)
- Page Objects
- twill.idyll.org (built off Mechanize)

### Pros

- Easy
- Fast
- Pythonic

### Cons

- No JS support
- too simple?

## 1.24.3 Concurrency

- Jesse Noller
- Networks are unreliable
- Bandwidth is never infinite
- Network is never secure
- Topology changes
- Transport is never free

## 1.24.4 Coverage Testing - good and bad

- WRITE SOME CODE! (happy)
- Does it work? (sad)

## Coverage Measurement

- Shows which lines of code are executing
- How much of your code is covered by your tests?
- Your tests test your product
- Coverage testing **tests your tests**

## Coverage tools

- trace.py in standard library
- figleaf by Dr. Titus Brown
- coverage

## Running coverage

cli:

```
$ coverage -e x test_mycode.py arg1 arg2
$ coverage -r -m
```

Can also annotate source files. Can run as a Nose plugin can run it programmatically:

```
import coverage
coverage.blah()
```

## Good side of things

- Statements are marked as executed or not
- window into your code.
- Fine tune by marking clauses to ignore via pragma
- Clauses can be ignored by regex
- Write more tests

## Bad: blunt tool

- Everything considered important
- Leads to many false alarms
- **Excluding code to boost coverage is too easy**
  - Tempting
  - You'll never come back
  - You are only hurting yourself



## Goals of coverage measurement

- **100% coverage**
  - ideal
  - Not always possible
  - real world issues are thorny. Hardware failures for example
- **Practical goals**
  - more coverage is better
  - Actual number doesn't matter
  - False quantifiability: bad! Only cover what matters!
  - Use coverage results to understand your code.

## 100% coverage issue

- You can fool yourself into thinking you are bug free.
- Tests can have bugs too!
- Doesn't deal with path coverage

## How coverage works

- `sys.stack_trace()`
- you can trick the trace function
- Lie to python about where the lines are
- Trace byte codes rather than statements

## 1.24.5 Django History

### Considerations

- Web development needs to be stupidly fast
- automate repetitive tasks
- **best practices**
  - Obsessed with web standards and best practices
  - You have to enhance, extend, and maintain

### Choices

- open source
- BSD license
- python
- **pragmatism over methodology**

- Didn't care about GoF, MVC, TDD?
- **Why break the standards?**
  - A lot of it is snake oil
  - “There is no silver bullet” - Fred Brooks
  - There is the right way and getting things done.
- **Ramifications on questioning anything new**
  - “What's the need?”
  - No architecture astronauts
  - You risk reinventing the wheel
  - You can take things to far (Ruby for Rails)
  - Messy internals
- **Web development ought to be fun**
  - Rails asks why software development can't be fun
  - APIs should match behavior, not show how they work
- **Good APIs almost documents itself**
  - Documentation driven development “DDD”
- **Why does Django ship full stack?**
  - Lack of choices at the time of inception
  - Might make separate choices again
  - There is a reason why glue code is called glue code
  - Full control over code means you get to define the API nicely
- **Don't repeat Zope's mistake**
  - Don't have people learn Django and not Python
  - Innovation happens elsewhere
  - Always someone smarter outside your own group
  - Small pieces advance quicker
- **Don't make opinionated software**
  - Ego is dangerous
  - It is not about you
  - You can't possibly predict what everyone wants
  - There is no “one obvious way”
- **Lots of choices is bad for new users**
  - There are always more beginners than experts for your projects
  - Make complicated things easy
- **Choices in the future**
  - Long tail or huge scale

- Learning curves
- Django in the Enterprise
- Does Django want to fight STRUTS
- **Avoiding inevitable backlash**
  - \* Inviting criticism and take it well
  - \* Make conscious decisions

### 1.24.6 Guido's Keynote Speech

- Python is too smart to fail.
- community

### 1.24.7 Things to look at

- Reddit C55 CSS compiler
- Don't lie about your user-agent when writing web crawlers

### 1.24.8 MetaClasses

- Everything is an object
- Classes are objects
- The class of a class is its metaclass

### 1.24.9 Lightning Talks last Day

#### GeoDjango

- Lets you add geography data to Django
- Works with PostGIS
- Attaches to normal django modelsgeodjango.rst
- Looks very easy to do!

#### What up with Zope

- Why is Zope still relevant
- PyPI is 90% Zope packages
- Age
- ZODB

### New command line parsing called argparse

- argparse
- look on [code.google.com](http://code.google.com)

### Zain's cheapo continuous integration tool

- <http://inzain.net/blog/>

### blis looks awesome

- For handling gigantic lists. Why not just use numpy?

### C types

- only for cpython
- Allows access to c constructs

### Using parenthesis to avoid backslashes in python

code:

```
>>> t = ("I am"  
... " a screwy geek"  
... " who likes python")  
"I am a screwy geek who likes python"
```

## 1.24.10 Pinax Talk

by James Tauber

- Pinax means blank
- **Seperation of presentation and content**
  - Persistence
  - UI
  - productivity through compromise

### Quisition

- Flashcard site

## Stuff Upcoming 0.7

- Better sign up
- New Group App
- Membership app
- endo vs exo
- Shared media
- Ships with famfamfam silk icons

## 1.24.11 Plone front end other front end

by Carlos A de la Guardia

### Why Plone?

- all the great features
- built on python

### Problems with Plone

- Complex
- Hard to extend
- Can be slow
- Plone does a lot of things
- Forced to use caching
- Caching is not always the answer to traffic
- Sometimes we are asked to add a couple of features that are Plone-y
- **Easy to end up with Frankenplone**
  - Becomes very hard to extend and maintain

### Content Mirror

- Developed by Kapil
- Serializes plone content into a relational database
- Supports 3rd party / custom archetypes content
- Simple to set up
- completely automated
- Works currently with 3.1
- Configure the database via ZCML
- Works with Oracle, PostgreSQL, and MySQL. I guess anything that SQLAlchemy

- **easily extended**
  - CouchDB anyone?
  - GAPE!

### Now we can

- Use plone to manage content and manage workflow
- Serve plone content fast

### How does it work?

- Integrates into the Plone event stream
- Uses SQLAlchemy to turn Plone schemas into tables
- Each Plone content type gets its own table
- Each custom content type gets its own table
- Once set up, content changes are sent synchronously to the database

### Several created front ends

- **Repoze.bfg**
  - Lots of Zope styles
  - Much lighterthan-Zope infrastructure
- Plango uses Django to generate Plone content

### 1.24.12 Sphinx mini-tutorial

- Written in python
- **Has two handy features**
  - Autodoc
  - Autosummary
- More than docstrings
- **Formats**
  - HTML
  - Latx
- Author in Restructured Text
- Works with easy\_install
- Has a quickstart tool
- Searchable
- Beautiful via styles

- Templates use Jinja, which is very similiar to Django templates
- Works with everything under pygments
- **Can do test discovery via nose**
  - Start a useage section with **bold**
  - Add in doc tests

### Steps

- Use sphinx-quickstart tools
- Configured in conf.py
- Autosummary pulls in all the bits of your module

## 1.24.13 5 essential strategies

Why?

- Quickest way to verify that all features ‘work’
- Freedom to experiment
- **Manual testing is error prone and time-consuming**
  - does not scale
  - people are lazy
  - Hard to test edge cases

### Useful tools

- <http://javascriptlint.com>
  - Validate against multiple browsers
- <http://www.mozilla.org/rhino>
  - no browser, command line
  - continuous integration
  - no DOM
- Check out python spidermonkey C bindings
- **Browser tests**
  - Selenium, Windmill
  - hard to automate
- QUnit for JQuery
- <http://saucelabs.com/>

## Strategies

1. Test Data Handlers
2. Test JavaScript
3. Isolate UI for testing
4. **Automate UI tests**
  - Continuous integration
  - Make it easy
5. **Gridify your tests**
  - Generate reports
  - Selenium Grid
  - saucelabs.com

### 1.24.14 Testing large, untested code bases

Dr. C. Titus Brown

#### tools

- figlead

#### The code base

- 8k of python, 2k of pyrex
- Little app/UI code almost all library and framework
- Grew of accretion and personal use at UCLA
- Lots of technical debt

#### Issues with code base

- written at high level throughout
- lots of unfamiliar code
- functional
- Mix of developers
- Crosses domains
- Performance is critical



### What we already know

- Unit tests and functional tests rock: use ‘em
- Code coverage is of limited utility because it doesn’t measure branch coverage
- **Code coverage is invaluable when aimed at:**
  - new test efforts on legacy code
  - understanding code bases
  - legacy code is code that does not have systematic tests

### Software forensics

- understanding big code bases is hard
- **Code coverage can be used as a lever:**
  - “Give me but a place to stand and with a lever I will move the entire world”

### Grokking code thru coverage

1. Start with minimum useful statement
2. Examine code that’s actually executed
3. Add additional statement
4. Examine executed code
5. Repeat

## 1.24.15 The State of Django

---

**Note:** Interesting to see how this differs almost 3 years later. 2011/12/19

- Django is 5 years old
- 

### What has happened since?

- Huge growth
- Merged models and admins
- DjangoCon
- Google App Engine

## Django 1.0

- Backwards incompatible
- Changed Signals
- QSRF (queryset refactor branch)
- Pluggable file storage
- newforms is now forms
  - Modelforms
  - Formsets
- Session backends
- File uploads work reliably
- autoescape
- Databrowse
- geodjango
- Runs on:
  - Jython
  - PyPy
  - IronPython\*

## Django 1.1 beta

- Aggregates
- Annotations
- Query Expressions
- Unmanaged models
  - Lets you subclass User and other things.
- Condition view processing
- Admin
  - Expose particular fields as editable
  - Define bulk actions

## What next?

- Django 1.1 final
- GSoC
- Multiple database support
- EuroDjangoCon in May
- USDjangoCon in the fall

- Django 1.2 in november
- Py3k thoughts
  - Support for python 2.3 will fade
  -

### 1.24.16 Using Windmill

A neat browser test tool that does what Selenium does but better

#### Windmill IDE:

- Supports IE, FF, Safari, Opera
- Controller API
- **Tools**
  - DOM Explorer
  - Assertion explorer
  - Firebug light
  - Firebug
- Results and performance output

#### Writing tests

- Building, recording and editing tests
- Playback and debugging IDE
- **Test languages**
  - python
  - JavaScript

#### Running and Debugging

- You can do it from the command line
- shell environment
- results and performance output
- Python proxy thingee

#### Continuous Integration

- Stop on failure
- Python Logger
- **Reporting**

- JUnit Compatible.

## 1.25 Plone Conference 2008

- Started: Wednesday 8 2008
- End: Friday 10 2008

Venue: Ronald Reagan Center, Washington DC

### 1.25.1 Day 1 - Keynote

#### **With Software as a service, is only the network luddite free?**

The history of computers and freedom

Date: Wednesday 8, 2008 Bradley M. Kuhn

#### **In the Beginning**

- There were computers
- There were users
- ...and users had freedom

#### **Ye Old Four freedoms**

- To learn
- To cope and share
- To modify
- To share modified versions

#### **Then freedom eluded us**

- Software licensing stuff
- Who invented software licensing?
- Gates convinced IBM to license rather than acquire DOS

#### **Who fixed the problem**

- Richard Stallman
- Lived through the golden age
- Programmed on old systems

## The Golden Age

- Academic computer
- software sharing
- no licenses

## MIT AI Lab

- Discovered that patents == money
- Spin-offs == \$\$\$

## GNU

- changed the world
- For the user of individual computers
- No reason now not to have freedom on your own machine

## Internet changes little

- Client/server computer model
- Freedom implications are basically the same
- Email example

## Email: The MTA

- Free email software
- Plenty of proprietary ones
- RFCs define interoperability
- We reverse-engineer RFC-less protocols

## Email: Maul User agents

- Yours has freedom

## What changed

- The browser delivered applications
- AJAX made it more powerful
- Where is computing done?
- Computing in the cloud
- Are our freedoms under threat by cloud?

## Email become gmail

- The experience
- Effectively thin client
- Mixes up free again: Price vs. Freedom
- RFCs no longer enough
- We are back to proprietary “lock-in”
- Other examples is twitter, amazon cloud, etc

## Users and the cloud

- Clouds affect us even when we don’t really we are using it
- Richard Stallman is nervous

## What’s the challenge?

- Not merely a question of code and licensing
- A question of code/data control
- Power to move code/data/people and transport it to somewhere else
- Autonomy: of code, data, community
- Example: Trying to leave Twitter

## How do we start dealing with this challenge?

- Plone is GPL which is good
- Plone is 100% open source

## Keep going

- User communities need ability to move
- Reclaim your data and relocate your community
- This is **tough** programming

## Projects to look at

- **identi.ca (laconi.ca):** Twitter replacement
- **Prophet: Distributed database for web applications**
  - Move community via sneaker-net, ideal for China
- **Ourselves**
  - **Affero GPL**
    - \* Extends copyleft to network service world

- \* Handles the code side well
- **Techniques to look at:**
  - \* Deployed applications auto-give users source
  - \* Data is downloadable in community-chunks

## Future of the Cloud

- Disjointed but integrated
- portable
- Developers decide next direction
- Ask if we are respecting user's freedoms
- Data belongs to users, and we are merely custodians

## More info

- <http://autonomo.us>

## 1.25.2 Day 1 - Feed the Masses

- Introducing Vice - Outbound syndication in Plone via the Zope Component Architecture
- by Paul Bugni
- Wednesday 8, 2008

## Agenda

- Syndication
- Vice History and Demonstration
- Zope Component Architecture
- Vice Integration

## Syndication

- Providing other ways for your site to provide content besides the browser
- Great for frequently updated data
- Web feeds, rss, atom, etc
- **Allows aggregators to target your site/feeds**
  - Net News Wire
  - Feedburner
  - Google reader

## Prevailing syndication formats

- RSS 2.0
- Atom 1.0

## Exposing the myth of RSS compatibility

- There are 9 different and incompatible versions of RSS
- FeedParser handles them all
- RSS has always had many branches
- Check out wikipedia for map of RSS branches

## Atom

- Designed to be done via consensus. No branches!
- Atom has required fields
- Atom is gaining momentum because its easy to work with

## Plone Syndication

- RSS 1.0 is the current view
- PLIP #128 is there to support RSS
- Uses Zope Adapters (via Five) to provide new views

## VICE

- Sponsored by GSOC
- P4A multimedia support
- Plone 3 only

## Installing VICE

- Add to the build.cfg

eggs = vice.plone.outbound fake\_zope\_eggs = True

## Features

- Allows for recursion on objects, their children, and their children's children
- By content type can select RSS, ATOM, or both
- And even RSS 1.0
- Can syndicate any container



## Goals

- Separation of concerns
  - easy to add new feed types
  - make it easy to add new attributes
  - Isolate complexity into manageable objects
- 

I need to bring in my icky one for Chris cause that at least runs cool

### 1.25.3 Day 1 - Plone Developer

- by Rob Porter

Here we go!

#### Useful Tools

- Web Developer
  - Turn off web caching
- Firebug
- plone.reload

#### Plan

- Completely reformat Plone
- Show number of days until X-Mas
- Use a minimum number of images and CSS to make changes

#### viewlets.xml

- lets you define where things are going to be in buildout

### 1.25.4 Day 1 - Lightning Talks

Last minutes of the day

#### Special Announcements

##### Nature Conservancy Announcement

- Announcing Conserve Online
- **Workers**
  - Rick Minor

- Sally Kleinfeldt
    - etc
  - **Neutral forum where conservationists around the world can work together**
    - Work spaces
    - Discussions
    - Calendars
    - Security
  - **Publishing**
    - provides Digital Object Identifiers (DOIs)
    - Clearing house
    - Collective
    - Collaborative
  - **Get it!**
    - GPL
    - <http://sourceforge.net/projects/conservonline>
- 

## Lightning Talks

Here we go

## KaizenPlone

- Way too many options in Plone
- Works well for Plone experts, not for others
- Its a FAQ
  - Page per problem
  - Paragraph per solution
  - Names underneath to vote
  - Sort of like Stack Overflow
- Not out yet

## atomising by Tarek Zaide

- Fetches RSS feed and sticks into a Sqlite3 database
- Managed via simple configuration file

## Cluemapper by Nate Aune

- Trac causes pain
- Cluemapper
  - Project management tool
  - Extension to Trac
  - Personal dashboard
  - easily create new projects
  - Hour management
  - Pretty mockups
- Cluemapper tech
  - Dojo AJAX framework
  - WSGI
- Timetracking rocks!
  - Handles hours
  - pretty graphs
  - collates
- Should replace the crap that is STACR
- cluemapper.org
- cluemapper.blit.tv

## PyPI Replication Project

- Buildouts for Zope highly depend on PyPI
- PyPI is a single point of failure (SPOF)
- Need to replicate PyPI
- z3c.pyimirror
- Allows you to mirror PyPI on your local machine
- About 5 GB disk space, 150 GB traffic per month
- Phase 1
  - About to go live in November 2008
- Phase 2
  - Refactored
  - Auto-mirror selection in buildout!
- <https://launchpad.net/pypi-mirror>

### Proposed improvements by Calvin Hendryx-Parker

- Make any folder a root plone site!
- Do so by adding a single interface!
- Answers ways to install 300-400 Plone sites
- Get buildout quick reference card
- Plugged for November buildout class thingee

### Net Site Lightning Talk

- Minor CSS tweaks to make editing more close to the final results
- Wrote a more elegant, useful Plone edit toolbar system

## 1.25.5 Day 2 - Agile Process

by Mike Robinson

### Introduction

- Agile is a conceptual framework
- Agile does have documentation
- Agile methods have been around since the 1990s and were united by the agile manifesto.
- Agile empowers the workers
- Agile knows their finished when they are finished
- Agile is:
  - Like driving a race car, you know the course but can't account for all the variables

### Agile Values

- Individuals and interactions > process and tools
- Working software > comprehensive documentation
- Customer collaboration > contract negotiation
- Responding to change > following a plan
- Planning is more important than plans

### Agile Principles

- **Highest priority is to satisfy the customer**
- **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage

- **Deliver working software frequently.** 90% completion is a lie. Its either not started, working, or finished. If you do not deliver working software you are of no value.
- Business people and developers must **work together** daily
- Build projects over **motivated people**
- Give developers the support they need and trust them to **get the job done**
- **Working software is the primary measure of growth.**
- Information is best done in **face-to-face conversation**
- Agile processes promote **sustainable development** by not overworking people to meet deadlines.
- The sponsors, developers, and users should be able to **maintain constant speed**.
- **Simplicity** is really awesome. Code is a liability to any organization. The less code you can deliver the better.
- The best architecture, requirement, and designs emerge from **self-organizing teams**.
- At regular intervals the team reflects on how to become more effective, then **adjusts its behavior accordingly**.

### Why use agile?

- Agile lets you meet what the marketing folks want to give to customers
- Agile means good for business so you can plan out hours better
- Operations likes it because your updates are simpler
- Development likes it because it empowers developers

### Developers need to be respected

- Don't agree on schedules without developer input
- Don't agree on tasks without developer input
- Developers need to be honest in describing what needs to be done
- Get developers to demo their work to the customer!

