

---

# **First News App Documentation**

***Release***

**Investigative Reporters and Editors**

**Mar 02, 2018**



---

## Contents

---

<b>1</b>	<b>What you will make</b>	<b>3</b>
<b>2</b>	<b>About the authors</b>	<b>5</b>
<b>3</b>	<b>Prelude: Prerequisites</b>	<b>7</b>
3.1	Command-line interface . . . . .	7
3.2	Python . . . . .	8
3.3	pip and virtualenv . . . . .	9
<b>4</b>	<b>Act 1: Hello Git</b>	<b>11</b>
<b>5</b>	<b>Act 2: Hello Flask</b>	<b>13</b>
<b>6</b>	<b>Act 3: Hello HTML</b>	<b>17</b>
<b>7</b>	<b>Act 4: Hello JavaScript</b>	<b>29</b>
<b>8</b>	<b>Act 5: Hello Internet</b>	<b>39</b>
<b>9</b>	<b>Epilogue: Hello CSS</b>	<b>43</b>



A step-by-step guide to publishing a simple news application.

This tutorial will walk you through the process of building an interactive data visualization from a structured dataset. You will get hands-on experience in every stage of the development process, writing Python, HTML and JavaScript while recording it in Git's version control system. By the end you will have published your work on the World Wide Web.