### Charging for the assessment phase

- Bill for it
- Tell customers that you'll bill for planning as long as the customer is willing to pay

### Roles on a team

- Product owner: Drives the project
- Project Manager: Handles the resources
- Team: Developers
- Stakeholders: Customers
- Users: obvious

## Artifacts

- Impediment list (use stickies on the wall to be obvious)
- Project iteration
- Working code

## Agile requirements analysis, estimating and planning

- Identify the users
  - Gather requirements from the users via
    - \* Interviews
    - \* Questionnaires
    - \* Observation
    - \* workshops
  - Ways of documenting
    - \* Use cases fixed the ATM returning cards after cash
      - too big to plan for and measure with
      - prone to include UI details
    - \* User stories
      - a tiny bit of information.
      - You can attach these to use cases
  - Estimating development
    - \* Prediction is difficult!
    - \* You will never be 100% accurate
    - \* Short efforts on estimates are accurate
    - \* Long efforts on estimates are always off
    - \* Breakdown tasks into manageable chunks
    - \* Estimation performed by development team
    - \* Deriving an estimate
      - Expert opinion
      - Analogy
      - Planning poker
    - \* Story points are a relative measure of size of a story. 10 points is more than 5.
    - \* Ideal time it would take to complete a task without interruptions. A football game is 60 ideal minutes and 120 minutes with interruptions
  - Planning poker
    - \* Each member gets six cards

- \* People put the value they think it will take down on the table. The most common value is how long it will take in story points.
- \* If one person has things way off, then talk out why there is a discrepancy
- \* After each task has point assigned, figure out how long a point is worth. Use previous effort to determine the length of a story point.
- Prioritizing user stories
  - \* Priority assignment is the primary responsibility of the Product Owner
- Velocity
  - \* Measure the rate of progress of a team
  - \* Amount of story points completed in the last iteration
  - \* Next iteration = same as last iteration (“yesterday weather”)
  - \* velocity corrects estimation error
  - \* Accommodate developer optimism
  - \* Burndown chart
    - Plots the amount of committed effort left against the time left to complete the iteration

Agile planning game

### 1.25.6 Day 2 - Bringing Open Source Practices to Educational Enterprises

moving k-12 to the modern era

#### Can’t do business with a web site that sucks

- Investigate: <http://clubs.psu.edu/up/taxidermy>
- Can’t afford the old school ways
- Chose plone because it gives a lot of web 2.0 model of user generated content

#### Ecosystem

- Engage learners, teachers, families, and communities
- Modernizing business processes
- Cleaning up a bucket of organizations
- Diverse organization
- Platform to build some homogeneity on
- Many little bunkers of IT staff
- IT at Penn state is distributed (not)
  - IT at Penn state is actually disorganized

## Behaviors and Structures

- Policies
- Standards

### 1.25.7 Day 2 - Consultant Talk

#### Figuring out charge rates

- Don't lose money
- Fixed price for very short projects only
- Give estimates for x number of days at x price
- Optional scope reevaluations can be critical
- Give a percentage rate on things you don't know

### 1.25.8 Day 2 - What makes a great software development team

- by Mike Robinson
    - He can code
    - He can manage
    - Irish
    - Cyclist!
- 

#### What is the problem?

##### From the customer's point of view

- Its expensive
- Its not what they wanted
- Its unpredictable

##### From the development team's point of view

- Its mentally hard
- Don't feel appreciated for their smarts
- It is hard work and the hours are often long, inconsistently hard
- The business doesn't know what they want



## Requirements

- Sometimes not technical enough
- Sometimes too technical

## Process is not the answer

- More control eats up engineer team

## Technology is not the answer

- Traditional or service doesn't answer

## Tools are not the answer

- trac is not the answer

## People are the answer

- Great people are what you want
- Great people need process, technology, tools

## Different process methods

- Pick the one you want, it doesn't matter
  - Principles and ethics are what is important
  - Values of the developer
- 

## What Mike Robinson thinks will work

### People

- Change
  - Requirements will shift
  - Scope will migrate
  - Don't think what you know will always work
  - Technology mutates over time
  - The market changes
  - Embrace change!
  - Work in such a way that change does not kill you

- \* Think like a sculptor in that you refine things over time
  - \* Simplicity is king! If you start complex, it will be harder to enhance later
  - Quality!
    - Do things right
    - Testing early, often, well
    - Are you building the right solution to a problem?
    - Get feedback
  - Commitment
    - Everyone has to be committed
    - Give them sense ownership
    - Make developers feel like what they are doing will make a difference
  - Visibility
    - Let people see what you are doing
    - Give constant status updates
    - Metrics are really important
    - You are either not started, started, or done. No statements of percentage completes! No more 90% or 75%
  - Collaboration
    - Get your people into feeling they are part of the team.
    - That includes developers, managers, testers, etc
  - Focus on Value
    - People need to be told what is considered to be of value
    - Developers need to be instructed as to what parts of what they do will be shown to have value
    - Got to see why what you are doing is important
    - If its not of any use, why do something?
- 

## Process

- How do you pick the right process for your time?
    - No one process is good for all teams.
    - Based on values of your effort, pick the simplest approach to process for your team
  - Review and adapt your process to see if it needs to be altered
  - Evolve your process as needed
-

## Conclusion

- No one method fits all projects, all the time

### 1.25.9 Day 2 - Summary

I was very rested by the time I arrived. I had slept well.

#### So you want to be a consultant by panel

No, I'm not looking to leave NASA these days. However, I do some side consulting, and even in my day job there are good project control lessons you can learn from the consulting crowd. For example, ideas on recruiting, customer relations, and how to handle billing when you need time to boost skills. There were 5 plone board members in the panel, and much wisdom was shared. Plone board members there:

- Nate Atune
- Gaer Baekholt
- Calvin Hendryx-Parker
- Jon Stahl
- Someone else whose name escapes me

#### What makes a great development team by Mike Robinson

He can code, he can manage, he can cycle, and he has an awesome Irish accent, and he taught us about agile programming. And he explained why and how it works in a great fashion. I'm completely sold. I have been for some time, but he gives great arguments for it, or at least on evaluating on how to best handle this sort of thing.

#### Future of the Plone user experience by Alexander Limi

Limi has strong opinions and what he said may not reflect what ends up happening. He wants deliverance, strong media handling, more widgets, deliverance, better kupu, z3c form improvement, better validators, and an easier way to handle templates. All good stuff.

And he had a much, much better version of what I've been aiming at with my customer editor view in NASA Science. Great minds think alike, although I must admit his method is much better than my own.

#### Simplifying Plone by Martin Aspelli

He wants to do it with a chainsaw and make Plone more approachable from a developer's point of view.

Plone has a number of embarrassments, with issues ranging from rich media support to import/export problems with the database. It is hard to learn and skinning is a challenge. He thanked Lennart for pointing out what Zope did wrong, and how developers expect things to be easy.

Wants to revamp much for Plone 4. Some quick bullets:

- Follow the guiding lights shown by some of the other Python web frameworks.
- Make learning follow a constant set of humps, not huge ones followed by a plateau followed by an insurmountable wall.

- make certain things real easy to do (logo, branding, content types, etc)
- Create one true way and remove the other ways
- embrace through the web but allow filesystem round-trip for deployment and collaboration

### **NASA Science case study by Katie Cunningham and Daniel Greenfeld**

Yes! I helped present!

So we presented and apparently did very well. We had a few luminaries in the room, including a couple Plone board members. I think we nailed all the points we wanted to make, which was a very awesome thing to do. We plan to send the slides off to Alex Clark shortly so we can have them on line for everyone to see.

### **Evening Agile Development workshop by Mike Robinson**

Two more hours of the awesome Mike Robinson ended the day for me. He gave a rock-solid lecture and then we played a game to support his statements. It was a fun game and learning was had by all. That said, I think this would have been better done as part of a day-long class, not at the end of a long day of conferencing.

### **General Socializing**

Where to begin? I had great fun with so many incredibly awesome people. The quick and dirty list:

- Vernon Chapman
- Tarek Zaide
- Alex Clark
- Amy Clark
- Matt Bowen
- Jon Stahl
- Nate Aune
- Katie Cunningham
- Gary Burner
- Joel Burton
- My whole agile development team

If I missed you, let me know!

### **1.25.10 Day 3 - Buildout by Clayton Parker**

- Buildout
  - parts
  - recipes
  - command-line
- ZopeSkel
  - Custom recipes

## Why buildout?

- because it rocks
  - lets us fetch all the dependencies easily
  - no more ugly checklists
  - lets us put everything involved in setup into one configuration file
- 

## Syntax

- Variable substitution
    - `${part:option}`
  - option additional and removal
    - `options = foo bar options += foo options -= bar`
  - Reserved characters
    - `:%{ }`
    - Don't use those things
  - `http-address: 11001`
  - `zoo-address: 10001`
- 

## Parts and Recipes

- Can't have a part without a recipe
- Part is identified in brackets:
  - `[plone]`

## Buildout

Some notes about buildout:

```
[buildout]
eggs-directory = where to put eggs
download directory = where to put downloads
zope directory = where zope should go
index = http://download.zope.org/ppix (for PYPI mirrors)

[instance]
event-log-level = debug
```

## Recipes

Find recipes on...

- PYPI
- collective

Plone recipes

- plone.recipe.plone
- plone.recipe.zope2install
- plone.recipe.zope2instance
- plone.recipe.squid

plone.recipe.zope2install:

```
[zope2]
recipe = plone.recipe.zope2install
url = ${plone:zope2-url}
fake-zope-eggs = true
additional-fake-eggs = ZODB3
skip-fake-eggs =
    zope.testing
    zope.component
    zope.i18n
```

---

## Extending Configuration

- buildout.cfg
- profiles
  - base.cfg
  - development.cfg
  - debug.cfg (high debug settings)
  - qa.cfg (testing tools included)
  - prod.cfg (squid, varnish, etc)

how to extend:

```
[buildout]
# profile we want to use
extend = profiles
```

## PIL integration!!!

doh:

```
[buildout]
parts =
    PILwoTK

[PILwoTK]
recipe = zc.recipe.egg
find-links = http://download.zope.org.distribution
```

## Handy tips

TODO: get all this stuff. Some handy stuff from this talk:

```
[instance]
environment-vars =
    TZ America/New_York

[debugging]
parts =
    debug-products
    debug-products-svn
    ipzope
    zope
eggs =
    plone.reload # real handy for development!!!
    Products.PDBDebugMode
    Products.DocFinderTab
    Products.Clouseau
    Products.PrintingMailHost # sends mailhost messages to console instead of to ↵
↵email!!!
zcml =
    plone.reload
```

## Useful command line tools

- [ipzope]
  - sets up ipython for Zope without the ugliness!!! Find this full setup!
  - Lots of handy featur
- [zopepy]
  - Python prompt with all the Zope eggs in it but doesn't start up zope.
  - Great for command-line stuff without the weight

## Versions.cfg

- Helps us control versions of everything in one simple file.

## collective.recipe.zope2cluster

- Controls instances

---

## Creating recipes

```
$ paster create -t recipe my.recipe.example
```

Recipe really just consists of:

```
class Recipe
    def __init__(self, buildout, name, options): pass
    def install(self): pass
    def update(self): pass
    def uninstall(self): pass # find conditions for things
```

## Question

- Plone Deployment workshop (Indianapolis Nov 19-21)
- Creates a plone site in your Zope! #asked by me!!!
  - collective.recipe.plonesite

## 1.25.11 Day 3 - Buildout with Tarek

- Expert Python Programming book
  - TODO: Buy it at conference and get a hug
  - NOTE: Bought it and still have it as of September 2012
- Did Nuxeo and Zope
- got sponsored by the Plone community to come!

## Part I - working with packages

### distutils

- Builds and distributes a package, registers and uploads it to PyPi
- screencast on <http://ziade.org/ploneconf>
- distutils is the standard of the whole Python community
- Alas, distutils is broken
  - but there is hope

### setuptools

- simple dependencies management
- namespaced package
- egg distribution
- provides easy\_install
- python setup.py bdist\_egg
  - Eggs are to python as Jars are to Java
  - egg = deployment format in a zip archive
  - look up “sdist” which is what Tarek recommends



- python setup.py sdist
  - awesome sauce
- easy\_install my\_package
  - will get and install “my\_package” from PyPI

### **Grrr... but setuptools is broken**

- setuptools is broken
- funny stuff inside
- packaging future is uncertain for python

### **Get the community behind packaging system!**

- Django
- Turbogears
- etc
- Python at-large

### **Problems with packaging**

1. PyPI == SPOF
2. packages need privacy sometimes
3. plone.org/products is dying

### **PyPI mirroring**

- Make a smart mirror so it updates all the mirrors
- easy\_install collective.eggproxy
- Run your own private PyPI mirror!
  - Using Plone as a host!

### **collective.dist**

- python 2.6 new “register” and “upload” commands
- This lets you use these commands in older versions of Python
- Lets you push PyPI mirrors with easy config files

## Make plone.org/products PyPI compatible

- Plone.org was suppose to switch to that for months. Lazy guys

Big picture:

```
My product  ----> plone.org
              ----> python.org
              ----> your company/agency
```

## summary of part I

1. Make mirror
  2. Run your own PyPI
  3. Push to several servers
  4. Use “mregister” and “mupload”
- 

## Part II - working with `zc.buildout`

local mirror:

```
[buildout]
index = http://my.mirror:8888
```

## 5 hours in 2006

- took 5 hours to get a buildout running
- Developers were engineers

## 5 minutes in 2008

- get the buildout
- `$ python bootstrap.py`
- `$ bin/buildout`
- start to work

## Not the main purpose for the creation of buildout.

- Reason was eggification of Zope
- Now we can updates on individual eggs rather than the whole stack
- Plone is following the same path
  - Plone pollutes Python site-packages
  - But `zc.buildout` isolates the plone environment

## zc.buildout best practices

1. Use the same layout for all your projects
    - folder layout ingunieweb uses
      - docs
      - buildout
      - packages
      - releases
    - collective.releaser is what they use to handle releases
  2. make sure all developers have the same environment
    - Windows developers are a problem
    - Get the windows installer: python2.4.4-win32.zip
    - Google “An installer for a buildout-ready Windows”
    - This should resolve the Windows issues
  3. use on cfg per target
    - Typical buildout layout uses the **extends** feature
      - buildout.cfg
      - dev.cfg (extends buildout.cfg)
      - prod.cfg (extends buildout.cfg)
      - bootstrap.py
- 

## Part III - application lifecycle

### releasing packages the old way

sample:

```
for package in packages:
    raise the version
    edit CHANGES.txt
    create a branch
    push to various mirrors
    make some code edits
    eggify stuff
    deploy
    etc
```

### releasing package the collective.release way

1. Do a config file thing!
2. sample:

```
for package in packages:
    release package
```

## 1.25.12 Day 3 - KSS by Joel Burton

- Joel is not a JavaScript expert

### Overview

- Why KSS
- Client-side convenience
- Server-side power
- Quick reference guide

### Challenges of JavaScript

- Another language
- Browser incompatibilities
- Even when done correctly...
  - Do it again in Python

### What is KSS?

- **Kinetic Style Sheets**
  - Power of JS, syntax of CSS
  - Allows you to declare behavior
  - Includes AJAX library

### KSS sample 1

basic example:

```
#logButton:click { /* identifier & event */
    action-client: alert; /* client action: alert */
    alert-message: 'clicked'; /* parameters for alert */
}
```

### KSS sample 2

More complex example:

```
#logButton:click { /* identifier & event */
    evt-click-preventdefault: True; /* Don't do normal thing */
    action-client: replaceInnerHTML; /* client action: alert */
    replaceInnerHTML-kssSelector: "#message"; /* */
    replaceInnerHTML-html: "Clicked (via KSS)"; /* parameters for alert */
}
```

## Registering KSS

- portal\_kss in zmi
- storing just like css

## Timeouts

- **Makes it so you can in-line page auto updates!**
  - Stock market changes
  - Chat system

## events

- click
- dblclick
- load
- mouse events
- timeouts
- blur, change, field things
- TODO: look up text change!

## coreset

- replaceInnerHTML
- insertHTMLafter
- addClass: add css class to html
- removeClass
- toggleClass
- focus on a given node

## zope set of commands

- refresh a viewlet
- zope.refreshViewlet (...stuff...)

### plone portlet functions!!!

- refreshPortlet
- issuePortalMessage

### Debugging KSS

- **Use Firebug!**
  - turn on portal\_javascript debugging
- type in your scripts directly

### Future of KSS

- **Other JS libraries**
  - for people who want to tinker deeply
- **Possible non-JS backends?**
  - Flash
  - Silverlight
- <http://plonebootcamps.com/resources>

## 1.25.13 Day 3 - Lightning Talks

### CSS Manager by Rob Porter

- Lots of handy push button tools for themes
- Puts changes in the custom folde

### Grok + Dexterity by Martin Aspelli

- showing something with grok and dexterity
- Grok inside of Plone!!!

### Plone Tune-Ups by Calvin Hendrix-Parker

- What is it?
  - Short sprints done one fridays
  - #plone-tuneup on irc.freenode.net
  - dev.plone.org
  - problem tags
    - \* newbie
    - \* green belt
    - \* black belt

- \* big bugs
  - for whom?
    - \* anyone
    - \* css experts
    - \* qa people
    - \* documentation
- why?
  - chance to give back to community
  - advance plone

### Fabric by Aaron Van Der Lip

Because Aaron is Lazy...

- computers are for automating tasks, not for creating tasks that automate us
- A framework for handling repeatable tasks via ssh

### Zope on Python 2.6 by so and so

- We get to have Zope on a modern server

### Photogallery for repoze.bfg by ???

- Not everything should be done in plone
- Some things should be done in repoze.bfg
- repoze demo gallery

## 1.25.14 Day 3 - Summary

### Day 3 summary

The night before I had weird dreams about angry pets. Angry dogs and angry cats. Very odd. I got out the door a little later than planned and rushed to get there to support the conference. Arrived and they seemed okay for staff and effort.

I did meet Shaun Saphton when walking in to the RR building. He is a certain South African poster on my blog who asked about NASA Science timeliness issues. We chatted and I got a couple others involved. He and his company work under murderously short deadlines. It seems that they have a good system in place to handle things, but one thing I noted was they had issues with skinning on time. Very interesting and from a very different point of view compared to our 6 week-ish long cycles.

That he had to post anonymously and could not easily provide his real email is yet more proof that I need my own real web site on a real web server. I want a simple blog that gives the features I want.

## **KSS by Joel Burton**

Joel gave a good, quick explanation of the why and how of KSS. I knew most of it but there were a few gems I picked out of it that I turned right into working functionality for an internal NASA product.

## **Buildout by Clayton Parker**

I thought I had been getting pretty good with buildout, but Clayton of Six Feet Up demonstrated how little I really knew. Thanks to him my skills really exploded forward. I am very happy right now with what I got from him. I may go to the buildout workshop in Indiana next month.

## **Distutils and more Buildout by Tarek Zaide**

Tarek gave a great overview of distutils and buildout, explaining the current status quo, how things are moving along, and the pains of getting python core people to improve certain things. He talked about the new products section of plone.org, mirroring PyPI, and lots more.

## **General Socializing**

More fun and yet another quick and dirty list in no particular order:

- Tarek Zaide
- Alex Clark
- Amy Clark
- Matt Bowen
- Jon Stahl
- Vernon Chapman
- Katie Cunningham
- Gary Burner
- Chris McDonough
- Chris Perkins (TurboGears)
- Mark (TurboGears)
- Calvin Hendryx-Parker
- Gabrielle Hendryx-Parker
- Alexander Limi
- Martin Aspelli
- Hanno
- Harito
- Darci Hanning



### 1.25.15 Joe Developer Class Curriculum Idea

---

**Note:** This was my idea during the conference. In essence, a class for the rest of us.

---

#### Goals of the class for our firm

1. Teach plone to more people
2. **Teach Plone A-Z using latest stable technologies**
  - buildout
  - Zope 3 style products
3. Have fun
4. Make enough money to get some Taco bell
5. Prospect for potential customers for consulting
6. Prospect for potential employees for consulting
7. Earn me street cred in Plone community

#### Quickie view of the curriculum

- **Class Pre-Requisites**
  - Basic knowledge of python
  - laptop, Mac OSX or Linux preferred
- **Our Setup**
  - Local Pypi mirror that we bring in to remove dependency on local internet connectivity
- **Assembling the tools (Class starts)**
  - python
  - easy\_install
  - paster
  - SVN
  - PIL
- **Buildout - Plone!**
  - Configuring our Plone instance with buildout.cfg
  - Building Plone
- Setting up a plone instance in the ZMI
- **Buildout - Product!**
  - Using Paster and ZopeSkel to build a sample product
  - Pointing buildout.cfg at our new product

## Class Pre-Requisites

- Basic knowledge of Python
- Laptop

## Our Setup

- Local PyPI mirror to expedite buildouts and remove dependency on network connectivity by host
- Projector

## Assembling the tools

- Correct version of Python
- easy\_install
- Paster
- SVN
- PIL

## Buildout

- Create your product
- Doing the bootstrap

## 1.25.16 Products to investigate

- GloWorm
- plone.reload
- Firebug
- webcoutier
- TAL portlet
- PloneFormGen
- Vice
- plone.z3c.form <http://plone.org/documentation/how-to/easy-forms-with-plone3>
- Deliverance for skinning - what do we gain?
- Jon Stahl on video embedding
- ipzope (ipython for zope)
- zopecy (python prompt with zope eggs built in, but no zope instance, so good for introspection)
- collective.release
- collective.dist
- testcoverage

## 1.25.17 Sprints

---

**Note:** I teamed up with Katie Cunningham in order to contribute to P4A.

---

### Projects I considered

- KSS (continue from last year)
- Generic Setup (Wanted to see it improved so configuration of Plone would be easier)
- Vice
- P4A (Package of libraries called 'Plone 4 Artists')

### What I did

```
Work some test tickets
#37 Provide alternative displays of ordering the audio tracks
- Drag and drop
- order fields
```

### What others did

- Faculty staff thing
- PloneFormGen
- Theming sprint (maybe)
- SchoolTool (Zope 3 school info thing)
- ZopeSkel
- Documentation sprint
- beginner's sprint
- Vice
- KSS
- Dexterity
  - Users
  - Integration
  - User interface
- Plone 4 Artists (what we did)
- Remote Include

## 1.26 Pycon 2008

Venue: Chicago! My first PyCon!

## 1.26.1 A database in the clouds

### Using the Google Spreadsheets API

- Lots of Python community support
- [http://docs.google.com/Present?docid-dcmg89gw\\_70g6ct9jcz](http://docs.google.com/Present?docid-dcmg89gw_70g6ct9jcz)
- <http://spreadsheets.google.com/ccc?key-pKq0CzjiF3YnZ5oeOipDoEA&hl-en&pli-1>

### Why?

- Server-side - streaminline server
- Portable

### Challenges

- Might be slower
- Requires internet connectivity
- Caching helps and gives you best of both worlds

### Why google spreadsheets?

- Free
- Available anywhere
- Scalable
- Restful API
- Nice UI

### GoogleSpreadsheet API

- Web service based on AtomPub
- Series of related Atom Feeds

### What sort of Database

- Non relational
- Supports queries
- Multiple concurrent users
- access controls

## Simplified Interface

- gdata.spreadsheet.text.db
- Keep it simple
- for mashups
- text data only

## Searches

- Ranges, equals, and more fun

### 1.26.2 What is a callback?

- Its all about framework code that ‘call back’ into your code
- Traditional code uses Hollywood principal, where “don’t call us, we’ll call you”
- Callbacks are used for event-driven architectures (actual events, structuring of control flow)

## Callback implementation

- Give someone a callable
- Someone may store it somewhere
- container, attribute, whatever
- or keep as local variable
- Calls when appropriate
- When it needs some specific functionality (for customization)
- Or when appropriate events occur (state changes, user actions, etc)

## Customization

- Customizing sort (by key)
  - Read up on DSU pattern
  - Note that a little workaround is needed with the usual ‘call a method on each object’ OO idiom

```
def DSU_sort(mylist, key):
    aux = [(key(v), j, v) for j, v in enumerate(mylist)]
    aux.sort()
    mylist[:] = [v for k, j, v in aux]
```

## OO Customizing: the TM DP

Template method design patter: perform the callbacks by self delegation:

```
class Tmparent(object):
    self.somehook()
```

And customize by inheriting & overriding:

```
class TMchild(TMparent):  
    pass
```

## Customizing scheduling

- Sched needs TWO callback functionalities:
- What time is it right now?
- Wait (sleep) until time T
- The OO way (more structured):

```
import time  
s=sched(time)  
  
* the FP way (more flexible):
```

```
s=sched(time.time,time.sleep)  
  
* You might supply callbacks or not  
  
* (Dependency Injections DP & variants)
```

## Events

- Events proper
- Observer/observable design pattern
- GUI frameworks (mouse, keyboard)
- asynchronous (event-driven) I/O
- System-event callbacks
- Pseudo-events
- parsing (SAX and other long tasks)
- scheduled (sched)
- concurrent (threads)
- timing and debugging (timeit, pdb)

## The Observer DP

- Target object lets you add observers
- could be simple callables or objects
- when the target's state changesm it calls back to the left observers know
- Design choices:
- General observers (callback on any state change)

- Specific observers (callbacks on specific states)
- grouped observers (objects with >1 methods for kinds of state-change)

## GUI frameworks

- Most classic of event-driven fields
- consider Tkinter:
- elementary callbacks for buttons
- flexible, advanced callbacks and events (pressing the ‘a’ button)
- can also bind by class, all, root window, etc

## Callback issues

- What arguments are to be used on the call?
- No arguments: simplest, a bit rough
- In observer: pass as argument the target object whose state just changed
- Or: a ‘description of the state changes
- Saves ‘round trips’ to obtain them
- other: identifier or description of event
- but? WHAT?!? check the PDF

## Fixed args in callbacks

```
functools.partial(callable, *a, **kw)
    prebind any or all arguments
    x.setCbk(f, *a, **kw)
```

## Callback dispatching

- What if more than one callback is set for a single event (or, observable target)?
- Remember and call the latest one only
- simplest, roughest
- Remember and call them all
- LIFO? FIFO?
- How do you remove a callback?
- Can one callback ‘preempt’ others
- Can events (or state changes) be grouped?
- use object w/methods instead of callable

## Callback and Errors

- Are errors events like any others
- Are they best singled-out?
- Twisted Matrix's deferred pattern: look this up! It holds:
  - N chained callbacks for successes +
  - M chained callbacks for errors
  - each callback is held with opt (*\*a, \*\*kw*)

## System-events callbacks

- For various Python system events:

```
atexit.register(callable, *a, **k)
oldhandler = signal.signal(signum, callable)
sys.displayhook, sys.excepthook, sys.settrace(callable)

* extension modules to that too:

* readline.set_startup_hook
* set_pre_input_hook
* set_completer
```

## Event-driven parsing

- SAX for XML
- sometimes very big!
- events are start and end of tags
- handlers are responsible for keeping stack or other structure as needed
- often not necessary to keep all
- XML DOM on other side

## Scheduled callbacks

- Standard library sched

```
s = sched.Sched
evt = s.enter(blah blah blah)
s.run() #runs events
```

## Concurrent Callbacks

- Useful for SR check
- `threading.Thread(blah)`
- `stacklet.tasklet` (stackless python)



- `processing.Process`( like `threading.Thread`)
- NWS sleigh: `eachElem`, `eachWorker`

## Timing and Debugging

- `timeit.Timer(stmt, setup)`
  - string arguments to compile and execute
  - dynamic language twist on callback
  - event for callback
  - setup: once before anything else
  - stmt: many times for running
- `pdb` module
- `pdb.run` and `.runeval`: strings
- `pdb.runcall`: callable, arguments

## 1.26.3 Django under the hood

### Intro

- This is the undocumented stuff under the hood
- Declarative syntax
- Easy ORM
- Extensible template language

### Pluggables

- HTTP middleware
- Database connections
- Cache mechanisms
- Authentication providers
- Context Processors
- Template loaders

### New features

- Easy admin customizations
- Subclassing of queries
- File storage

## Metaclasses

- Class for creating a class.
- Aren't these just factories?
- Signals using PyDispatcher

### 1.26.4 Why Py3k

by Guido van Rossum

- Open source needs to move or die - Mats (creator of ruby)
- To fix early, sticky design mistakes (classic classes, int division, print statement)
- Changing times (str/unicode, int/long)
- New paradigms come along (dict views, argument annotations)

## Major Breakages

- Print function: *print(a,b,file=sys.stderr)*
- Distinguish sharply between text and data
- `b"..."` for bytes
- `"..."` for (Unicode) str literals
- Dict `keys()` returns a set view [`+items()/values()`]
- Mutates as the dict underneath it mutates
- No default `<`, `<=`, `>`, `>=` implementation
- `1/2` returns 0.5
- Library cleanup

## Long anticipated breakages

- Kill classic classes
- int/long unification
- Kill string exceptions
- Raise syntax

## Major new features

- Argument annotations
- *def f(a: 2\*2, b: 'hello') -> 42: ...*
- Abstract Base Classes
- kinda like interfaces
- In Python 2.6+ as well

- Expanded iterable unpacking
- `a, b, *x, y = range(5)` # 0, 1, [2,3], 4
- new `str.format()` method:
- `"got {0}{kind}.format(42, kind='bugs')"`
- got 42 bugs

### What's in it for me

- More predictable unicode handling
- Smaller language
- Makes 'Python fits in your brain' more true
- There's only one way to do it
- common traps removed
- Fewer surprises
- Fewer exceptions

### Enables future evolution

- Examples
- Argument annotations
- `print()` function
- `str.format` method
- abstract base classes
- unicode letters in names

### 2to3 tool

- Context-free source code translator
- Handles syntactic changes best
- Handles built-ins pretty well
- Doesn't do type inferencing
- Doesn't follow variables in your code

### When do we switch

- no hurry! 2.6 will be fully supported for at least 5 years. 2.7 and maybe even 2.8
- Switch when both of these are true
- You are ready
- All your dependencies have been ported
  - PIL

- WSGI
- DB-API
- There are tools to help you switch

### Getting ready to switch

- Start writing future proof code for 2.5
- Don't bother with the trivial stuff though
- Focus on what 2to3 can't do
- Stop using obsolete modules
- Start using iterators and generators
- Inherit exceptions from BaseException

### What about text handling

- Yes, its a difficult issue
- Expect for help by this summer
- Isolate handling of encoded text
- use bytes and b'...' for all data
- Use unicode for all text

### The role of Python 2.6

- Stable, compatible, supported!
- Many 3.0 features backported
- But not text/daat distinction
- Warns about non-3.0-isms with -3 flag
- Especially for things that 2to3 can't fix

### 1.26.5 Iterators in action

- Jim Baker
- [jbaker@zyasft.com](mailto:jbaker@zyasft.com)

### Overview

- Functional LINQ
- Recursvie Generators

## Functional LINQ

- LIN primitives
- Relational Algebra
- map, reduce reduce
- Lots of functionality
- Language INtegrated uery
- Streaming

## Row composition

- `__iter__, next`
- All iterators use it

## Column Composition

- Columns are harder
- Fundamental ops in relational algerbra

## Naming

- Solution to integration in python is naming
- namespaces

## namedtuple

- Subclassing the tuple type
- associated the column names with an index in the tiple
- part of 2.6 - collections.namedtuple # LOOK THIS UP!!!

## Select

- write helper functions
- look up tee - part of itertools.tee

## operator.itemgetter

- extracts multiple items from a seurence

## create join primitives

- hash\_join adding two iters on a defined predicate

## Research items

- `itertools` in general
- `collections.namedtuple`
- `collections` in general

## Concepts for use

- Joining SQL data with other data for mashups
- Best to convert all data to `namedtuples` because then you have more finicky control

## Recursive Generators

- # cookbook recipe 190465
- Graph traversal across edges and nodes
- Workhorse algorithm
- Bone up on `yields` again

## Fork

- 2.5+
- Given a generator runs it in a separate thread
- Look up `fork`

## Global Interpreter Lock

- Not in *cpython* or in *jython*
- `from __future__ import global_interpreter_lock`
- `from __future__ import GIL`
- Not going to happen!

## 1.26.6 Netflix Prive

- Use `Pyflix` library to squeeze dataset into 600MB
- <http://pyflix.python-hosting.com>

## More CPUs

- Some algorithms take weeks
- Need runs over many sets of data
- Problem if your resources are limited

## Beowulf Clustering

- Expensive

## Amazon EC2

- `__init__`(Amazon Machine Images)
- Pay for what you use
- About \$0.10 per hour per small box, 0.80 for that

## Parallel programming in Python

- basic prototyping done in Numpy
- Find clustering stuff to extend it

## ElasticWulf

- Cheap
- copies Beowulf but uses Amazon EC2 to handle stuff

## What is MPI?

- Use Ipython1
- High performance message passing interface (MPI)
- Implemented in multiple languages
- Point to point collective operations
- Very flexible and complex

## Basics of MPI

- Each process has a size attribute: num of operations
- Each process has an id attribute
- `import mpi`
- `local_array = mpi.scatter(my_list)` # runs a list of functions across multiple systems
- `root_data = mpi.gather(local_array)` # grabs the data from the processes

## Getting started

- Sign up for Amazon Web Services
- Get your keys/certs
- Download Elasticwulf python stuff

### 1.26.7 nose and TDD

- Write tests first so you never have any wasted code
- For this, writing and running tests has to be easy
- And the tests have to be useful, especially when they fail

#### How laziness drove the development of NOSE

- traditional unittest is high-friction
- py.test is great, but hard to install and complex
- no boilerplate

#### Basic of Nose

- Extends unittest, doesn't replace it
- use assert to test
- use print for debugging
- generative tests

#### Reasons to use nose

- nose makes it easier to write and run useful tests
- useful tests make it easier to write useful code

### 1.26.8 Why Pyglet?

- Content display
- No extra compilation
- Multiplatform
- Uses an OS' native ctypes
- Well-designed (or so they say says me)
- Handles many screens and many windows in Python so if you have extra monitors
- Ate their own own dogfood (presentation is done in Pyglet)

#### Pyglet!

- Modern graphic cards are powerful
- Showed of mandelbrot sets that were live
- Demo done in eee pc.
- Works on Windows, OS X, and Linux
- Put it all together and you get Pyglet



### Compared to other things (pygame, pyopengl)

- Runs without extra bits
- easier to control
- very pythonic
- Is Python 2.5+

### Components (Linx, OS X, Windows)

- Open GL
- xlib, Carbon, Win32
- gdk-pixbufm Quicktime, GDI+
- Optionally PIL if you have it
- libpng, libjpeg (problematic)
- Audio (OpenAL, ALSA, DirectSound)
- Lets you do lots of fonty stuff with true text fonts
- Linux media is a mess (unreliable installs, wants to own playback)

### Features

- streaming audio and video
- image stuff
- resource loading
- resource loading
- event loop
- graphics buffers
- fast sprites
- fast text and formatted text layout
- animated gif support

### Third parties

- Cocos (A sprite and game engine that runs over Pyglet)
- rabbyt (a fast sprite rendering and animation library)
- ToGepy ()

## 1.26.9 Python for the sys admin

---

**Note:** I wonder how much of this has been subsumed by Fabric and the advent of Chef/Puppet. 12/22/2011

- <http://dev.tummy.com/~jafo/pycon2008/>
  - Monitoring
  - Helpers
  - Automation
  - Accounting
  - Random scripts
- 

### Why Python?

- Perl not comfortable
- Python is comfortable
- They use Nagios and needed custom bits

### Sys admin constraints

- 5% programmer
- Using older versions of Python on older systems
- Security/privilege issues
- Integration (system commands, file-system, GUIs, web)
- Heavy Data processing

### Q&A

- System monitoring & reporting
- They use Nagios
- Backups and recovery
- Configuration management
- User accounts and security / SSH
- Using python for virtual servers/testbeds

## 1.26.10 Turbogears Philosophy

- grab best of breed items and created framework
- Replaced Kid with Genshi (faster, more flexible)

## Why TG2?

- World was moving towards WSGI
- Pylons was already there and had similar components (SQLAlchemy, Paste, buffet)
- Make it easier to understand
- Wanted to get where Pylons was with WSGI
- Share developer efforts with Pylons
- WSGI which allows the fun of Middleware and has a well-defined interface
- WSGI lets you build up your framework from Middleware

## Why not just merge TG and Pylons?

- Pylons likes everything to be independent
- TG likes standard options to speed things up
- (Debian != Ubuntu) == (Pylons != TurboGears)

## Stuff that was shared

- Pylons middleware for exception handling
- Make a TG style controller in Pylons

## Going forward with other frameworks

- Maybe lots of frameworks was good
- Web frameworks need to work better
- Need to communicate better
- Biodiversity is the SIGN of a healthy ecosystem
- Lots of ways to handle different problems like forms and security
- repoze.tm
- repoze.profile
- Stop reinventing the wheel!
- DBSprockets (django admin for other frameworks!) – look this up!!!
- WebOp makes writing middleware easy
- Social prerequisites
- Play nice!
- Respect each other
- willingness to listen
- try other frameworks
- develop a thick skin

### 1.26.11 Roll your own persistence in python

- <http://blog.codejams.net/pycon/>
- Performance boost over Sqlite + SQLAlchemy

#### Example: Invoice

- Invoices have lineitems
- lineitems may be split into accounting codes
- invoice has payments
- Must record amount paid on each accounting code for AR
- Invoice/lineitem
- simple JSON creation

#### Implementation

- Choose a way to persist string keys and string values
- Choose a serialization format
- Add features as ‘middleware’
- add querying by wrapping our db in a MapReduce implementation
- Serve our database over HTTP using WSGI

#### Storage: Roll your own

```
class DictMixin(object):
    def keys(self): pass
    def __getitem__(self, key): pass
    def __setitem__(self, key, value): pass
    def __delitem__(self, key): pass
```

#### Other ideas::

- Use gdata, amazon, or other on-line db

#### Ways

dumbdb:

- written in Python, a flat file
- fallback option

dbm:

- Unix only

gbdm:

- Non-standard format

dbhash:

- BSD DB library

Shove:

- Recommends shove which lets you use various Amazon, gdata and other handy ways.
- Found on cheese shop

## Tradeoffs

- Scaling
- Editing of data

## Serialization

- cPickle
- marshal and repr/eval is not safe
- JSON - yay!
- YAML
- XML

## Serialization tradeoffs

- interoptability
- speed (cPickle, JSON are fast)
- security

## Adding features

```
class UserDict(object):  
    pass  
class JSONSerializer(UserDict):  
    pass
```

Use formEncode to validate a document has the correct schema

## Data replication & versioning

```
class VersionDict(UserDict):  
    def __getitem__(self, key): do magic code  
    def __setitem__(self, key): do magic code
```

## Map-Reduce Querying

- use Google's tool to handle storage and data optimization

## The server

- class HTTPDict(UserDict): lots to do (disk space, memory usage, parallelizing, concurrency, locking, transactions, many others)

## Summary

- Worse is faster and in some ways better
- UserDict and DictMixin are fascinating
- Document oriented databases are fascinating

## 1.26.12 History of SQLAlchemy

- 0.1 - 2006, very small API, threadlocal, ActiveMapper added to handle objects nicely
- 0.2 - Lots of users, changed the API to be closer to today, inspired by Hibernate, Threadlocal turned off, wrote very fast
- 0.3 - Rewrite a lot internals, ORMs reorganized, users needs evaluated, API stabilized
- 0.4 - Here come developers to help out!, Lots of internal refactoring, speed profiling done, SQL expressions constructs done, lots of transactions and support for more databases.

## Highlights of 0.4

- Much faster than 0.3
- Underlying code is simplified
- lots less callcounts and redundant method calls
- Smart operators in SQL calls to handle different database types

## SQL Expression language

- `s - select([employees,c.id, employees,c.name])`
- `select([func.now()])` will work with SQLite!

## New ORM Query ORM

- Query object is fully generative. Mapped properties now have relational operators

## Inline Aliasing

- Alternate table ids to handle self-referencing tables in order to map out heirarchical data
- All the work can be filtered against a single query expression
- Easy dialogue making it very easy to implement
- In 0.4, `join()` generates aliased joins and aliases criterion for you

## High level operators

- Has and Any are synonyms for SQL exists() method.

## New ORM Configurations

- In SQL 0.4.4 familiar table and mapper constructs can be moved into class declarations called ‘New Declarative Layer’.

## Collections API

- Sets, dictionaries, and any user-defined collection may be mapped using an open API
- This means any python collection object can be used by SQLAlchemy
- Use declarations to turn any object of your choice into SQLAlchemy objects. NICE!

## Dynamic Relations

- Very large collections can be managed by a ‘dynamic’ relation, which issues queries for every access
- This way you don’t have to load a gigantic item into memory, just intelligent bits. A few restrictions, but it speeds things up

## Polymorphic Inheritance

- Whole inheritance hierachies can be loaded automatically
- So mapping out standard fields in a table (Title, Description, Create, etc) is easy. This was a gotcha in that Django effort, remember?
- Can use to specify criteria against a subclass.

## New transactional features

- Session can be configured to be ‘always transactional’, and optionally auto-flushing
- Databases which supprt SAVEPOINT can nest transactions
- Connections and ORM sessions both support two-phase commit semantics for supported databases

## Other features

- ORM supports mutable primary keys, ON UPDATE cascade.
- Arbitrary SQL expressions can be assigned to object attributes for ‘atomic’ update behavior
- metadata.reflect() can load full schemas at once
- New dialacts: MaxDB, Sybase, Access, Informix, DB2
- Horizontal sharding: extension for ORM, transparently loads and saves rows across multiple databases

## What is coming back

- Migrate is back!
- Dialects will soon decouple SQL compilers from DBAPI behaviors, allowing reuse among JDBC, ODBC, multiple native DBAPI connectors
- Jython support
- Customize class instrumentation; PJE using it to integrate with Trellis
- SQLAlchemy books

## 1.26.13 A command line's tools dream come true

- Optparse: option handling
- Subprocess: elegant

## Ideas for command line tools

- Pure Python - no systems calls but do Unix stuff
- Unix Mashups: mix python + system to solve new problems
- Network tools
- Wrapping over existing tools

## The basics: Subprocess + Optparse

- Use subprocess.call
- If stout is needed, use subprocess.Popen

## Some ideas

- Mix threading

## 1.26.14 Stackless 101

---

**Note:** Pydanny 2011/08/23 = Speaker was replaced by an alternate who wasn't a coder. This was the second worst talk I've ever seen.

- Stackless is cpython with a few modifications
  - Stackless adds tasklets, channels, cooperative multitasking
  - Stackless is about lightweight threads
- 

## Channels

- Akin to unix pipes



### 1.26.15 Why python sucks

- Not enough developers know Python
- Few organizations have Python solutions
- Python language weirdness
- Python is slow

#### Not enough Python developers

- Python developers found are of higher quality
- All levels of Python experience provided real insight
- It can be good to hire people interested in new languages
- They like to hire people who are open minded

#### Few organizations have Python solutions

- Competitive advantage due to language
- Competitive barrier to entry - people are afraid of the language
- Regular opportunity for advocacy
- Likes client that trust them to deliver
- Best customers don't care about fishing rods, just about the fish caught

#### Python language weirdness

- Most of this is personal preference.
- Forced indenting forces readability
- Dynamic typing demands more robust testing

#### Python is slow

- Some problems do not fit a pure Python solution
- Forces you early on to pick the right tool
- No tool meets every need, but knowing when it doesn't work for your task is the an important skill

#### Summary

- Python rocks
- Many weaknesses of Python have often turned into great opportunities

## 1.27 Plone Conference 2007

### 1.27.1 Ajax with Plone 3 : KSS development Patterns

by Godefroid Chappelle - Bubblenet

---

**Note:** KSS is a Python/JavaScript library that runs in Plone that uses CSS-style dialogues instead of JavaScript ones. It is something I hope the Plone community has abandoned. Stick with JQuery instead. (Danny 07/05/2011)

---

#### Overview

- Goals of KSS
- Design
- Development patterns
- test patterns

#### Goals

- Business logic should be computed on the server
  - Javascript implementations are not standardized
  - Transaction differences because you are creating a fat client
- Good integration with current development process
- Ensure we keep accessibility
- As few JS as possible
- Business logic should be computer on the server!