# CHAPTER 1

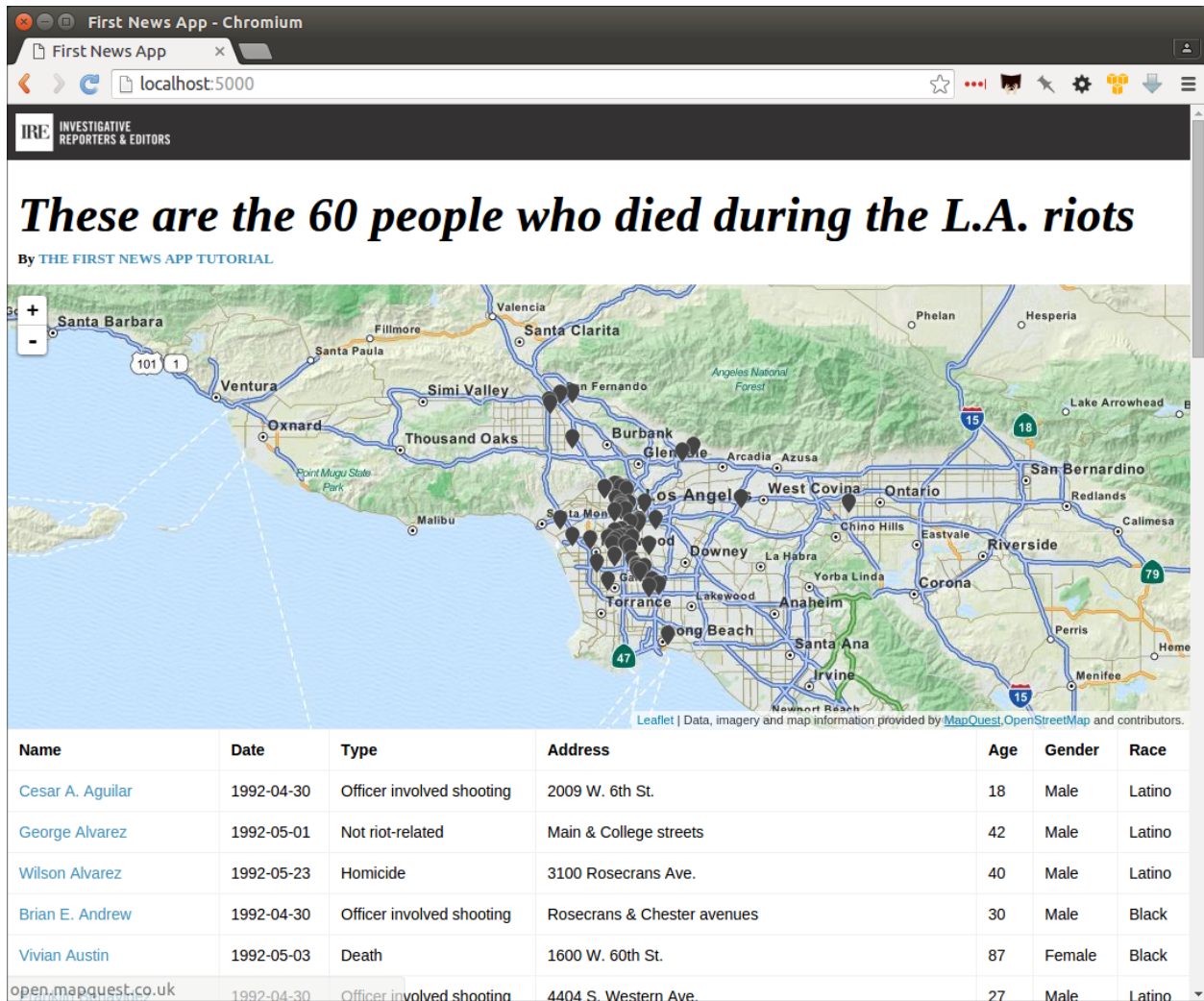
---

## What you will make

---

By the end of this lesson, you will publish an interactive database and map about the dozens of people who died during the riots that swept Los Angeles for five days in 1992. You will do this by repurposing the data from [a Los Angeles Times application](#) that accompanied a story released on the 20th anniversary of the riots.

A working example of what you'll make can be found at [ireapps.github.io/first-news-app/build/index.html](https://ireapps.github.io/first-news-app/build/index.html)



Past students of this class have gone on to use the skills they learned to create projects like The Chicago Reporter's police complaints database, the Naples Daily News' greyhound dogs death database and the San Antonio Express-News' homicide database.



## CHAPTER 2

---

### About the authors

---

This guide was prepared for training sessions of Investigative Reporters and Editors (IRE) and the National Institute for Computer-Assisted Reporting (NICAR) by Ben Welsh. It debuted in February 2014 at NICAR's conference in Baltimore. A revised version was presented at the 2015 conference in Atlanta and the 2016 conference in Denver. It was taught for the fourth time at the 2017 conference in Jacksonville by Armand Emamdjomeh and Ben Welsh.



# CHAPTER 3

---

## Prelude: Prerequisites

---

Before you can begin, your computer needs the following tools installed and working.

1. A [command-line interface](#) to interact with your computer
2. A [text editor](#) to work with plain text files
3. Version 2.7 of the [Python](#) programming language
4. The [pip](#) package manager and [virtualenv](#) environment manager for Python
5. [Git](#) version control software and an account at [GitHub.com](#)

**Warning:** Stop and make sure you have all these tools installed and working properly. Otherwise, [you're gonna have a bad time](#).

## 3.1 Command-line interface

Unless something is wrong with your computer, there should be a way to open a window that lets you type in commands. Different operating systems give this tool slightly different names, but they all have some form of it.

On Windows you can find the command-line interface by opening the “command prompt.” Here are [instructions](#).

On Apple computers, you open the “[Terminal](#)” application.

Ubuntu Linux comes with a program of the [same name](#).

### 3.1.1 Text editor

A program like Microsoft Word, which can do all sorts of text formatting like change the size and color of words, is not what you need. Do not try to use it.

You need a program that works with simple “plain text” files, and is therefore capable of editing documents containing Python code, HTML markup and other languages without dressing them up by adding anything extra. Such programs are easy to find and some of the best ones are free, including those below.

For Windows, I recommend installing [Notepad++](#).

For Apple computers, try [Sublime Text](#).

In Ubuntu Linux you can stick with the pre-installed [gedit](#) text editor.

## 3.2 Python

Python is a computer programming language, like many others you may have heard of such as Ruby or PHP or Java. It is free and open source. You will need to have version 2.7 installed.

If you are using Mac OSX, Python version 2.7 is probably already installed and you can test to see what version, if any, is already available by typing the following into your terminal.

```
$ python -V
```

---

**Note:** You’ll note that the example above begins with a “\$”. You do not need to type this. It is only a generic symbol commonly used by geeks to indicate a piece of code should be run from the command line. On Windows, this prompt could even look quite different, likely starting with a phrase like `C:\`.

---

You should see something like this after you hit enter:

```
$ python -V
Python 2.7.12
```

If you get an error instead, Mac users should install Python by following [these instructions](#) .

If your version is 2.7.11 or 2.7.13 or some other slight variation from what’s above, that’s okay.

If it’s Python 3, that’s another issue and you should try to install Python 2.7 instead. If you continue with Python 3, this class may largely work, but you could encounter some small problems you’ll need to sort out on your own.

Windows people should open their command prompt and investigate whether Python is already installed by typing the following command.

```
$ python -V
```

---

**Note:** You’ll note that the example above begins with a “\$”. You do not need to type this. It is only a generic symbol commonly used by geeks to indicate a piece of code should be run from the command line. On Windows, this prompt could even look quite different, likely starting with a phrase like `C:\`.

---

You should see something like this after you hit enter:

```
$ python -V
Python 2.7.12
```

If you get an error instead, install Python by following the installation guide [here](#) which will have you download and install Python from [here](#). After that’s done, ensure Python is installed by reopening the command prompt and running the command above again.

If your version is 2.7.11 or 2.7.13 or some other slight variation from what’s above, that’s okay.

If it's Python 3, that's another issue and you should try to install Python 2.7 instead. If you continue with Python 3, this class may largely work, but you could encounter some small problems you'll need to sort out on your own.

## 3.3 pip and virtualenv

The [pip package manager](#) makes it easy to install open-source libraries that expand what you're able to do with Python. Later, we will use it to install everything needed to create a working web application.

Verify pip is installed with the following command:

```
$ pip -V
```

If you get an error, that means you don't have pip installed. You can get it by following [these instructions](#).

The [virtualenv environment manager](#) makes it possible to create an isolated corner of your computer where all the different tools you use to build an application are sealed off.

It might not be obvious why you need to do this, but it quickly becomes important when you need to juggle different tools for different projects on one computer. By developing your applications inside separate virtualenv environments, you can use different versions of the same third-party Python libraries without a conflict. You can also more easily recreate your project on another machine, handy when you want to copy your code to a server that publishes pages on the Internet.

You can check if virtualenv is installed with the following command:

```
$ virtualenv --version
```

If you don't have virtualenv, install it with pip.

```
# On Windows:
$ pip install virtualenv
# On a Mac or Linux you will need to install it as a superuser with the sudo command.
# When you are prompted for a password, use the same one that logs you into your
↪ computer
$ sudo pip install virtualenv
```

If that doesn't work, [try following this advice](#).

### 3.3.1 Git and GitHub

[Git](#) is a version control program for saving the changes you make to files over time. This is useful when you're working on your own, but quickly becomes essential with large software projects when you work with other developers.

[GitHub](#) is a website that hosts git code repositories, both public and private. It comes with many helpful tools for reviewing code and managing projects. It also has some [extra tricks](#) that make it easy to publish web pages, which we will use later.

GitHub offers helpful guides for installing Git for [Windows](#), [Macs](#) and [Linux](#).

You can verify it's installed from your command line like so:

```
$ git --version
```

Once that's done, you should create an account at GitHub, if you don't already have one. [The free plan](#) is all that's required to complete this lesson.



## CHAPTER 4

---

### Act 1: Hello Git

---

Start by creating a new development environment with virtualenv. Name it after our application.

```
# You don't have to type the "$" It's just a generic symbol
# geeks use to show they're working on the command line.
$ virtualenv first-news-app
```

Jump into the directory it created.

```
$ cd first-news-app
```

Turn on the new virtualenv, which will instruct your terminal to only use those libraries installed inside its sealed space. You only need to create the virtualenv once, but you’ll need to repeat these “activation” steps each time you return to working on this project.

```
# In Linux or Mac OSX try this...
$ . bin/activate
# In Windows it might take something more like...
$ cd Scripts
$ . .\activate
$ cd ..
```

Create a new Git repository.

```
$ git init repo
```

Jump into the repository.

```
$ cd repo
```

Visit [GitHub](#) and create a new public repository named `first-news-app`. Don’t check “Initialize with README.” You want to start with a blank repository.

Then connect your local directory to GitHub with the following command.

```
$ git remote add origin https://github.com/<yourusername>/first-news-app.git
```

Create your first file, a blank README with a [Markdown](#) file extension since that's the preferred format of GitHub.

```
# Macs or Linux:
$ touch README.md
# In Windows fire it up in your text editor right away:
$ start notepad++ README.md
```

Open up the README in your text editor and type something in it. Maybe something like:

```
My first news app
=====
```

Make sure to save it. Then officially add the file to your repository for tracking with Git's `add` command.

```
$ git add README.md
```

Log its creation with Git's `commit` command. You can include a personalized message after the `-m` flag.

```
$ git commit -m "First commit"
```

If this is your first time using Git, you may be prompted to configure your name and email. If so, take the time now. Then run the `commit` command above again.

```
$ git config --global user.email "your@email.com"
$ git config --global user.name "your name"
```

Now, finally, push your commit up to GitHub.

```
$ git push origin master
```

Reload your repository on GitHub and see your handiwork.