#### Design

- Kinetic stylesheets
- event binding
- Generic client-side engine
  - same as HTML
  - HTML snippets manipulation
- Simple server-side API
  - dom on the server
  - commands
- Plugins
  - avoid dependency on JS libraries

## Development Patterns

Do it in HTML only first. That way you have complete accessibility and its cross browser

- Client-side
  - bind events
  - css selectors
  - get Data from HTML
  - Value providers
- Server-side
  - KSS views
  - z3 browser views
  - inherit from `kss.core.KSSView`
- Command Sets
  - Queried by name (core, zope, plone)
  - Z3 subscribers and components
- FireKiss
  - Demo
  - Debug mode is possible
  - [kssproject.org/downloads/firekiss.xpi](http://kssproject.org/downloads/firekiss.xpi)

## Test Patterns

- Don't use Selenium unless you create JS plugin
- Check commands in KSS response
- `Kss.core.KSSViewTestCaseMixin`
- Check HTML elements in manipulated page
  - Selected for events
  - Selected for targets

### 1.27.2 Anti-patterns and patterns for successful projects

**Speaker:** Kamon AYEVA

---

**Note:** I had hoped to get new things out of this but as an experienced US government software contractor, none of this was new. (Danny 07/05/2011)

---

## Introduction (Why should we care)?

- Any website project is a ‘software’ project
- All projects are CMS projects
  - Users always want to update the site
  - Types of People involved in development
    - \* User
      - Works to get tool to address my needs
      - Tool should be easy to use
      - critical: does not want to change their habits
    - \* Project Manager
      - Speaks with user
      - speaks with developer
      - then manages the budget
    - \* Developer
      - Tries to get the right stuff involved: CMS, DMS, Web 2.0, Flex, AJAX, JavaFX, Python, Ruby, .NET, etc...
      - Trying hard to get best CMS done right!
  - \* Success depends on the three parties working together
  - \* Challenge is that each party has their own priorities
  - \* Tip one: Try to understand each other
  - \* User should be put first, since funding and interest comes from them

## Follow these rules so life is easier

- Avoid things that don’t work (anti-patterns)
  - Don’t reinvent the wheel and fight people who do this sort of thing
    - \* File servers, DMS, Mail Server, Calendars, security, LDAP
  - If you build it they will come
    - \* What problem are you trying to address?
    - \* Have others done it already?
    - \* Is there a third party tool that already exists for this task?
    - \* Am I building the solution for me, my customer, or my ego?
    - \* Trap: Gas Factory
      - Bloatware
      - We tend to add feature after feature until the application becomes unmanageable
- Apply rules and patterns that work

### 1.27.3 Extending and customizing Plone 3

---

**Note:** I don't remember this talk at all, but I see from my notes instructions for both buildout and generic setup, two things that caused me a lot of grief during my Zope/Plone days. In retrospect, these are tools that employ *configuration over convention*, which probably explains my dislike. (Danny 07/05/2011)

---

by **Martin Aspelli**

Plone core developer and author.

#### Agenda

- Love the buildout
- Embrace the egg
- Policy: Repeatable
- Dependencies
- GenericSetup
- Tests
- Visual customization
- Through the web development sucks

#### Interesting stuff

- Plone 3 is built off smaller packages so easier to test and reuse
- They want to nix through-the-web development cause it SUCKS
- Tell SVN to ignore Eggs

#### Glossary terms

- Policy Product
- A pattern whereby a project has a single product which upon installation, installs all dependencies and customizations in one step.

#### Buildout

Sample:

```
$ paster create -t plone3_buildout test
$ python bootstrap.py
```

kewl stuff - start building Plone 3 this way instead of manually <http://plone.org/documentation/tutorial/buildout>

## Policy Product

- (((Read the tutorial on buildout!)))
- (((Read onout setuptools (easy setup) handles versions with dependencies)))
- Create a new egg:

```
$ cd src
$ paster create -t plone example.policy
```

- Namespace is example, package is policy, Zope 2 product is True, Zip-safe is False
- Tell buildout about products
  - Declare it as a development egg so buildout doesn't go to cheese shop
  - edit things so its recognized to Zope 2 and zcml
  - Edit buildout.cfg:

```
eggs =
    elementtree
    example.policy
develop =
    src/example.policy
[instance]
zcml =
    ???
```

- rerun buildout:

```
$/bin/buildout -o (or do -N to not update the egg)
```

- Egg now ready
- zcml stuff
  - install dependencies:
    - \* `<include package="plone.browserlayer" />`

## Generic Setup

- XML syntax - bleah
- Extension profile bolts onto a base profile to amend or change configuration.
  - This is the most useful kind for third party developers
- edit `configure.zcml`
- include GS namespace
- add in generic setup tag
- managed via `portal_setup` but `portal_quickinstaller` knows how to install them too:

```
$ mkdir -p example.policy/example/policy/profiles/default
```

## Installation tests

- Need to get boilerplate off presentation documentation
- This sets up a Plone site for testing with our example
- create test\_setup.py which tests that we got basic setup right

## Visual Customization

- Zope 3 resources are customized with the ‘layer’ ZCML
- Visual components involved

### 1.27.4 Introduction to KSS, Kinetic Style Sheets

---

**Note:** Before JQuery let me walk the dom effortlessly, I had grown annoyed with JavaScript. I didn’t know then it was the DOM, and not the language. KSS seemed like a solution - *because CSS syntax against DOM agony is better than JavaScript syntax against DOM agony?* In any case, Plone later introduced JQuery and I hope KSS is out. (Danny 07/05/2011)

---

#### by Balazs Ree

<http://kssproject.org/docs/tutorials/simple-kss> [http://codespeak.net/svn/kukit/docs/introducing\\_kss/trunk](http://codespeak.net/svn/kukit/docs/introducing_kss/trunk)

get FireKSS for firefox. Then do stuff like:

```
h2: click {
    action-click: alert;
}

h2: click {
    action-click: setStyle;
    setStyle-node: backgroundColor;
}
```

### 1.27.5 Lightning Talks Thursday

---

**Note:** I rarely take this many notes on Lightning Talks these days. (Danny 07/05/11)

---

#### Plone HRM

---

**Note:** Kind of neat and the sort of thing a Plone CMS might handle well. You need tons of docs plus some core tools. Add in *PloneFormGen* and you’ve got a pretty nice HR tool. (Danny 07/05/2011)

- Human Resources Product
- Handles different locations

- Functionality
    - Salary
    - Contract Builder
    - Performance Reviews
- 

## Quills

---

**Note:** This is a tool that lets you turn your Plone site into a blog farm. (Danny 07/05/2011)

- Blogging for Plone
  - Not ready but BOF is happening
- 

## GoReplace

- SmartFolders + Regular Expressions
- Looks good

## appy.pod

-Turns Plone items into PDF, Docs, RTF, and other things

## Interop-Kmap

- Semantic indexing and semantic search of documents/contents
- Hard to understand

## Grok

---

**Note:** This is about the *simple* Zope powered framework. The work Chris McDonough did on repoze.BFG / Pyramid that eclipsed it clearly eclipsed it in both design strategy and leveraging lessons learned from outside the community. (Danny 07/05/2011)

- Luciano Ramalho
  - Simple is good
  - Start playing with it now
- 

## Manage your releases with Bundleman

---

**Note:** Even back then this seemed like a duplication of where the rest of the community was doing with buildout. However, the notes I have from this period claim ‘Good Documentation’, something that can be argued was a failure of the buildout community.

---



- Handy management tool for releases
  - Ties right into SVN
  - Used to manage externals. *I think I like builtout better.*
  - Good for maintaining older applications perhaps
  - Good documentation
  - <http://public.dev.nuxeo.com/~ben/bundleman/>
- 

### Repoze: Getting Plone to WSGI

- Paul Everitt
- WSGI, Eggs, Paste, virtualenv
- Advice: Record your coding, don't do it live

### Storage for Archetypes with SQLAlchemy

- Godefried Chapelle
- Good logging

### Entransit Content Deployment, bridging the presentation gap

- Alan Runyan
- Gives human users the ability to design and light data architecture
- Empowers end users
- Can control page layouts

## 1.27.6 5 Plone theme tips

---

**Note:** Lost the name of the author but this has the distinction of being the first Plone (or Python) conference talk I ever got to see. (Danny 07/05/2011)

---

### Tip 5 - Centered Design

Fixed width:

```
#visual-portal-wrapper {  
width: 980px;  
margin-left:auto;  
margin-right:auto;  
}  
  
- Liquid Design  
#visual-portal-wrapper {
```

```
width: 980px;
margin-left:20%;
margin-right:20%;
}
```

### Tip 4 - integration of IE

- put all stuff in IEFixes.css
- Write styles for IE7 first
- Then hack your styles for other IE version
- portal top is top part of Plone
- For IE6 and lower hack:

```
// Below is IE7 , Firefox, and Safari
#portal-top {
    background: blue;
}
// below is IE6 version
html #portal-top {
    background:red;
}
```

- easy to use
- almost no chance of breakage

### Tip 3 - Styles alterations

- different sections styling
- site root
- news: blue
- products:orange
- events: yellow;
- about:green

how:

```
body.section-news {
    background-color: blue;
}
body.section-products {
    background-color: orange;
}

// This is auto-written and any_custom_view could be working_group.pt
body.template-any_custom_view {
    background-color: spotted duck;
}
// example
body.template-working_group_view.pt {
```

```
background-color: spotted duck;  
}
```

### Tip 2 - Drop down menus

- Suckerfish for Plone 2.5.x
  - Accessible
  - valid CSS
  - Obvious and clean XHTML
- Plone Dropdowns is for Plone 3.x
  - webcouturier.dropdownmenu
  - Build it out in folder items and use a heirarchy
  - Autobuilds via Plone 3 into portal tabs
  - Uses INavtreeStrategy
  - Uses SitemapQueryBuilder()
    - \* Can change the depth of the navtree via sitemap properties

### Tip 1 - Rounded Corners

- Cornerstone of designer's minds
- pure CSS solution
- Initial nifty corners
- Too ugly XHTML
- No hooks in Plone
- People don't like dealing with CSS if they have to change images
- Images based solutions
- Sliding doors - often used for rounding corners
- Adam Kalsey technique
  - Plone has XHTML hooks in portlets for this
  - pretty simple css
  - Most of the cases use nested HTML elements
  - Fixed set of images for the corners
- JS + CSS solution
- The most flexible
- Doesn't require nested elements in HTML
- Does not require additional CSS
- Potential Solutions
  - Nifty Corners Cube (Javascript Library)

- \* First doesn't work with borders and background images
- JQuery corners
  - \* Requires jquery and does not work with Safari
- CurvyCorners library (recommended)
  - \* Supports most modern browsers
  - \* Works with borders
  - \* Works with background images
  - \* Supports antialiased corners
  - \* Cons:
    - Some problems when background images are used and box has different colors
    - Does not work well when used with multiple boxes
- collective.roundedcorners
  - \* On presenter's laptop
  - \* Normal Plone Package/Product
  - \* Uses a mix of Javascript + CSS
  - \* Raw, and will be released hopefully soon

### 1.27.7 So you want to be a Plone Consultant

by Nate Aune et al

---

**Note:** This was exciting back when I hoped to consult on the side for Plone. When I finally got my chance I hated it. Still, I respect and admire Nate, and can't say enough good things about him (Danny 07/05/2011)

---

#### Agenda

- getting started
- growing your business
- practice and process
- being a good citizen

#### Getting Started

- How do you find projects
- Work for an existing Plone firm
- Don't sell Plone, sell service
- How do you sell Plone?
- How do you charge?

## 1.27.8 Subtyping Pattern!

---

**Note:** In a recent design discussion with Nate Aune he said he thinks some of my common design patterns are really similiar to this approach. After striping away the Plone-isms, I have to say there is merit to what Nate thinks. (Danny 07/05/2011)

---

by Rocky Burt

### What is subtyping?

-Allowing for the subtyping of an existing content type

### Why subtyping?

-Many possible faces for existing content type -simple conversion -delayed specification -not exclusive to p4a.subtyper, just built off of it

use cases =====

-Need diferent content identify depending on situation -ogg file is video or audio -transforming of identity based on event reaction -How do you handle when a file type is uploaded?

### p4a.subtyper

-minimal framework -Hooks up subtypes into content menu -\$easy\_install p4a.subtyper -use workingenv or virtualenv or buildit as preferred -special subtype events

- add ISubtypAdded
- addISubtypeRemoved

-extension by adapters and schema's

### demo

-new modules interfaces and IUlradoc

- IUltadoc os marker interface
- interface.alsoProvides(needs marker interface, IContentType)
- create descriptors module
- new descriptor class called UltraDocDescriptor
  - for\_portal\_type = when this content type is displayed, this descriptor will be provided
- create ultraddoc.pt
- register descriptor as utility
- hook up new view applying to utility
- lets you have different views all with files named differently

### cases in Plone4Artists

- Look at Audio, Video, Calendar
  - no mention of p4a.subtyper
  - But uses these techniques

### 1.27.9 Untested code is broken code

---

**Note:** Some of this stuff is awesome and I use it every day. The doctest stuff led me astray and I'm glad to be free. (Danny 07/05/2011)

---

by Martin Aspeli and Philp von Weiterhausen

### arguments against

- they take time to write
- I'm a good developer
- my customer/community does the testing

### justification

- You rarely catch subtle problems
- put the time you spend writing silly bugs into writing tests
- you end up saving time when you refactor

### tests in python

- Easy, especially in Python

### Test driven development

- Write the test first
  - You don't forget to write the tests
  - You can catch design mistakes early on

### Executable documentation

---

**Note:** This section was designed to support *doctests*. Now I advocate unittests, sphinx, and rtfld.org (Danny 07/05/2011)

- Tests should exercise APIs, demonstrate how to use them
- Developers may find documentation in tests
- Why not turn them into documentation?

## doctests

**Warning:** Use doctests and you'll hate yourself later, I'm keeping this for the sake of history (Danny 07/05/2011)

- looks like an interpreter session
- restructured text - can be rendered to HTML, PDF, etc

## documentation-driven development

- write unittests first
- science fixture
- tell a story to an imaginary user
- use 'we' and 'you'
- put the story on the product home page!!!
- Danny likes this idea tons!

## General Test concepts

- Each test should be totally independent of other tests
- Only load the bits you need, don't load all components! So...
  - Use `zope.component.provideAdapter(UpperCaser)`
  - `zope.testing.doctest.DocFileSuite(..., ..., teardown=True)`
  - \$instance test
- Don't use: `from Testing import ZopeTestCase` unless you are doing integration
- *from Products.Five.testbrowser import Browser*
  - Simulates browser actions
  - learned in the class

### 1.27.10 Zope 3 for Plone Class Notes

\_Zope 3 Training for Plone Developers: <https://dev.serverzen.com/svn/public/documents/serverzen/training/zope3-for-plone-developers/docs/z3pd-training-preparation-notes.html>

- Primary focus is component architecture (CA).
- Secondary focus is an application server.
- Adds frameworks and apis built on the CA.
- Five has been incrementally adding Z3 functionality for Z2.
- Zope 3 DOES NOT deprecate archetypes
- Complementary

- Some area's overlap (schema's, form generation)

### **Exercise: Creating a functional test**

- Create a new Zope 2 product called Thoughtfulblog
- Reed discovered that PloneTestCase only works in the Instance/Products directory, but you can semlink to it .

### **Gotcha**

Can't mix and match Zope2.zpt and Zope3.zpt objects in Plone 2.5, but can do it in Plone 3.0. So METAL is bad there. This is why we should move to Plone 3.0.

### **Interfaces**

- Interfaces define what methods are there but doesn't define the logic inside. It is an agreement of how to build a class and nothing else. So any logic can be tacked on later.
- zope.interface is a good solid way to do Python interfaces. It can be used outside of Zope as seen in Twisted and other non-Zope projects.
- Marker interfaces are interface classes that lack any attributes (methods, properties, etc). A regular interface includes attributes.

### **Basic UI: View Components**

- A view component consists of a page template, python class, and some ZCML glue
- Conventions places view components in a browser subpackage.
- Views can be traversed to
- Classes always take precedence
- Five mapped a bunch of Zope 2 naming permissions to Zope 3. - Zope2.view is mapped to the standard Zope2 view system. - zcml: allowed\_attributes is a list of properties that can be vied with same permissions

### **Writing your own permissions**

- Use Products/Five/permissions.zcml as example
- id is name of Zope2 permission
- title must match string in CMF Permissions
- Rocky admitted it was a crude system

### **Integrating Archetypes**

Archetypes based classes are very good



## Setting up the view

- Views can be treated like regular CMF skin items
- Views can be registred as content actions

## Python scripts inside of templates

- Just say no
- Only return items that the pages really need.

Example inside a browser.py file:

```
return [{ 'title': x.Title(), 'link': x.absolute_url() } for x in self.context.posts() ]
```

## Testing

- Look at ThoughtfulBlog/tests.py
- And the relevant browser.txt and content.txt file
- Awesome stuff

## Sub-Type Pattern Syllabus

- Setting up the interfaces
- Modifying the view
- Introducing adapters
- Adapting folders
- Adapting smart folders
- Summary and Q & A
- Used in Plone4Artists Calendar, Media, and other bits
- Three new interfaces
  - IBlog
  - IBlogCapable
  - IBlogEnhanced
- Mapped browser:page:blog.html to work for IBlogEnhanced by using the for="".interfaces.IBlogEnhanced"
- Difference between implements and provides, which is critical semantics
  - A class implements
  - An object provides

## Adapters

- A bridge from one type of object to another type
- Say “no” to monkey patches
- Say “yes” to adapters
- Subclassing and aadapting are not exclusive
- Done via ZCML (<adapter />) or ZCML plus python
- Example:
  - ICare extends IVehicle
  - ITransport could be an adapter for getting mileage info for IVehicle
- Exercise:
  - Create new atct module - This is atct.py
  - Create new class called ATCTFolderBlog
  - Implement new posts method which calls context.contentValues() and returns BlogPost instances
  - Wire up the adapter in configure.zcml

## Revisiting Adapters

- zope.app.annotation
  - Akin to property sheets
  - A way to mark unrelated metadata onto an existing object
  - Reusable method of reusing dictionary objects
- Multi adapting takes more than one object to adapt
- Sometimes it takes two object to make a bridge
- example:
  - from zope import component
  - adapted = component.getAdapter(myfolder,provides=IBlog) # -or-
  - adapted = IBLog(myfolder)# -or maybe?-
  - adapted cmoponent.getMultiAdapter((somelang, myfolder), provides=IBlog)
- Views are multi-adapters
  - Adapts the context and request
  - Most often used as callables
  - getMultiAdapter((context,request),Interface,name=u'blog.html')

## Utilities

- Global Utilities
  - Most common

- Akin to typical python module lookup
  - can be overridden
- How to lookup a utility
  - `getUtility(ISomeInterface)` #-or-
- Exercise: Creating a global utility
- Local Components
  - Defined at the site level
  - Zope 3 'site' is mostly noted by the presence of the `ISite` interface.
  - Most folderish objects can become Zope 3 `ISites`
  - A site is just a place to store the “component registry”
  - Example: A blog share might have a site for 'news blogs' with news components and another site for 'food blogs' for food specific components.
- Sites can be nested
- All components can be overridden with the closest component registry
- utilities are commonly overrideen.
- Cannot be registered via ZCML, must be done in install module
- Exercise: Make our global utility a local one
- blocking somerthing from loading in Plone 3: `zcml:condition="not installed Plone.app.portlets"`
- Tools to util
- CMF tools are being deprecated in favor of utilities
- CMF tools use `getToolByName`
- Local utilities similiar to CMF tools
- Interface + name is important, not just name
- CMF tools being deprecated in favor utilities
- Sources
  - Vocabularies (similiar type of source), similiar to `Archetypes DisplayList`
  - Vocabularies are frown upon when seperation of concerns is important
- Standard 'source' ensures the 'view' of an item is calculated at request time (good time to figure out `il8n`)
  - `ISource` requires only that the 'in' operator works
  - Iterable sources (very common) require `__iter__` and `__len__`
  - Source binders are another utility used to generate a source based on context
  - Excercise:
- **-Permissions**
  - Permissions are actually utilities providing `IPermission`
  - Permission objects have `id`, `title`, and `description` attributes
  - No longer 'just strings' in Zope 2
  - Example: Getting all the permissions via `zopectl debug`:

```
>>> p = zope.security.interfaces.IPermission
>>> p
<InterfaceClass zope.security.interfaces.IPermission>
>>> from zope import component
>>> component.getUtilitiesFor(p)
<generator object at 0x25ab9e0>
>>> [x for x in component.getUtilitiesFor(p)]
```

- Custom Events
  - Common way to get notification when 'something; happens
  - one component fires an even
  - `zope.event.notify(evt)`
  - one component 'subscribes to the event
  - Most common use of events
  - Registered callables (often functions)
  - Does it's work because an event was fired
- Object events
  - Set of events provided and fire by core Zope
  - Examples are `IObjectCreated` and `IObjectMoved`
  - Used throughout Zope 3 and should be manually fired when necessary
  - `manage_afterAdd` is not good and is replaced via events
  - Since Zope 2.9, `ObjectManager` fires events properly
  - `Object-Manager` container `manage_XXX` methods deprecated in favor of listening for `object` events

#### Handy events:

```
+ IObjectWillBeAddedEvent
+ IObjectAddedEvent
+ IObjectWillBeRemovedEvent
+ IObjectRemovedEvent
```

- Current Archetypes `base_edit` fires `object` modified events
- Plone 3 provides richer set of `object` events being fired

## Advanced UI

- Zope 3 schemas
  - Simply an interface with more detailed attribute information
  - uses fields as described by `zope.schema`
  - Provides no UI specific information
  - Fields provided for all Python primitives (`Int`, `TextLine`, `List`, etc)
  - Fields provided for higher-level types (passwords, `URI`, `DottedName`, etc)
  - Text field type is for hold a string with many lines
  - `TextLine` is for holding a string with just one line

## Forms & Widgets

- Widgets are essentially views on schema field instances
  - Widgets provide IDisplayWidget or IInputWidget
  - Widgets must be callable
  - Widgets typically return HTML
  - getMultiAdapter((field,request),IInputWidget)
  - Automatic form generation via zope.formlib and Products.Five.formlib
  - forms are browser views which extend base classes provided by Products.Five.formlib
  - Edit forms can automatically populate form with current data (similar to base\_edit)

### plone.app.form (part of Plone 3)

- makes formlib generated forms more Plone-like
- provides extra widgets
- useful with Plone 2.5 and moreso in Plone 3

## Useful components

- Python properties rock
- p4a.subtyper might be worth looking into
- CMFonFive - Can be used to design menu items in Zope 3 style that will work for CMF - Interesting Stuff:
  - <browser:menuItem /> is how you do it
- workingenv.py
- zc.buildout - [svn.plone.org/svn/ploneout/trunk](http://svn.plone.org/svn/ploneout/trunk)
- zope.app.intid - provides unique integer based id's for objects - fast lookup, uses btree's - alternative to UID lookups with the reference catalog - five.intid brings support to Zope 2 /Plone - Go find the bloody readme
- zope.cachedescriptors - cache descriptors cache their data upon first invocation - lazy properties overwrite themselves with actual non-descriptor attributes
- lovely.tag - Provides a fast tagging engine - Uses zope.app.intid to manage id mappings - can generate tag clouds, etc - <http://Plone.tv> has this as an example

## Things to look at

- Plone4Artists
  - Uses sub typing
  - Interesting stuff:
    - \* Enables both file and Blobfile
    - \* Uses Interfaces
      - IAudio

- IPossibleAudio
- IAudioEnhanced
- Plone4ArtistsAudio
- Interesting stuff:
  - Keeps Zope 3 products separate from Zope 2 stuff

### Formlib issues

- No calendar widget
- No reference widget
- No wysiwig rich text widget

### Tangent: Plone 3

- Look up Plone 3 configlets

## 1.27.11 Relations with Alex Mitchell

---

**Note:** This tool lets you establish relationships between any two object types in Zope. The problem is that it is functionally duplicating with Zope/ZODB what relational databases give for free. (Danny 07/05/2011)

---

### What is `zc.relationship`?

- A low level ZODB index for querying relationships
- Highly optimized for simple relationships across large data
- Default confi allows relations between arbitrary persistent objects
- index can be configured to index complex relationships including non-ZODB objects
- provides transitive searches
- con: hard to use in Plone

### What is `Plone.relationships`?

- A local utility built on `zc.relationship`, which is application to a wide variety of relationship models
- A relationship class that models many-to-many content relationships
- Some optional aspects of the relationship are also indexable

## What is Plone.app.relations

- higher level API
- Content object for UML
- A set of optional adapters and subscribers
- DC workflow for relationships
- “Holding” relationships
- Relationships which are copied when their source is copied.
- example Plone code:

```
src = IRelationshipSource(obj)
src.createRelationship(target=obj)
src.getTargets()
```

## Relationship Source

- IRelationshipSource
  - Create (createRelationship), supports multiple targets
  - Query (getTargets, isLinked, getRelationshipChains, getRelationships)
  - Modify (deleteRelationships)

## Relationship Targets

- IRelationshipTarget
  - Same query methods and parameters as IRelationshipSource + getSource
  - ISymmetricRelation
- Query (isLinked, getRelationships, getRelations)

## Code Samples

stuff from the presentation:

```
>>> class IFriendship(IDCWorkflowableWorkshop):
    """A friendship"""
>>> source = IRelationshipSource(obj)
>>> rel = source.createRelationships(obj, relation='friend', interfaces=(IFriendship))
>>> list(source.getRelationships(relation='friend'))
<lazy list response>
>>> list(rel.targets), list(rel.sources)
>>> list(source.getTargets())
>>> target.isLinked()
```

### What can you do with it?

- Model non-container relationships you might need
  - Social networking
  - User favorites
  - Placeless content
  - taxonomies or complex vocabularies

### What has been done with it

[dailyreel.com](http://dailyreel.com)



## 2.1 Google Apps Script Hackathon

<https://sites.google.com/site/appsscripthackathonlosangeles/>

### 2.1.1 What is Apps Script?

Google Apps Script is a JavaScript cloud scripting language that provides easy ways to automate with Google products. Started in the spreadsheets and was expanded from there.

see <http://www.google.com/script/start/>

#### Features

- Editor in the browser
- Works with spreadsheets
- Sending email
- JDBC connector
- XML parsing
- SOAP
- Make HTTP requests with URLfetch
- Outbound OAuth support
- Build custom UIs
- Run script as a service

## 2.1.2 Use Cases

### Grading made easy with Flubaroo

- Create quiz using google forms
- Automatically grades against answer key
- Can email results and answers to each student
- Provides charts to analyze the results

### Many types of mail merge

- Define a template
- Mail!

### Vacation calendar for Brown University

- Aggregates staff that are on vacation
- Displays the results in a calendar instance

## 2.1.3 Sample code snippets

```
function emailTest() {  
  // Send myself an email  
  MailApp.sendEmail("pydanny@gmail.com", "LA Hackathon test email", "Body of email");  
  
  // Get a list of my files with 'Django' in it  
  var files = DocsList.find("Django");  
  
  // Loop and log  
  for (var i in files){  
    Logger.log(i + '-----');  
    Logger.log(files[i].getId());  
  }  
  
  //var app = UiApp.createApplication();  
}
```

## 2.1.4 My createHomePage code

This is my implementation of [https://developers.google.com/apps-script/articles/sites\\_tutorial](https://developers.google.com/apps-script/articles/sites_tutorial). It uses a mix of Contacts, Calendar, and Sites to generate a page tracking IRC discussions.

Unfortunately the problem is that the tutorial seems to have fallen behind the current API. Took me a while to hack this to work.

```
// create a new site  
  
// ISSUE: had to grab existing site because SitesApp.createSite throws strange_  
↪errors
```

```

var site = SitesApp.getSiteByUrl("https://sites.google.com/site/pydanny/");

// add team members from our Gmail Contacts as collaborators, and create a profile_
↪webpage for each contact
var contacts = ContactsApp.findContactGroup("Python").getContacts();
for (var i = 0; i < contacts.length; i++) {

    // ISSUE: Did this because the first item is a title field of 'contact'
    if (i==0){
        continue;
    };

    try {
        site.addCollaborator(contacts[i].getPrimaryEmail());

        var name = contacts[i].getFullName();
        var pageName = name.replace(/\s/g, "");
        var phone = contacts[i].getWorkPhone();
        var description = contacts[i].getNotes();
    } catch(e) {};

    var welcomeMessage = name + "'s profile page<br/><br/>Phone: " + phone + "<br/>
↪<br/>" + description;
    try {
        // ISSUE: Did this because on additional runs this for people withpages it_
↪throws errors
        var webpage = site.createWebPage(name + "'s Page", pageName + "sPage",
↪welcomeMessage);
    } catch(e) {};
}

// notify club members about future matches

// TODO: Make this work by trying site.getChildByName and then site.
↪createAnnouncementsPage
try {
    var annPage = site.createAnnouncementsPage("PyCon Annoucements", "Announcements",
↪"New announcements for the PyCon thunderdome team will be posted here.");
} catch(e) {
    var annPage = site.getChildByName("Announcements");
};
var d1 = new Date("10/20/2012");
var d2 = new Date("12/30/2012");
var events = CalendarApp.openByName("Daniel Greenfeld").getEvents(d1, d2);
for (var i = 0; i < events.length; i++) {
    var message = "<p>There will be a thunderdome chat from " + events[i].
↪getStartTime() + " until " + events[i].getEndTime() + "!</p>";
    var count = i + 1;
    var notice = "Thunderdome Chat #" + count
    Logger.log(count);
    Logger.log(message);

    // ISSUE: No easy way to check if an announcement has already been created
    try {
        annPage.createAnnouncement(notice, message);
    } catch(e) {};
}
}

```



## 3.1 Women in Engineering

### 3.1.1 Talks at L.A. Girl Geek Dinner #1 – Google Venice

- URL: <http://ggdla.com/l-a-girl-geek-dinner-1-google-venice>
- Date: September 20, 2012

#### **Socially Assistive Robotics and Discoveries on the Research Path**

by Maja J Matarić

- <http://robotics.usc.edu/interaction/>
- Ph.D University of Southern California
- BS in CS at University of Kansas
  - Go Jayhawks
- Ph.D earned at MIT
- Center director of CRES, WiSE Chair in Engineering, President of the Faculty, Vice Dean for Research
- Give much credit to her students

#### **Full bio of speaker**

Dr. Maja J Matarić is a professor of Computer Science, Neuroscience, and Pediatrics at the University of Southern California, and founding director of the USC Center for Robotics and Embedded Systems. She received her MS and PhD in Computer Science and Artificial Intelligence from MIT and is fellow of the American Association for the Advancement of Science and of the IEEE, and recipient of the Presidential Awards for Excellence in Science, Mathematics & Engineering Mentoring, the Okawa Foundation, NSF Career, and the MIT TR35 Innovation Awards.

She is featured in Michal Apter's movie "Me & Isaac Newton", in The New Yorker ("Robots that Care", J. Groopman, 2009), Popular Science ("The New Face of Autism Therapy", 2010), IEEE Spectrum ("Caregiver Robots", 2010), and is one of the LA Times Magazine 2010 Visionaries. Her research into socially assistive robotics is aimed at creating caregiving machines that can provide personalized assistance in convalescence, rehabilitation, education, and eldercare. Her group is developing robot-assisted therapies for children with autism spectrum disorders, stroke and traumatic brain injury survivors, and individuals with Alzheimer's Disease. Details are found at <http://robotics.usc.edu/interaction/>.

## Robotacist

- Enabling technologies
  - Bodies (sensors, effectors... humanoids)
  - Brains (Moore's Law)
  - Affordability
- Socio-economic factors
  - Growing aging population
    - \* In 20 years there will be as many old people as young people.
    - \* Who will take care of all these old people?
  - Healthcare crisis
  - Increased concerns about safety
  - Tech-savvy youth
- People and robots can get closer than ever: HRI is finally possible and interesting

Question: What is HRI?

## Bridging the Care Gap

We don't have the people to provide care for the young, ill or aged.

Imagine a robot:

```
... that can assist a phisical confgitaive therapist/coach
... that is enjoyable to interact with
... that is easy to command and interact with
... that is unobstrusive
... that encourages socialization
... that increases human quality of life
... that can help identify early signs of disorders
... that can provide continuous support
```

They call this robot a 'shepherd / guide'.

## A new frontier of Sience and Engineering

Human centered robots has the potential to benefit both how we do science and how we develop technologies:

1. Robots as tools for scientific inquiry into human behavior and learning processes

2. Personalized robot technology helps people, improve health and performance and quality of life

## Socially assistive robotics

Robots that help through social rather than physical interaction:

- Monitoring
  - Early detection
  - decreased risk of injury
- Coaching/training
- Motivation
  - Get people to do what they need to do to improve their quality of life.
- Companionship/socialization
  - Avoid isolation and depression

---

**Note:** Robotics enhance, not replace, human care

---

## Research questions

- **Why a robot?** The role of embodiment and physical presence.
  - embodied communication
  - body language
  - presence
  - believability vs realism
  - compliance
  - uncanny valley issues! **TODO** Get a link for Uncanny Valley cause I've found this interesting for years.
- **Making friends and influencing people?** Social monitoring and steering interaction dynamics
  - Socially appropriate behavior
  - personality
  - engagement
  - influencing human behavior and habits
- **Will it last?** Long-term personalized user adaptation.
  - sustained engagement
  - improved human wellbeing
  - Adaption along w/ the user indefinitely

---

**Note:** Social time has to be realtime. Chess is not realtime. A conversation with body language has to be done in realtime or you lose engagement with the user.

---

## Autism Spectrum Disorders (ASD)

Those with ASD will live full length lives and are at least as intelligent as the rest of us. How do we make them more productive within society?

- Children with ASD interact socially with robots in ways they do not with people or computers
- Robots seem to elicit social behaviors, communication, joint attention, turn taking, initiating play, even the first social smile
- An opportunity to develop robots as tools for ASD diagnosis, intervention and therapy

---

**Note:** They look at ways that kids with ASD suddenly begin acting with robots in a way they don't interact with humans or computers. They do amazing astonishing things they don't do otherwise.

---

## Stroke Rehabilitation

- Most stroke sufferers are left with permanent deficits due to a lack of long-term supervised rehabilitation
- 40% of traumatic brain injury symptoms (TBI) are similar to stroke
- Rehabilitation requires hours of supervised daily exercise
- Continual motivation has shown to be a critical aspects of recovery; rehabilitation is depressing.

---

**Note:** Stroke sufferers after the 12 weeks of physical therapy often don't continue working the body to improve

---

---

**Note:** Stroke sufferers will stay engaged but cheat if they can!

---

**Warning:** Robots are always interpreted as male. Adding a wig and bra to a robot is not cool. Real tests have shown that trying to change the gender of a robot is counterproductive.

## Eldercare, Alzheimer's Disease, and Dementia

- We're all headed there
- Aging-in-place requires a social component to offset isolation
- Evidence supports physical fitness as effective against Alzheimer's as medication
- Research has shown that senior citizens really like robots. Any statements otherwise are not backed up by science
- Singing games with too-perfect voices (like Frank Sinatra) are counterproductive. This is a good example of Uncanny Valley.

## Role modeling: How to be a mentor

- Be transparent and don't compartmentalize



- Expect 100% effort and commitment
- Demand collaboration. *We are a team*
- Be completely honest but fair and mindful of difference. *You are not me but this is how it works*
- Be kind but not wimpy. *Life strikes again but so what?*

### Mentoring advice

- Do what you love and love what you do
- Don't take **no** for an answer.
- Be true to yourself: speak up, bit wisely (and increasingly with seniority)
- Do outreach, so you do good and get perspective
- Refuse the false choice of work vs. family
- Be excellent and demand excellence of others
- Form a network of peers; it's all about people
- be a mentor and recruit mentors lifelong

### Hitting the Road to Mars

by Nagin Cox

- Cornell University, Robotics & Psychology
- Joined the US Air Force
- Wanted to be part of JPL since she was 14
- Been involved in Galileo, Spirit, Curiosity, and Opportunity

### Full bio of speaker

**Nagin Cox** is a Systems Engineer and Manager at NASA/JPL. Nagin graduated from Cornell University with a BS in Operations Research and Industrial Engineering, as well as a BA in Psychology, and was commissioned as an officer in the US Air Force. She worked in F-16 Aircrew Training and received a masters degree in Space Operations Systems Engineering from the Air Force Institute of Technology. As a captain, she served as an Orbital Analyst at NORAD/Space Command in Cheyenne Mountain, Colorado Springs. In 1993, joined JPL and has since served as a systems engineer and manager on multiple interplanetary robotic missions including NASA/JPL's Galileo mission to Jupiter, the Mars Exploration Rover Missions and the Kepler telescope mission to search for earth-like planets around other stars. She is currently on the mission operations team for Mars Science Laboratory (MSL)- NASA's next rover to Mars that launched in Nov 2011 and successfully landed in August of 2012. Nagin has spoken to audiences around the US, in Europe, and the Middle East on the stories of the people behind the missions.

### Many NASA centers

- Kennedy
- HQ (I worked there)
- ARC

- JPL

### About JPL

- Used to be an Army base
- Doesn't do jet testing anymore
- Does robots and exploration for NASA as part of Caltech

### Four stages of planetary exploration

1. Flyby
2. Orbiting mission
3. Landing
4. Human exploration (only done on the moon)

### About Mars exploration

- First started going there in 1960s
- Because of solar orbits, we can get there once every 2 years
- Landing on Mars was first done in 1976 with Viking
- Landed on Mars again in 1997 with *Sojourner*. Didn't go far - only about 10 feet.

### Spirit & Opportunity: Same mission in 1997 applied in 2003

- <http://science.nasa.gov/missions/mars-rovers/>
- <http://science.nasa.gov/missions/mars-exploration-rover-spirit/>
- We were only 35 million miles from Mars, closest in 65,000 years!
- Designed to investigate rocks for evidence of past liquid water

---

**Note:** She remembers the last day Spirit's wheels moved on Earth

---

### Landing the rovers

- Needed a flat landing spot. Scientists wanted the grand canyon but the engineers wanted flat terrain.
- Target landing spot is in a long ellipse pattern
- You bounce many times at five stories high over 25 times.
- Waited 10 minutes for the signal to come back after landing

## Driving the rovers

- Found lots of bedrock
- Being stuck in sand happened while trekking over dunes
- Rovers kept functioning longer and longer
  - Spirit stopped in 2010
  - Opportunity is still going as of 2012!

## Curiosity

The next big step

- <http://science.nasa.gov/missions/msl/>

## Curiosity's Capabilities

- Robot field geologist
- Mobile Geochemical laboratory
- Nuclear powered
- Really big!
- Weighs more than a 2011 minicooper

## Landing target

- Landing ellipse is tiny compared to previous missions is tiny
- Not using balloons for landing
- Guided entry, spacecraft guided itself during descent

*Do not go where the path may lead. Go instead where there is no path and leave a trail.* – Ralph Waldo Emerson

## 3.2 AngularJS

San Francisco

### 3.2.1 Rebuilding DoubleClick with AngularJS

- <http://www.meetup.com/AngularJS-SF/events/76149102/>
- August 14, 2012
- At Google San Francisco
- Case study about use of AngularJS for DoubleClick

### Meetup Description

This month we'll have a very special guest speaker - Marc Jacobs, tech lead of the DoubleClick Digital Marketing Manager team in NYC. If you want to get first hand info about building a large enterprise app with AngularJS, this is the meetup for you!

This presentation will be done over video-conference (from Mountain View) and attendees are encouraged to ask questions and join the discussion over the video link. Additionally, one or two AngularJS core team members will be present on site and can answer questions locally as well.

### Old app

- DFA6 FE was C# on ASP.NET
- DFA6 BE was Java on Oracle

### New App - challenges

- Rebuild DFA using technologies that made more sense for Google.
  - Engineering skills
  - Google tech backends
- GWT or not?
  - Principal UI technology across Google advertising products
  - No resources on their team familiar with it
  - Hard to find engineers with GWT

Other tools they considered

- Backbone.js
- JavaScriptMVC
- others

Enter AngularJS

- It was declarative
- Had two-way data binding
- Used plain old JavaScript objects
- Happily integrated with JQuery
- It had a full testing story
- It had a straightforward architectural model

### Transition to prototype

- One engineer created a basic prototype in two days.
  - Used existing web services and CSS
  - 10% time spent compared to GWT
  - 10% of the size of code base

## Transition to AngularJS

- Early Strategy: Upgrade the app page-by-page
- Eventual Strategy: Rewrote the app from scratch

Their AngularJS strategy caused a lot of FUD:

- Why weren't they using GWT?
- AngularJS maintained by less than 5 people.

Google got over the FUD and now embrace AngularJS.

## Single Page Architecture

Issues:

- Common view content
- bookmarkability

## Other libraries they use

- JQuery
- Underscore.js
- Closure Compiler (<https://developers.google.com/closure/compiler/>)
- LESS
- HTML5 Boilerplate

## Third-party controls

Started with JQueryUI but now have their own:

- list
- grid
- validation framework
- notification framework

## Open sourcing their work?

- Sharing across google
- Content that it's too specific for their use cases

---

**Note:** I think that they haven't released some of their commonly used code is not good.

---

## Internationalization

- DFA is translated into 10 languages
- They have a custom wiki-like language for things to be translated
- They rewrote the `gettext` protocol.

## Caching

- Server
- HTTP
- Browser
- `$http`
- Application

## AngularJS in review

### advantages

- Leverages standard web technologies
- New engineers get up to speed quickly
- Testing is deeply embedded in the Angular way
- Velocity of development is excellent
- Passionate framework team
- Growing community of framework users

### disadvantages

- AngularJS scenarios are ignored, poorly documented, and not at all obvious to use.
- AngularJS unit testing technologies are currently in flux.
- AngularJS documentation is quite buggy and is missing some key conceptual content.

### realities

- Flexible, powerful, but also complex
- Dependency Injection is awesome, but a mind-bender for many JS engineers
- Does not attempt to solve all key problems in large-scale web application design
- Mixing client/server-side templating can be messy, yet it may be unavoidable.

---

### Closing thoughts:

I wish I came out of this talk knowing something about how AngularJS actually works.

The documentation and style of what I see in the AngularJS docs makes me think of a modernized Dojo.

---

## 3.3 SFV Developers

San Fernando Valley developers

### 3.3.1 Talks

May 1, 2012

---

**Note:** I co-organized the event and gave a talk!

---

#### spire.io stack

- By Jason Campbell, <http://twitter.com/jxson>
- One of the guys at <http://spire.io>
- Details at <http://www.spire.io/posts/our-architecture.html>

Stack:

- node.js
  - coffeescript.js
  - redis
  - Ruby (actually jruby)
  - Lots of workers and controls of them.
- 

#### Django and MongoDB: The State of Things

- By myself
  - Slides forthcoming
- 

#### Bringing down the system using Python to save lives

- By Randall Degges

#### The issue

- Syrian Uprising
- Military killing civilians

### What can I do to help?

- Shut down military communications!

BUT HOW?!?

### F@#\$ the government!

- Syrian military monitoring civilian communications then dropping bombs
- How to protect people fighting a terrible regime?
- Shut down the system!

### Using Voip to save lives

- Use Asterisk and pycall to make millions calls
  - just a few bytes to make the call
- 

### Why Pyramid?

- By Michael R
- CTO of <http://cars.com>
- Known on IRC as ‘goodwill’
- Organized the PyCon Web Summit

### Very flexible

- Any template engine
- Any ORM or ODM

### Support

- Fast, 100% test coverage, well-documented
- Open source with strong community support
- Pay for what you get

### What Pyramid doesn't do

- Make choices for you
- No assumptions about the structure of your application



## Features

- Creates a WSGI-compliant application
- Handles HTTP requests and responses
- views, requests, and more
- Awesome t-shirt
- Works with gevent for asynchronous behavior

## Example

```
from paste.httpserver import serve
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello, World!')

if __name__ == '__main__':
    config = Configurator()
    config.add_view(hello_world)
    app = config.make_wsgi_app()
    serve(app, host='127.0.0.1')
```

---

**Note:** Finish rest of sample app

---

## Boilerplate available

- Libraries
- Scaffolding (see <http://pyramid.opencomparison.org/categories/application-scaffolding/>)
- FormAlchemy can do roughly what Django admin

## Heartshark

by Daniel Stewart

- <https://github.com/heartshark>

## Plan

- A tool to help you make up for lost meetups.
- People list people they saw on craigslist.
- Parses the craigslist data to figure out where, when, and other details.

## 3.4 JS.LA

JavaScript Los Angeles

### 3.4.1 Talks

September 25, 2012

- Event: <http://js.la/>
  - <http://jobs.js.la>
- Hosted by <http://www.crosscamp.us/>
  - <https://twitter.com/crosscampusla>
- Sponsors
  - HTML5 Dev Conf
  - EdgeCast

#### Text Mining

- Speaker: Danny Tran
  - Works for Walt Disney
  - [https://twitter.com/digi\\_danny](https://twitter.com/digi_danny)
  - <https://github.com/digidanny>
  - <http://www.dannytran.me/>

---

**Note:** very good speaker!

---

#### 80% of all data is unstructured text

- Sentiment analysis is how you do mining
- Understanding huge amounts of Twitter data
- Wanted to figure out how to investigate this via node.js
  - <https://github.com/NaturalNode/natural>
- But could have used python:
  - <http://pypi.python.org/pypi/nltk/2.0.3>
  - Which would have been **many** times faster than node.js if used with **numpy**

## Types of Sentiment

- Polarity: Positive or negative
- Continuous: -10 to +10
- Categorical: Happy, sad, angry, frustrated

## Assumptions

- Bag of Words
- English

## Techniques

- Semantic Summation
- Machine learning classification

## Pre-processing

- Unrepeat (His description of long terms like *sweeeeeeet*)
- Entity Extraction (Pulling out nouns/entities from text have to not skew the results. For example: Fight against **evil** can make a positive description sound bad)
- Tokenization
  - Problem: Abbreviations
  - Problem: Emoticons
- Spell correct <http://norvig.com/spell-correct.html>
- Word stopping
- Limitization
  - Reduce word to it's root form
  - Result is a word
  - Running => Run
- Stemming
  - Reduce word to a stem
  - May not be a word
  - Carry => Cari

## N-Grams

TODO: Get definitions of this.

## Naive Bayes Classifier

TODO: Show javascript example

---

**Note:** the data coming needs to train this algorithm

---

## Getting training data

- sentiment140.com
- Use emoticons to pre-classify text

## Test!

- Against 70k+ records from sentiment140.com
- Humans only agree 70-80% of the time so don't expect perfection

## July 26, 2012

- Hosted by <http://www.citygrid.com/>
- Event: <http://js.la/>

## Backbone: The framework that isn't

---

**Note:** Presenter used a presentation tool that doesn't have a back button. Ahem.

---

- by Daniel Hengeveld
  - <http://twitter.com/thedaniel>
  - <http://www.favortree.com>

## Synopsis

Backbone.js is often discussed alongside the many other popular client-side frameworks like Ember, Knockout, et cetera. This talk will discuss how Backbone.js is less like those opinionated frameworks and more like a set of glue libraries to build your own framework, and will also cover some useful patterns you can use in your own Backbone application.

## Lots of hot client frameworks

- Angular
- Ember
- Knockout

- Spine
- Backbone

### Backbone is kind of opinionated

The fact that it leaves out even basic framework structure yet makes you make your own decisions

- but you shouldn't have to rewrite boilerplate
- and you should still have a set of drop-in components that work well together
- You should have the freedom to do what you want.

### What do frameworks provide?

- Models
- Views
- Controllers
- Presenters
- Observers
- Routers

In other words, the glue you need to make something work.

### What does backbone provide?

- Models

```
var view = new SandwichView({model: rubeen});  
// view.model attribute set to rubeen
```

- Events instead of callbacks

```
this.collection.on('add', this.addItemView, this);  
  
function(item) {  
  view.$('.js.item.container').append(  
    (new SomeItemView({model:item})).render().el  
  );  
}
```

- Ability to combine template context and user data

```
var BaseView = Backbone.View.extend({  
  // snip giant chunk of code  
});
```

- Do the anti-pattern of having a fat router

```
app = Backbone.Router.Extend({  
    // snip giant router function  
});
```

## Presenter's opinions about Backbone

- Provides Tight coupling between models and views that represent them.
- 4 classes and a default sync method and kind of sets you loose with them
- Doesn't force you to follow a 'Backbone Way'.
- If your app isn't trivial, Backbone might not be the right choice
- If you need to handle lots of DOM and JS executions, Backbone might be a right choice.

## Going Native: Practical scripting with the JavaScript Image Object

### *A Practical JavaScript talk*

- by Aaron Martin
  - <http://twitter.com/citygridmedia>
  - <http://www.citygrid.com/>
  - Big fan of Mootools (<http://en.wikipedia.org/wiki/Mootools>)

---

**Note:** This was an AWESOME talk. Great to see how digging into the fundamentals can allow you to do some really impressive things with a tiny bit of code.

---

## Synopsis

jQuery, Dojo, Mootools, Ext, Prototype, Script.aculo.us, Backbone, Modernizer...[sigh]...it's hard to keep up. Remember when there was just something called JavaScript? I do. Don't get me wrong I love all the aforementioned libraries like a father loves his children, but when it is time to get practical and create performance solutions those babies got to go. I'm going to introduce you to an old friend of mine, the JavaScript native image object. This ol'boy is perhaps one of the most powerful objects in JavaScript and he's uber simple to get to know. I'm going to show you some of his tricks and try to convince you that he belongs in your JavaScript solutions tool belt!

## What can the Image object do?

- Handle Image Data
- Server Communication (pre AJAX)
- Analytics
- Page Performance

## Basic Image Object

```
var foo = new Image();
foo.src = url;
if(foo.complete) return true
else return false
```

## Image Events

```
foo.onerror=function(){}
// 404
foo.onsuccess=function(){}
// 200
foo.onabort=function(){}
// client timeout
```

## image\_beacon.js

To see events fired by rollover, these send out business processes on rollovers. This way they can track people's usage of the site. For example, which image are people perhaps 'right-clicking' on and other things not normally detectable.

```
var quirkyURL = 'http://url.xyz'
var backupURL = 'http://url2.xyz'

var ms = 500;
var totalms = 5000;
var tries = Math.ceil(totalms/ms);
var beacon = new Image();

// create a recursive function
var check = function(){
  if (beacon.complete == false){
    if (tries--){
      // keep on checking until we run out of tries
      setTimeout(function(){check()}, ms);
    }
    else {
      window.removeEvent('unload', quickSend);
      beacon.src = backup; // never went so we go to backup location
    }
  }
  else{
    window.removeEvent('unload', quickSend);
    //ahhhhhhhh don't leave. Need. Moar. Time.....
  }
};

//failsafe
var quickSend = window.addEvent('unload', function(){
  tries = 0;
  check();
});
```

```
beacon.src=quirkyURL;
check();
```

## image\_lazyload.js

How to load things as the page is scrolled:

```
var img = new Image();

img.onload=function(){
    el.src = img.src;
};

img.onerror=function(){
    el.src = (config.fallback)? config.fallback: "about:blank";
};

img.src = (config.src) ? config.src : config.fallback;
```

## The point of it all

- Adding in Frameworks slows things down. If you have a huge amount of data going back and forward on the site, say images, then loading JavaScript frameworks will slow you down.
- If you can figure out how to do it in Native JavaScript in 4 lines of code, that's better than loading Mootools or JQuery.

## JS To Rule Them All

- by Michael Anthony
  - [https://twitter.com/\\_activetheory](https://twitter.com/_activetheory)
  - <http://activetheory.net/>

---

**Note:** This guy is a super-talented front end developer. Incredibly good JavaScript and CSS guy.

---

## synopsis

One of the more cringe-inducing terms for any JS developer is “web page”. We’ve now reached the point where any meaningful experience needs to be an application. It also needs to be thoughtfully crafted to work on phones and tablets all the while having the level of animation and polish users have come to expect from the technology they use regardless of platform.

Static pages with responsive layouts aren’t going to cut it in this new universe, so we’ll look at using only JavaScript to structure applications, create and style markup, and also be platform aware in order to present users with an interactive experience suited for each device.

This talk will be a general theory overview accompanied by plenty of examples and very brief code demonstrations. It will cover interesting points along the way such as best practices for smooth UI and animation, content management, SEO, and even how to use these exact techniques to create native mobile applications.



## Current state of affairs

- Desktop sites like <http://nfl.com> or <http://techcrunch.com>, smashing as much content as possible into big scrolling pages
- Separate mobile sites that “*feel janky*”, not smooth like mobile apps
- iOS/Android/etc mobile app

## What can we do about it?

Write apps that run everywhere!

## But...

- You may be familiar with the term, “write once run everywhere”
- But this is different. Write multiple views and route end users to the appropriate view

## How?

- Use good code structure
- Device identification tools
- Write controllers that present the correct view for whichever device the application is being used on.
- The models, and controllers (mostly) can stay consistent
- Coding flexible layouts

## Why do this?

- Above all else, it improves the user experience. Don’t send users to mobile app download pages!
- Just the idea of creating applications instead of “web pages” helps narrow the focus to engaging users with a meaningful experience.

## Try to stick to using only JS for the view

- In the past:
  - Write a ton of markup
  - Separate out pieces of HTML via Mustache et al
  - But this gets clunky on larger efforts
- Going forward:
  - Create an element, style it, and then appending it to a parent
  - Easier to debug, much faster to debug, less files open
  - Problem: Can be a bit problematic using JQuery on mobile. Maybe use JQuery Mobile?

```
var $tst $("<div class='test' />");
$ttest.size(100, 100).setBG('image.png');
$ttest.css({top: 100, left: 100});
$parent.append($ttest);
```

## Not just static pages

- Bad animation can be distracting and in the way, good animation can be useful to invoke emotion and hierarchy.
- Getting rid of “click, reload, repeat”
- Using animation to bridge the gap between content presentation

## It's like butter

“making things look pretty on mobile devices”

- Going back to detecting device capabilities
- Use CSS3 transitions applied with JS is the best performance way
- Use top/left for desktop, and translateX/translateY for mobile. Use device detection to automatically convert left/top.
- Watch out for firefox. You can easily have flickering images.
- User translate3d for iOS
- **Use canvas only where absolute necessary**
  - Use the smallest rendering area possible
  - Keep your canvas small!

## Content Management

- Techniques to use WordPress or other traditional CMS with this type of JS application.
- Serving and loading JSON data
- Using models to store and make that data accessible within the application.

**Warning:** CMS are not ideal for this sort of API thing unless your app is supposed to be the front end for a CMS.

## SEO

- Setting your CMS to accept request from Google based on <https://developers.google.com/webmasters/ajax-crawling>
- Separating marking in the application design from markup for SEO.
- Properly routing users to the relevant content with the application.

---

**Note:** in other words, be able to present the content in standard markup so Google can pick it up for Search.

---

## Mobile Applications

- Now you have Smooth animation and performance indistinguishable from a native app
- Same code base can be made to accommodate different devices with good routing controls
- Thoughts on PhoneGap vs. Titanium:
  - **Titanium > PhoneGap**
  - PhoneGap hijacks the click method and this causes issues

## Questions

- **When is your library going to be released?**
  - My previous company owns the rights. I'm doing something new and hope to release it soon.
- **Can you use Modernizr for device capability detection?**
  - Yes
- **Do you use load different size images or do you resize them in the JS?**
  - Always load different size images. Resizing is a performance sink.

## April 26, 2012

---

**Note:** Arrived late, missed the intro talk about the venue. As always, got a seat up by the front. :-)

---

## farmhouse.LA

- <http://farmhouse.la/>
  - Holds a regional mini-conference held in Hollywood once a year.
    - <http://farmhouse.la/2012>
- 

## Visualizing Sound

- By David Guttman
- Visualizing music in realtime
- Visualize anything
- <http://davidguttman.github.com/easy-ears/>
- <https://twitter.com/davidguttman>

- “I love that for every CD ever created, there is a monster that lives inside.”
- Good speaker

---

**Note:** Very good description of calculus and functional programming. :-)

---

## Fundamentals

- Accept a bunch of values
- Convert it to output

## Sample

```
ears.updateAudio(0.5);

ctx.clearRect(0, 0, canvas.width, canvas.height);

ctx.fillStyle = "rgb(80,80,80)";

var w = Math.floor(canvas.width / 3);
ctx.fillRect(0 * w, canvas.height, w, -canvas.height * ears.lows());
ctx.fillRect(1 * w, canvas.height, w, -canvas.height * ears.mids());
ctx.fillRect(2 * w, canvas.height, w, -canvas.height * ears.highs());
```

With colors:

```
ears.updateAudio(0.5);

ctx.clearRect(0, 0, canvas.width, canvas.height);

/* adding colors */
var red = Math.round(ears.lows()*255),
    green = Math.round(ears.mids()*255),
    blue = Math.round(ears.mids()*255);

ctx.fillStyle = "rgb("+red+", "+green+", "+blue+)";

var w = Math.floor(canvas.width / 3);
ctx.fillRect(0 * w, canvas.height, w, -canvas.height * ears.lows());
ctx.fillRect(1 * w, canvas.height, w, -canvas.height * ears.mids());
ctx.fillRect(2 * w, canvas.height, w, -canvas.height * ears.highs());
```

## Native performance on Mobile with JavaScript

- Michael Anthonya
- <https://twitter.com/flashtml5>
- Using a framework called Aurora, not yet open sourced
- <http://flashtml5.com/>

## Mojito

- by Gamaiel Zavala
  - at Yahoo for 6.5 years
  - github: [gzip](#)
  - Demo and slides are here: <https://github.com/gzip/mojito-app>
  - <http://developer.yahoo.com/cocktails/mojito/docs/faq/>

---

**Note:** Way too many bullets on the slides. Links are just underlined blue titles. Sigh.

---

## What is Mojito?

- Open Source MVC client/server framework for node.js
- Core component is the mojit (not quite a module, not quite a widget)

## Mojito Features

- Built on YUI 3
- Server, client, or both
- View engine is mustache
- Broadcasts easily between server and client
- routing, lazy loading, localization
- Mojito is open source

## Installation

```
$ npm i mojito -g
$ npm i supervisor -g
```

## Mojito Terms

- **action context** is the central access point to rest of mojito in a controller
- **affinity** determines where a resource will be executed. *client*, *server*, or *common*
- **binder** only execute in the client and provide access to the controller.
- **context** key/value pairs for resolving configuration

## Mojito request flow

1. Middleware
2. Routing
3. App configuration
4. Mojits
  - COntrollers
  - Addons and Autoloads
  - Mojito configuration
  - Models
  - Views
  - Binders
  - Assets (css, images, non-YUI libs)

## Middleware

built on express.js

```
module.exports = function (req, res, next){  
  // do something  
  next();  
};
```

## Routing

- stored in *routes.json* and can vary based on context
- Specify an HTTP method and a mojit action
- May contain Sinatra-style named params
- Calls in entry in specs or an “anonymous” mojit
- Can build URLs from routes using the url addon

```
[{  
  "settings": [ "master" ],  
  "conf": {  
    "verbs": [ "get" ],  
    "path": "/conf",  
    "call": "conf.index"  
  },  
  "preso": {  
    "verbs": [ "get" ],  
    "path": "/slides/:slide",  
    "regex": { "slide": "[0-9]*" },  
    "call": "preso.index"  
  },  
  "home": {  
    "verbs": [ "get" ],
```

```
    "path": "/",  
    "call": "weather.index"  
  }  
}]
```

## App configuration

- Stored in application.json
- Can vary based on context
- Configures YUI in yui

## Controllers

- May run on the server or client
- Responds to actions
- Have access to action context, addons, autoloads, models, views, and configuration

## Models

- Have access only to configuration, autoloads, and other models.
- Any additional data must be passed through the controller.

---

**Note:** I'm guessing that's for the sake of security

---

## Views

- Out of the box views are Mustache
- They are switching to Handlebars soon, which is possible because the template languages are pluggable

## Binders

- Sole access point to controllers and the rest of Mojito in the client
- Have access to autoloads, and other things. **TOO MUCH STUFF IN EACH SLIDE**

## Addons and autoloads

- *Addons* are how you share code
- Have access to *addons*, *autoloads*, *models*, *views*, and *configuration*
- *Autoloads* are normal YUI modules and will be deployed automatically if required in a binder, controller, etc
- Mojito ships with many addons

## Assets

- Assets are css and non-YUI libs and images. They have no affinity
- Does not deploy automatically
- May be included in *application*, *'json'* or *controller*.
- JSON Dialogue to call the asset is really uncomfortably nested control structures. Hmmm...

## My Impressions

- Lots of boilerplate
- looks 'enterprisy'
- Probably good for Yahoo's use cases of dozens of types of content, a hundred languages, and thousands of regions. Not so useful for the rest of us.

## 3.5 Southern California Python Users Group

A collection of talks given at SoCalPiggies.

### 3.5.1 Talks

#### December 12, 2012

- Hosted at QConnect (Get confirmed URL soon)

## OpenCNAM

by Randall Degges

- <https://twitter.com/rdegges>
- <http://rdegges.com>
- <http://opencnam.com>

## Problems he wants to solve in Caller ID

- Ancient tech dominates (1905 era)
- 15 character strings for caller ID isn't enough
- Caller ID doesn't really work on Mobile - it involves serious hacks
- Users pay around 1 cent per query
- resolving it is **hard**
  - Pricey: Telco negotiating means talking to Executives and very expensive
  - Resources: Tiny community of developers who know CNAM telephony



- Complex: Based off really weird switches that cache CNAM data without updating
- Summary
  - Hacks for mobile
  - Data quality issues
  - Pricey

```
"It's just a string"  
-- Every developer, ever
```

### How they resolved it

- Figure out how to replace it
- Provide longer Unicode string
- Make it cheap: .04 cents per query (really cheap)
- Encrypt everything
- Provide an API:

```
$ curl -H "Accept: application/json" # TODO add rest
```

### Tools

- Flask
  - Fast
  - Under 1000 lines of code so they know it's stable
  - Lots of good libraries for it
- AWS
  - Dynamic scaling
  - DynamoDB
    - \* Similar to MongoDB, at least as fast and much more stable
    - \* Really, really fast
      - **In their case** it's 2500% (25x) faster than PostgreSQL if you are in the Amazon Datacenter
      - DynamoDB is so fast in their case it's faster than using MemcachedD
    - \* Reliable (data replicated across data centers)
    - \*
- Heroku
  - Simple to get started
  - Reliable (99.99%)
- PostgreSQL

- Via Heroku can scale up as needed
- Use read-slaves to increase for performance
- Stripe
- BitPay for BitCoin

## Method - SOA

- They broke up the project into a bunch of components
  - API service (handles telephony)
  - Accounts App (user authentication)
  - Website app (public facing site)
- Benefits
  - Developers can focus on individual components
  - If one part breaks, the other parts will still function

## Flask Libraries they used

- dynamodb-mapper
  - Replicates ORM functionality
  - Pretty fast
- flask-login
  - Handles authentication
  - flask-login decorator

## Marketing

- You don't need marketers
- Use this formula
  - Google your industry
  - Find your customers
  - Put names, urls, and emails into a google docs
  - Email 3 customers per day, asking for feedback
  - Listen to feedback

## Selling: xCNAM

A better protocol for Called ID

- 100 character Unicode string
- CNAM Storage product

- Make it faster
- Make it cheaper

## The Role and Representation of Python in the Online Educational Boom

By Justin Hampton

### Background

- First distance learning system was proposed in 1892!
- In the 1970s correspondence education picked up with television lectures
- Now we have a ton of on-line courses

### The Future is in the Headlights

- Modular vs. Linear
- Personalized vs. Generic
- Distributed vs. Localized
- Accessible vs. Elite

... but we're not there yet.

### What we have today

- MOOCs: **Massive Open Online Courses** (e.g. Coursera, Udemy, Udacity, EDX, etc)
- DIY courses: (hackerspaces, Maker Fair)
- Peer-to-peer Learning Exchanges (Meetups, PyCons, et al)

MOOCs are the new kid on the block. MIT Courseware and other things have been around for a while, but haven't taken off until recently.

**Why this is important:** 1 trillion dollars in US student debt means that many students feel can't educate themselves via college.

### The MOOC Controversy

Issues that people raise against MOOC:

- Is that a real education that you're getting?!?
  - read Clay Shirky's essay on the subject.
- The encroachment of entrepreneurs
  - Is it wise to drive education through business?
  - What skills will entrepreneurs want people to learn?
- Potentially inapplicable for most forms of education

- Can you get your medical degree this way?
- Impossible to stop cheating/other forms of academic dishonesty
  - How do you **really** test the student?
  - How can the education system get validated if online?

### What does this have to do with Python?

- Everyone offers a Python programming course.
- Python pops up even in non-programming courses such as statistics
- Python is “the new Basic”

### Taking Advantage

- Getting with the programs: Working with EDX, Coursera, et al on local tutoring programs (can be part of paid portions)
- Making room for the influx
- Reinforcing via tutorials and small instructional efforts like PyCon tutorials
- Devising our own platforms: the black best system
  - College degrees take this place in traditional educational systems
  - What can we do to enforce standards across MOOCs

### Continuing Education

- Not much use for advanced Python developers - perhaps just reinforcement.
- There is a need that exists to educate towards a an individual’s unique circumstance, no matter who one is.
- Opportunity and money for experimentation is there.

### Issues

- Collegiate/University education provides a type of socialization that can be a factor in employment. This factor is often ignored by people who advocate not attending Collegiate/University. IE - lack of cultural context needed to interact at certain levels:
  - History
  - Literacy
  - Sociology
- Entrepreneurs are often shockingly uneducated and inexperienced, relying on sales rather than any other skill. Do we want them determine the future of education?
- Really interesting discussion about online education at SoCal Python.
  - Are MOOCs really the answer?
  - Do we want business driving education?

September 26, 2012

- Hosted at <http://dreamhost.com>

---

**Note:** I'm speaking at this event but can't take notes on myself. :P

---

## Introduction to OpenStack

By Jonathon LaCour

- Experienced 15 years with Python
- Dreamhost lead developer
- Teaches high level Python classes too

## Let's talk about cloud

- The term *cloud* has been ruined
- ground rules:
  - Cloud - elastic infrastructure providing compute, network, and storage via well-defined APIs
  - Public cloud - A multi-tenant, publicly available cloud where resources are metered and billed as a utility, and any entity can place workloads.
    - \* Often makes money by charging monthly or service fees
  - Private cloud - A privately owned cloud generally used by a single organization for one or more applications

## What is OpenStack?

- OpenStack is a cloud operating system or cloud management system
- Use it to build public or private clouds
  - VMs
  - Volumes
  - Multi-tenancy
  - APIS - well defined and documented

## History of OpenStack

- 2008 - NASA creates Nebula
- 2009 - Rackspace creates Swift
- 2010 - OpenStack is born!

---

**Note:** My last few months at NASA we were begging and pleading for the right to use Nebula so we didn't have to deal with standard NASA server provisioning which is both time intensive

---

## Components of OpenStack

### Nova

- OpenStack's Compute controller, originally created by NASA
- Communicates with the hypervisor to create/destroy VMs
- "Schedules" where VMs are created
- Originally handled network and storage allocation
- RESTful API

### Swift

- Object storage system created by Rackspace
- Highly available, distributed, and eventually consistent
- RESTful APIs - S3 and Swift
- Can be swapped out for Ceph, a unified distributed storage system created at Dreamhost

### Cinder

- OpenStack's block storage as a service platform
- Enables the creation of block devices via API that can be attached to VMs
- Spun out from Nova during the latest release cycle
- RESTful APIs

### Glance

- OpenStack's image service
- Provides services for discovering, registering, and retrieving VM images
- Integrates with Ceph to enable Copy-on-write (COW) for fast VM creation
- RESTful APIs

### Quantum

- OpenStack's networking-as-a-service layer
- Replaces the networking services originally in Nova
- Supports the concepts of complex virtual networks, including isolation from other tenants in the cloud

- L2 services currently supported and some L3 services currently deployed

## Keystone

- General identify framework for OpenStack

## Horizon

- Django based user interface
- Provides web based interface to all the above components

## Parts you may want to add to OpenStack

- Routing
  - <https://github.com/dreamhost/akanda>
- VPN
- load balancing
- Firewall
- Metering system
  - Way to monetize OpenStack!
  - Ceilometer is an open source version: <https://github.com/stackforge/ceilometer>
- Billing

Commercial options are available - and open source options are being authored:

## How does OpenStack work?

- Components in designed to be loosely coupled
- Several methods of communication
  - REST APIs - simple HTTP services used to communicate between components
  - Messaging - messages sent via AMQP to a message broker, generally RabbitMQ

## Getting involved

- OpenStack is implemented in Python using standard processes
  - Familiar frameworks and tools (Django, SQLAlchemy, boto, lxml, etc)
  - Github + Gerrit + Jenkins for management and CI
- OpenStack is a community
  - Mailing lists
  - Governance moved to the newly formed OpenStack foundation. Thousands of members, including 170 companies.

- Key members of OpenStack just joined the PSF for more community crossover
- Documentation
  - <http://docs.openstack.org>
- Getting started
  - DevStack - <http://devstack.org>
  - TryStack - <http://trystack.org>

## Dreamhost's OpenStack Deployment

TODO - Get the image displayed here.

## PaaS implementations on top of OpenStack

- OpenShift
- CloudFoundry

---

**Note:** Why I like OpenStack is that you have good control over your server provisioning. If a vendor doesn't play nice, you can move your stack or deploy on your own servers.

---

## July 20, 2012

- Hosted at <http://twitter.com/crosscampusla>
- Meetup <http://www.meetup.com/socalpython/events/71023052/>
- <http://www.crosscamp.us/>
- Run by <http://grapheffect.com>

## Redis and Python: It's PB and J time

- Dr Josiah Carlson, the author of the upcoming Redis in Action
- [http://twitter.com/dr\\_josiah](http://twitter.com/dr_josiah)
- <http://bit.ly/redis-in-action>

## Who am I?

- Python user of 12 years
- Former python-dev bike-shedder
- Former maintainer of Python async sockets libraries
- Author of a few small OS projects
  - `rpqueue`, `parse-crontab`, `async_http`, `timezone-utels`, `PyPE`
  - `PyPE` is his own text editor



- Worked at some cool places we've never heard of:
  - Networks in Motion
  - Ad.ly
- Worked at some cool places you have heard of:
  - Google
  - YouTube
- And cool places will hear of: ChowNow
- Very heavy user of Redis
- Author of upcoming Redis in Action

### What is Redis?

- In-memory database/data structure server
  - Limited to main memory; vm and diskstore defunct
- Persistence via snapshot or append-only file
- Support for master/slave replication (multiple slaves and slave chaining supported)
  - No master-master, don't even try. In other words, if you have two masters running on the same system, you'll have serious, major, painful issues.
  - Client-side sharding
  - Cluster is in-progress
- Supports data structures + publish.subscribe
  - Strings, Lists, Sets, Hashes, Sorted Sets (ZSETs)
- Server-side scripting with Lua in Redis 2.6

### Comparing Redis to other databases/caches

- Memcached
  - in-memory, no-persistence, counters, strings, very fast, multi-threaded
- Redis
  - in-memory, optionally persisted, data structures, very fast, server-side scripting, single-threaded
- MongoDB
  - On-disk, speed inversely related to data integrity, bson, master/slave, sharding, multi-master, server-side map-reduce, database-level locking
- Riak
  - on-disk, pluggable data stores, multi-master sharding, RESTful API, server side map-reduce
- MySQL/PostgreSQL
  - On-disk/in-memory, pluggable data stores, master/slave, sharding, stored procedures, ...

## Redis Strings

---

**Note:** Apparent other types in Redis are really just strings. Hrm.

---

- Really just scalars of a few different types
- Character strings
  - concatenate values to the end
  - get/set individual bits
  - get/set byte ranges
- Integers (platform long int)
  - increment/decrement
  - auto “casting”
- Floats (IEEE 754 FP Double)
  - increment/decrement
  - auto “casting”

## Redis Lists

- Doubly-linked list of character strings
  - push/pop from both ends
  - **blocking** pop from multiple lists
  - **blocking**

## Redis Sets

- Unique unordered sequence of character strings
- Backed by a hash table
- add, remove, check membership, pop, random pop

## Redis Hashes

- kind of like Python dict
- Key-value mapping inside a key
- get/set/delete single/multiple
- increment values by ints/floats

## Sorted Sets (ZETS)

- Like a hash, with ‘members’ and ‘scores’, which are limited to numeric values

---

**Note:** Maybe good for search generation?

---

## Publish/Subscribe

- Readers subscribe to “channels”, exact strings or patterns
- Writers publish to channels, broadcasting to all subscribers
- Messages are transient so they can be missed

## Why Redis with Python?

- Python has reasonable sane syntax/semantics
- Python allows you to easy manipulation of data and data structures
- Python has a large and growing community
- Redis has reasonable sane syntax/semantics
- Redis allows you to easy manipulation of data and data structures
- Redis has a medium-sized and growing community
- Available as remote server
  - Like a remote IPython, only for data

Doesn’t that sound like a good match?

```
from itertools import imap
import redis
def process_lines(prefix, logfile):
    conn = redis.Redis()
    for log in imap(parse_line, open(logfile, 'rb')):
        time = log.timestamp.isoformat()
        ...
        conn.zincrby(prefix + hour, )
    ...
```

---

**Note:** code examples coming too fast. :P

---

---

**Note:** TODO: Download the slides later and try to get the code examples out.

---

## Cool stuff you can build with Redis + Python

- Reddit

- Caching
- Cookies
- Analytics
- Configuration management
- Autocomplete
- Distributed Locks
- Counting semaphores
- Task queues
- Publish/Subscribe
- Messaging
- Search engines
- Ad targeting
- Twitter
- Cat rooms
- Job searching

---

**Note:** Ahem. You can do all of this in a relational database. Or an XML filestore. Or anything.

---

**May 17, 2012**

### Amazonian Fabric

- Evan Cofsky
- <https://twitter.com/tunixman>
- Hangs out in Glendale while waiting to go back to the UK.
- Just migrated a shop off of custom virtualization to EC2.
- See <http://www.memrise.com/dev/>

---

**Note:** Good talk about patterns and techniques but I'm having trouble keeping up with switching to slides and code non-stop.

---

### Basics

- Why another management framework?
  - Wanted to be up and running quickly.
  - Wasn't familiar with Chef, Puppet, or even cfengine.
  - Was very familiar with sysadmin, Python, boto, and EC2

## Starting instances

Let's start a backup database server:

```
$start_instance web web99
```

This will:

- Create new instance of the “database” type
- Give it the host name db10.memrise.com
- Install MySQL 5.5 from source
- Install our custom .cnf

## Instance Types

EC2 Instance Types and AMIs are configured using the... (lost the slide content here)

## Control Server

- “Master” server for running other servers and tracking state.
- Issue: Another system to maintain
- Very good at getting out of sync with running systems.

## Deployment in Parallel

The `fabric` command for deployments now can:

- Fund the host types that should be deployed to
- Run common global deployment code once
- Run common deployment code across all servers in parallel
- Update the load balancer

## Building Role Lists

- Create lists of each server type using `instance_type` metadata and instance state
- Add ssh username and port
- Cache results to speed repeated calls

## Scaling Web Servers

- Can't use EC2 Elastic Cloud Service because of metrics
- Will often know in advance when load spikes will occur
- Want to make adding and removing capacity very easy

## Load balancing

- Site stack deployed to all active web servers
- New active servers added to load balancer, DNS updated
- Inactive servers removed from load balancer
- When servers inactivated, also removed

## February 22, 2012 - 12 Factor App

- by Craig Kersteirns of Heroku
- <http://www.12factor.net>

## Where we've been

- Version control
  - folders
  - cvs/svn
  - git/mercurial
- Dependency Management
  - *python setup.py install*
  - *easy\_install somepackage*
- Deployment
  - rsync myproject runserver
  - scp

## Where we're going

- Modern Version Control makes it possible to:
  - 1 code base with many deploys
- Dependency Management
  - *pip install somepackage*
  - *pip freeze > requirements.txt*
- Deployment
  - Capistrano
  - Tarball
  - slug
  - Bundle everything together so rollbacks are trivial

## What is 12 Factor?

A set of practices used by Heroku that serves as a mantra as to what they are doing.

- Declarative Setup - Minimize setup time
- Clean contracts - Portability to run anywhere
- Deploy practices - Cloud and horizontal scaling
- Minimize Divergence - Continuous Deployment

## Dev/Prod Parity

- Dev = Staging = Prod
- sqlite != postgres = postgres

**Warning:** Don't use sqlite in dev or you get screwed! This happened to me!

## Config

Don't do this:

```
$ ls
settings.py
prod-settings.py
dev-settings.py
local-settings.py
```

This is fail:

```
# settings.py
from local_settings import *
```

Or this:

```
# settings/prod.py
from base import *
```

Do this instead:

```
# settings.py
MY_SETTING = os.environ.get("MY_SETTING")
```

## Concurrency

- Don't worry about languages/frameworks being able to scale.
- Better to let the server side handle it.
- Use worker processes

## Logs

We use logs thus:

```
$ tail -f access_log
```

How we percieve logs:

```
$ ls
access_log
error_log
```

But when it comes down to it, logs are really an event stream!

- Imagine if logs were aggregated into one stream
- Then you can filter out the parts you want
- Order things the way you want

## Future Thoughts

- pip should be able to specify the version of Python
- environment settings should be set as environment variables rather than specified in Python code. Makes local setup and deployment much easier. From experience this is very true.

## June 6, 2011 - Social Python meetup at Dreamhost

---

**Note:** Arrived 30 minutes late so my notes are woefully incomplete

---

## Asynchronous programming techniques in Python

---

**Note:** My normal answer is to use Celery + Redis/RabbitMQ to handle this stuff. Is that cheating? ;)

---

- by Dreamhoster Tommi Virtanen
- slides: <http://eagain.net/talks/concurrency-oh-my/index.html>

## Talk

- *Walked in the on the part about Twisted...*
- Threads vs processes are a debatable issue.
- Cost of using Python to create new processes is considered slow
- Blocking and memory access issues
- Async is a problem in Python because not all libraries support it
- Someone mentioned a library called *pystates*



- GIL
  - Multiple threads in Python are OS processes and you can get blocking on memory objects
  - On large processes on things like *dict* hash-tables you can get blocks/locks on the wrong thing

## Multiprocessing

- Called ‘giant hack’
- Looks like the threads API
- Uses a messages system to use OS processes to share data between processes/threads
- Uses pickles to share data via messages, which means anything that is deserialized executes the code
  - Which means you should watch out for code injection!

## Communicating Sequential Processes

- Title of an academic paper
- A system of sharing tasks and data

## Gevent

- Wonderful monkey patch that does the bulk of the work needed for multi-tasking.
- Does not use threads, replaces certain libraries on the fly.
- Uses co-routines, built on lib-eb
- I’ve played with it, and it is fun.
- See:

```
monkey.patch_all()
```

---

**Note:** See my notes on Gevent at [http://pydanny-event-notes.readthedocs.org/en/latest/KiwiPycon2011/python\\_dist\\_gevent\\_redis.html](http://pydanny-event-notes.readthedocs.org/en/latest/KiwiPycon2011/python_dist_gevent_redis.html)

---

## Metaclasses: Look behind the curtain

by Dreamhoster John LaCourt

---

**Note:** Great talk, with presenter is saying it’s not magic, I agree. However, IMO, 95% people use Metaclasses, they have no reason to do so. So I listen to this talk with concern because debugging bad Metaclass code is a pain.

---

**Warning:** If you use Metaclasses badly and I have to debug your code I reserve the right to complain loudly in all public venues.

## What does a class do?

- Class constructs are called instances
- What does it really mean to construct an instance?
  - A class provides an instance with it's namespace
  - Attributes of a class define the namespace of the instance
  - Example of a class:

```
class Person(object):
    greeting = 'hello'

    def greet(self, who):
        print self.greeting, who

j = Person
print j.greet('SoCal')
'hello SoCal'
```

## Example libraries

- SQLAlchemy
- FormEncode
- Django ORM

## What is a metaclass?

- A metaclass is a class of a class
- A metaclass is a class whose instances are classes
- This is called metaprogramming

## The *type* metaclass

- If the instance of a metaclass is a class, can we insubstantiate the class just using *type*

```
def greet(self, who):
    print self.greeting, who

Person = type(
    'Person',
    (object,),
    {'greet': greet, 'greeting': 'Hello'}
)

j = Person
print j.greet('SoCal')
'hello SoCal'
```

First metaclass:

```
class MyFirstMeta(type):
    def __init__(cls, name, bases, ns):
        cls.uses_my_metaclass = True

    def mystery_method(cls):
        # All methods in metaclasses are metaclasses, which is why
        # the variable is 'cls' and not 'self'
        return 'I am a myster method'

# the grungy way of building that class
MyClass = MyFirstMeta(
    'MyFirstMeta',
    (object,),
    {'greet': greet, 'greeting': 'Hello'}
)

# the easier way of building that class
class MyClass(object):
    __metaclass__ = MyFirstMeta
```

## Practical example

Enforce all the things, like in Java

```
class Field(object):

    def __init__(self, ftype):
        self.ftype = ftype

    def TODO(self): #get this method
        pass

class EnforcerMeta(type):
    def __init__(cls, name, bases, ns):

        cls._fields = {}

        for key, value in ns.items:
            if isinstance(value, Field):
                cls._fields[key] = value

class Enforcer(EnforcerMeta):
    __metaclass__ = EnforcerMeta

    def __setattr__(self, key, value):
        if key in self._fields:
            if not self._fields[key].is_valid(value):
                raise TypeError('{0} is not valid'.format(key))
            super(Enforcer, self).__setattr__(key, value)

class Person(Enforcer):
    name = Field(str)
    age = Field(int)
```

## Great! Now be @#\$%ing careful!!!!

- Because they are constructing classes on the fly, bugs in your metaclasses will often happen during import statements
- Please, please use them judiciously

## Approaching Technical challenges as a Startup

by David Litwin

- Django site for film
- <http://www.cinely.com/>

### Cinely

- Website to connect and organize the entire production community
- Allows people to connect with each other, share work, and find jobs
- Transcoding uses zencoder
- Uses amazon ec2
- Details:
  - 10K lines of Python
  - 1K of unittest
- Needs to justify the cost of everything that they do. Startups have small budgets!

### Video transcoding

- Priority feature
- Need to be able to handle high load
- No tolerance from users about failure
- Needs to be fast:
  - 1 minute of video needs to be done in 1 minute.
  - 10 minutes video in 10 minutes
- Chose zencoder rather than ffmpeg probably because they've got dedicated resources and experiences

### Search

- Thought about Haystack, Solr, Sphinx, Google
- He's tried the all and they all suit his needs

## Real-time feeds

- Tornado + MySQL triggers?!?
- Needs to get something working, doesn't have to be too fancy
- Uses Tornado with long polling
- Uses Django signals instead of triggers :o

## Slow ORM queries

- Django ORM sometimes slows things down so you have to optimize.
  - 95% of the time it's not an issue
  - 5% of the time he hits a bottleneck
- Sometimes you have to break it out into SQL with the *.extra()* method.

## Lessons Learned

- The biggest technical challenge is determining which technical tasks take priority.
- Stay focused and excited
- Took 6 months to develop:
  - Learned to program for this project!!! Wow!!!
  - Choose Python because...
    - \* Wanted an enthusiastic community that isn't crazy
    - \* Community answers questions nicely

## June 6, 2011 - Python at AOL

---

**Note:** Late because of traffic so missed part of the first talk. Some of the audience commentary was not as polite as it should be, and I'm guilty of a little of it. However, in conversing with other attendees and especially potential speakers, it was determined that as a group we need to be better at handling speakers.

---

By Jathan (Security Engineer)

- Most of AOL is Java and Perl, and there remains a lot of TCL.
- A third of AOL staff are developers.
- Moving towards open source more and more!
- But they had 7 people at PyCon 2011.
- Still have 70% of the old infrastructure run by 25K employees now maintained by 4K employees
- Data centers in:
  - Dullas, Virginia
  - Manassas, Virginia

- Mountain View, California
- Somewhere in Germany

## about.me

A custom profile & personal analytics dashboard. Serves as a landing page for all your social sites.

- 100% Python
  - TurboGears
  - Celery
  - SQLAlchemy
  - CouchDB (entire backend?)
- Presented at PyCon 2011
  - Luke Gotzling (lead engineer)
  - [http://bit.ly/aboutme\\_pycon2011](http://bit.ly/aboutme_pycon2011)

## StudioNow

- Create, manage, and syndicate online video
  - acquired by APOL in 2010
  - AOL's content farm
  - Seen eHow.com? Yeah, like that.
- Web Services
  - Django
  - Unicorn + Gevent
- Video Transcoding (ffmpeg)
  - Celery
  - Fabric
  - Boto (EC2 interface)

## Relegence

“A major part of the AOL content infrastructure”

- Topical, Real-Time News
  - Acquired by AOL in 2006
  - Subscription-based service
  - Like lingospot.com
  - Used by most of AOL web properties.
- Topics API

- All Django all the time

### **AOL Mail - Anti-Spam Operations**

- Protecting AOL users from spam
- Whitelisting the Internet
- Tons of CLI Tools!
- Django Web Services
  - Anti-Abuse API
  - Mail Reputation Service

### **AOL Networking Engineering - Networking Security**

What Jathan does for work.

- Protecting AOL from becoming Sony
- Blacklisting the Internet!
- Firewall Policy Management
  - 4 data centers
  - 850K+ policy lines
  - 37000+ host nodes
  - 5,300 host nodes
- Simian Python suite of tools:
  - Twisted
  - Django
  - SimpleParse
  - Suds
    - \* A SOAP library for Python
  - Netaddr
  - Nudge (Evite's API library)
    - \* Not selling this product so they don't need to worry about the GPL on Nudge.

### **History of Getting Python into AOL**

- Moved to Python from Perl in 2006.
- Had to deal with thousands of moving components and twisted solved a major problem.
- Entire network/security system is in Python.

## June 6, 2011 Knowing Things

By Jeff Schenk

---

**Note:** Way too much heckling and too many comments from one person who kept taking him off topic so I asked Jeff to not take so many questions. That did not go over well with the hecklers, but actually asking on the mailing list uncovered a serious problem in the user group. Now things are better.

---

### Ad.ly's story

- Kinda knowing things is easy
- Really knowing with certainty a lot of complex things is maybe harder
- Need to know things with precision is really important

### Thing Types

- Integers (how many people clicked)
- Strings
- Booleans
- Bling (money)
- Time

### Bling

Do you want to misplace money and get fired? No? Use Decimal:

```
from decimal import Decimal
moneys = Decimal('100.01')
```

Decimal oddities in rounding so use quantize:

```
>>> moneys / 2
Decimal('50.005') # Copied something wrong here..

# so use quantize
>>> (moneys / 2).quantize('001')
Decimal('50.005') # ... or maybe here cause these numbers are the same
```

### Time

- Timezones suck
- Computers like integers
- So they use hours since epoch
- all the work is done in about 10 lines of Python code.



## Storage

- SQL
  - Joins are death
  - If you join, you will die
  - intelligent index are super
  - if you're going to group by it or filter on it, you probably want it indexed.
- Pre-Aggregate
  - When you're working with a lot of data, you need to aggregate chunks as you go.
  - *My Guess*: A lot of Celery tasks!
  - Spooned into a single report table that breaks normalization

## Iterate

They use itertools a lot!

- `itertools.chain` is your friend
- `itertools.tee` is also your friend

## heapq

Algorithms makes merging of iterables really powerful:

```
import heapq
for result in heapq.merge(query1, query2):
    # merge results and know they are in order
    print(result)
```

## Caching is key!

- They need flexibility to slice and dice the data
- Once its been sliced, they want to be able to view, page, and sort the data
- Redis gives the speed of cache with the power to sort and page
- They use *redis-py* as their library

## Questions

- Test coverage?

## May 26, 2011 - Python at Walt Disney Animation Studios

WDA = Walt Disney Animation Studios

## How many ways does WDA use Python?

- Scripting in Maya.
- DLight (lighting controls)
- Scene Navigation (navigation through scenes in a script once audio/video added)
- Art version control systems.
- Scripts to facilitate moving images and data in and out of sequences
- syrup - SAP scraper. Python Interface to our SAP based timecard system.
- Python CGI that interfaces with Production management tools and generates Excel.
- munki - A client/server to distribute 3rd party packages to max
- Coda - queuing system. C++ application with Python expressions and API
- Squish - 3rd party QT GUI test automation. Lots of Python scripts there.
- Mentor - Unittest library that interfaces well with Maya.
- GEMS - Python CGI system for searching Disney Archives
- Most 3rd party tools for the visual effect community comes with a Python API. How cool is that?
- Another 20 things I couldn't keep up with!

## Specifics on how WDA use Python?

Sample apps out of many:

- Dlight
  - C++ 52%, Python 48%
  - Surfacing
  - Lighting
  - Render Picture
- DLight GUI
  - C++ 48%, Python 52%
- Scene Navigator
  - Python 100%
  - Shot management
  - Element properties

*all tools are fully scriptable in Python*

## Disney Messaging

- Defines messages independent of Language
- Generates C++ header files with Python bindings

## Regression Testing

- Approx 2,000 regression tests between products
- Tests cover both C++ and Python functions
- 99% written using unittest

## Performance

- A hot topic!
- PyQt vs Qt
- Smart use of classes far more important than Language interface
- Time-critical functions written (or rewritten) in C++

## Package interoperability

All provide a Python API!!!

- Maya (autodesk)
- Houdini (Side effects)
- Renderman (Pixar)

## Choosing Python

- Performance trade-off
- Development cost
- Support cost
- Pipeline integration

## May 26, 2011 - Python tricks Fun with Python's newer tools

by Raymond Hettinger

### You should be on Python 2.7!

- To be supported for 5-10 years
- Earlier you convert the better

## `collections.Counter`

Tool for making rapid tallies or counts

- Modeled after:
  - Multisets in C++

- Bags in smalltalk and Objective C

Very flexible, unrestricted implementation as a directory

You can put anything in it:

- Count with positive and negative numbers
- Count with decimals, floats, or fractions
- it is just a dictionary

### Simple design as a dict subclass!

With `__missing__()` that returns zero:

```
c[x] += 1 # easy to tally
```

Has `__delitem__()` to match `__missing__()`:

```
del c[x] # easy to delete
```

### Convenience methods

- `most_common(n)`
  - returns sorted list of the n highest counts
  - reduces the code to a simple one-liner
  - implemented using either `sorted()` or `heapq.nlargest()`
- `elements`
  - lists all the contents individually
  - if an element has a count of three, it is emitted three times
  - Differs from `__iter__` which returns pairs
  - Done this way because otherwise you change the **dict** API.
- Multiset use case
  - With multisets, the counts are always positive
  - Math operations with omit zero or negative counts from the result
  - Operations are: + - & |
  - The subtraction operation is said to be saturating
  - When the counts are all one, works just like regular sets

### `collections.namedtuple()`

Works just like a regular tuple and lets you assign names to each field.

- Makes the code self-documenting
- Makes the printed `__repr__` intelligible

- Let's you change tuple order without affecting client code
- No extra memory cost over tuple

One of the best single best changes you can make to existing code.

Doing an enum in python (not needed but its kind of cool):

```
Color = namedtuple('Color', 'red orange yellow')._make(range(3))
```

## Caching

Simple unbounded caches can grow without bound.

How to do it:

```
from functools import lru_cache

@lru_cache(maxsize=100)
def big_computation(*args):
    ...
```

Fibonacci example:

```
@lru_cache()
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(5))
```

## Grotesque Hacks

Fun things to do with Python

### Not so bad

The iter() function has a two argument form that takes a sentinel value:

```
for block in iter(partial(f.read, 20), ""):
    ...
```

Awesome way to stop a loop!

## Dicts

Use DefaultDict for 2-D sparse arrays:

```
d = defaultdict(dict)
d['Canada']['Quebec'] = 1
```

## Getting grotesque

Make dict sparse to speed up a dictionary:

```
d.update(dict(d)) # doubles space in hash table
```

Wrap it up in a function:

```
def sparser(d):  
    return d.update(dict(d)) # doubles space in hash table
```

Turn-off thread-switching (cheap locking):

```
ci = getcheckinterval(0)  
sys.setcheckinterval(0) # switching off  
value = max(tasklist)  
tasklist.remove(value)  
sys.setcheckinterval(ci) # switching on
```

Atomic actions (no locks required):

```
v = d.pop(key) # find and remove in one-step  
d.setdefault(key, []).append(v) # one-step init
```

Speed-up builtin access:

```
from __builtin__ import *
```

Slow constant-function:

```
>>> def make_constant_function(x):  
...     return lambda x=x: x  
>>> f = make_constant_function(3)  
>>> f()  
3
```

Fast constant functions:

```
>>> def make_constant_function(x):  
...     return itertools.repeat(x).next
```

## April 20, 2011 - Python tricks

- April 20, 2011
- <http://code.google.com/p/lingospot>

## Questions

- Shouldn't attempts to modify Immutables cause a TypeError instead of an AttributeError?
- What is this global thing? (Don't use it)

## Thoughts

- Looks like overly complex implementation of memo-ization
- Suffering from too much Not-Invented-Here

## 3.6 PyLadies

### 3.6.1 2011/10/22 Unit Testing

- by Professor Ray Toal Loyala Marymount
- October 22, 2011

#### How to unit test

- Not manually
- You wrote code that exercises your code
- Perform assertions that record
  - That you get the results you expect

# TODO - get more info

#### Test Driven Development

1. Write some tests
2. Run them
  - They'll fail cause you have no code
3. Write some code
4. Now run the tests again
5. Work until the tests pass
6. Iterate!

#### Unit testing in Python

- Uses the module *unittest* is already there
- Docs
  - Python 2.7
  - Python 3 (<http://python.readthedocs.org/en/latest/library/unittest.html>)

## First example

Interleave two lists:

```
interleave([], [])
interleave([1,2,3], [1,2,3,4,5])
interleave([1,2,3], ["hello, world",])
```

Test code:

```
# test_interleave.py
from interleave import interleave
import unittest

class TestGettingStarted(unittest.TestCase):
    def test_interleave(self):
        cases = [
            ([], [], []),
            ([1,2,3], [1,2,3,4,5]),
            ([1,2,3], ["hello, world",])
        ]
        for case in cases:
            self.assertEqual(interleave(case[0], case[1], case[2]))

# interleave.py
def interleave(a, b):
    # will fail. Fix
    return [x for x in izip_longest(a, b) for y in x if y is not None]
```

## 3.6.2 2011/10/22 Running Django Projects on Heroku

- by Randall Degges (@rdegges)
- October 22, 2011
- Heroku = python + zombies

## Hosting vs PAAS

Hosting

- cheaper
- more time, knowledge, headaches
- more flexible
- constant revisions to architecture
- Less time for coding

PAAS

- More expensive
- More time to coding
- Risk of the service



## A bunch of commands

---

**Note:** get Randall's text from him

---

## Bunch more commands

---

**Note:** get Randall's text from him

---

## Get moar features from him

### 3.6.3 2011/10/22 trash-or-treasure

- by Emma Shroder
- <http://djangolookslikefun.wordpress.com>
- <https://github.com/ecschröder/remake>
- <http://trash-or-treasure.djangozoom.net/trashure/>

This is the world of a beginner who has been growing her skills over time. Nice to see the progression!

### 3.6.4 2011/10/22 ProtoRPC

By Leo Chen, google staffer

### 3.6.5 2011/10/22 Know Your Time Complexities

- by Ryan J. O'Neil
- rzo on #pyladies

## Whole bunch of data

Remove duplicates from a large system and remove dupes

```
import random

choices = range(100000)
x = [random.choice(choices) for i in xrange(1000000)]
```

## The Bad Way

```
order = []
for i in x:
    if i not in order:
        order.append(i)
```

## The Good Way

```
order = []
seen = set()
for i in x:
    if i not in seen:
        seen.add(i)
        order.append(i)
```

## 3.7 LA Django

### 3.7.1 LA Django 2012-04-17 various talks

All lightning talks. I was going to give one but was too busy to finishing assembling my talk.

#### Talk #1 - Deploying Django with wsgi and nginx

by Sandy Mahalo

- Demonstrated against Django 1.4
- Very technical and spoke fast so didn't get a chance to write things done. :P
- Material attribution: <http://bit.ly/hf6974>

#### Talk #2 - Naming things

by Michael Avila, works at EdgeCast

#### Two things hard in computer science

- Caching invalidation, naming things, and off-by-one errors. – Phil Karlton

#### Naming things is very hard

- Programs are just theories for how computers can interact with the world.

#### Implementation patterns

Using house and street as our naming pattern target:

#### class - simple super class name

- Top of inheritance chain
- Stick to one word
- **Do this:** Street
- **Not this:** HouseList

### class - qualified sub class name

- How are they alike?
- How are they different?
- **Do this:** Hose -> GardenHose
- **Not this:** Hose -> WaterSource

### variable - role suggesting name

- What is it's type?
- What is it's scope/lifetime?
- What is it's role?
- **Do this:** adams
- **Not this:** adamsHouse

### function - intention-revealing name

- Stick to verb phrases
- **Do this:** street.add\_house(a)
- **Not this:** street.houses(a)

### Useful books for naming things

- [Random House Word Menu](#)
- [Ultimate Visual Dictionary](#)

## Talk #3 - Extracting Domain Objects

by Brian Riley, Edgecast

- Target of the talk was some of the code from the Django tutorial :-)
- Django is for serving out the web, not not necessarily other things.
- Sometimes it can be good to extract the business logic from the obvious place and put into other control systems.
  - Celery comes to mind.

```
# old way
selected_choice.vote += 1

# domain way
class Voter(object):
    DEFAULT_NUMBER_OF_VOTES = 1

    def number_of_votes(self, user):
        if user.is_authenticated():
```

```
    return user.votes
    return Voter.DEFAULT_NUMBER_OF_VOTES
```

## Talk #4 - Integrating Facebook into Django

by Dan Loewenhertz of

- Has built a number of Facebook apps
- Very interesting decorator approach to getting this done.
- <https://github.com/elmcitylabs/ECL-Facebook>

## 3.7.2 Django testing dos and don'ts

**Note:** Example code was dark background with gloomy fonts that no one could read.

mahalow 90401902

### The presentation

- Do: use AssertEquals/AssertNotEquals
  - Don't: AssertTrue is not good
- Do: Build TestCase Sub-classes
  - Don't repeat yourself
    - \* Create users
    - \* Create objects
    - \* Log users in
    - \* Clear cache
  - Don't make your reusable test methods not too complex
- Do: Test all possible user cases
  - **Analyze all possible exit points of a function**
    - \* All returns
    - \* all expected exceptions
  - Shoot for 80% code coverage
- Do: Test that data changes on success
  - Don't just assert that the function ran successfully
  - If data has changed, assert that the DB holds updated data
  - If cache was manipulated, assert that it is correct
- Do: Use controls for complex testing
  - Create a control record/object to make sure complex functions don't touch unrelated elements.
  - Example: To check user deletion, create *user\_to\_be\_deleted* and *control\_user*

- Do: Write **very** specific tests
  - in most cases, a given test function should be no more than 10-15 lines long (use your setUp function!)
- Do: Test more than one type of use-case
  - Testing all error use-cases are as useful as testing for success
  - Success use-cases can still succeed, but for varying reasons over time.
- Do: Keep your tests dirt simple. Don't write tests that require tests of their own
  - Don't: Write TestCase-specific Helper functions
- Don't: Test everything in one big TestCase
  - Use many test cases: *test\_managers*, *test\_models*, *test\_views*
- Don't do these things
  - Write tests that go beyond your codebase
  - Use fixtures for everything
  - Write unnecessary code
  - Repeat yourself

### 3.7.3 Django testing

Sandy's great talk on test organization! I do things very similarly!

#### Organize and optimize your tests

- All apps need tests
- As your apps become more complex
  - test suite will grow larger
  - Increasingly complex test scenarios arise
  - seperate **code** and **service/infrastructure**
- Don't keep all your tests in one monolithic file named *test.py*

#### Don't do this

- Don't have a 6000 line test.py file! Break it down into *test\_models*, *test\_managers*, etc.
- Don't test 3rd party APIs
  - Sandy prefers Mock and other faking of those systems
  - Danny: But how do you test your payment gateways and other API calls?
    - \* Paypal, Authorize.net
    - \* Github, Bitbucket, etc
    - \* Twilio, Tropo
    - \* Maybe have a stand-alone test suite called *third\_party\_apis*

## Do this

sample of how I do it:

```
core/
  tests.py # This is where my test utils go!
myapp/
  tests/
    test_models.py
    test_views.py
```

I prefix my test modules with *test\_* so that way it is easier to identify them in various text editors. We already have enough models and views that it confuses me!

Sandy does things a bit differently, in that she calls test modules *models* and *views*. Really though, it is a measure of what you prefer for naming conventions

## Dev environments and code testing

- Services and infrastructure should not be tested in code tests
- Code should be written to handle this sort of thing.
- Hard to get away from.
- Example: Multiple servers

## Continuous integration

- Jenkins is your friend (<http://jenkins-ci.org/>)
- Track code coverage

## Worst Practice: Not testing

There're lots of things you can do wrong in testing

- Not having them
- Writing them badly

## 3.7.4 April 19, 2011

### Caching

Concept:

```
class CacheManager(managers.Manager):
    """
        usage:
            objects = CacheManager()

    """
    ...
    DNE = "DoesNotExist"
```

```
# put caching methods here  
# put signal handling here
```

## Unittesting

FactoryBoy looks awesome!

## FB Integration

Using the Facebook python SDK?

## 3.8 LA Hack Night

### 3.8.1 May 23, 2012 LA Hack Night

- Location: <http://spire.io> HQ
- Meetup: <http://www.meetup.com/Los-Angeles-Hack-Night/events/60609452/>

**Warning:** I like the spire.io guys but I'm going to do my best to pierce the marketing hype tonight and see what they are really doing.

---

**Note:** Presentation was given on a monitor. Should have been done off of a projector.

---

## What is Serverless?

- You can just write HTML and CSS and they do the rest.
  - They handle users.
  - They handle authentication and authorization.
  - They handle messaging.
  - They handle load balancing and scaling.
  - You don't have to worry about choosing databases.
- Fast prototyping
- Scalable
- Secure (security is not an afterthought)

## Load testing

- Can handle HN and Tech Crunch swamping.
- 10K requests per second, which will dwarf anything HN or Tech Crunch can do.
- Currently they manually spin up servers. They are working on automation.

## Current APIs

- Identity
  - Create and authenticate users
- Messaging
  - Send messages to thousands of clients in real-time
- Data (coming soon)

## Spire.io is Secure

- Capability security (principle of least privilege)
- All traffic in HTTPS
- No need to put password or secret in your client side code

## Spire.io is Scalable

- Distributed architecture
- Built with Node.js, Redis, and JRuby
  - Node.js for dispatch handling
  - Redis for speed
  - JRuby for spinning off of tasks

## Simple app demo

- Code: <https://github.com/spire-io/hacknight-note-app>
- Demo: <http://iriecycle.net/hacknight/>

## Spire.io uses CORS

- Like JSONP but better
- See [http://en.wikipedia.org/wiki/Cross-origin\\_resource\\_sharing](http://en.wikipedia.org/wiki/Cross-origin_resource_sharing)



## CHAPTER 4

---

### Articles

---

- <http://pydanny.blogspot.com/2011/12/story-of-live-noting.html>



## CHAPTER 5

---

### Credits

---

- Audrey Roy. She makes this and so much more possible. She is my life.
- Eric Holscher and the <http://rtfd.org> community.
- Github for DVCS hosting.
- The Django, Python, and the Open Source communities.



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`