# CHAPTER 5

## Act 2: Hello Flask

Use pip on the command line to install [Flask](#), the Python “microframework” we’ll use to put together our website.

```
$ pip install Flask
```

Create a new file called `app.py` where we will configure Flask.

```
# Again, Macs and Linux:
$ touch app.py
# Windows:
$ start notepad++ app.py
```

Open `app.py` with your text editor and import the Flask basics. This is the file that will serve as your application’s “backend,” routing data to the appropriate pages.

```
from flask import Flask
app = Flask(__name__) # Note the double underscores on each side!
```

Next we will configure Flask to make a page at your site’s root URL.

Configure Flask to boot up a test server when you run `app.py` like so:

```
from flask import Flask
app = Flask(__name__)

if __name__ == '__main__':
    # Fire up the Flask test server
    app.run(debug=True, use_reloader=True)
```

**Note:** You’re probably asking, “What the heck is `if __name__ == '__main__':`?” The short answer: It’s just one of the weird things in Python you have to memorize. But it’s worth the brain space because it allows you to run any Python script as a program.

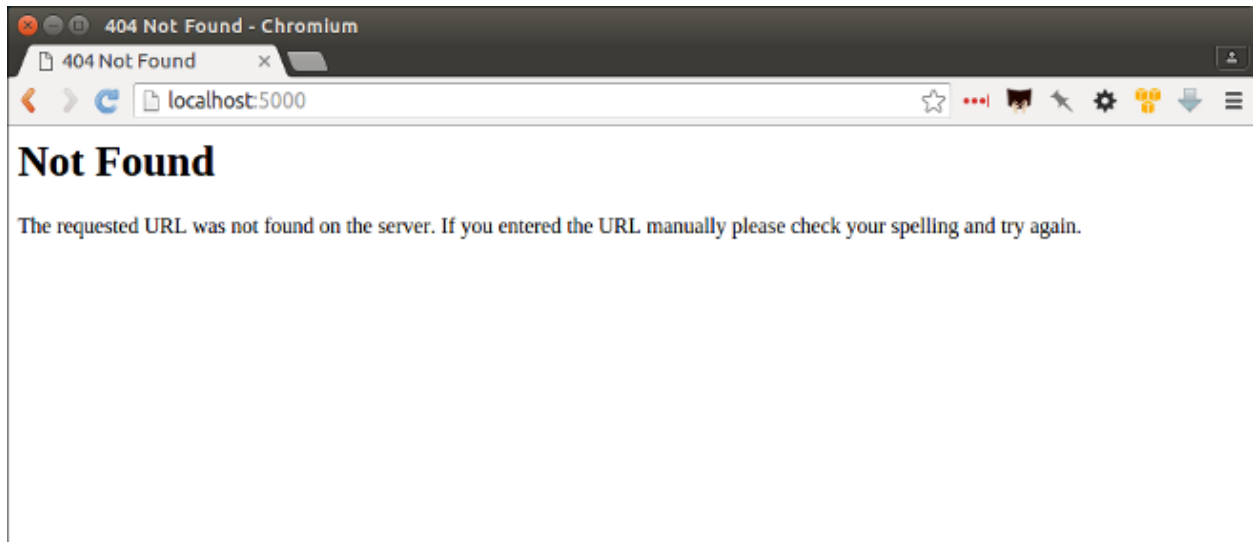
Anything indented inside that particular `if` clause is executed when the script is called from the command line. In this case, that means booting up your web site using Flask's built-in `app.run` function.

---

Don't forget to save your changes. Then run `app.py` on the command-line and open up your browser to `localhost:5000`

```
$ python app.py
```

Here's what you should see. A website with nothing to show.



Next we'll put a page there. Our goal is to publish the complete list of people who died during the riots using a template. We will call that template "index.html".

Before we do that, return to your command-line interface and stop your webserver by hitting the combination of CTRL-C. You should now again at the standard command-line interface.

Now in `app.py` import `render_template`, a Flask function we can use to combine data with HTML to make a webpage.

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

if __name__ == '__main__':
    # Fire up the Flask test server
    app.run(debug=True, use_reloader=True)
```

Then create a function called `index` that returns our rendered `index.html` template.

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
```

```
# Fire up the Flask test server
app.run(debug=True, use_reloader=True)
```

Now use one of Flask's coolest tricks, the `app.route` decorator, to connect that function with the root URL of our site, `/`.

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    # Fire up the Flask test server
    app.run(debug=True, use_reloader=True)
```

Return to your command line and create a directory to store your templates in the default location Flask expects.

```
$ mkdir templates
```

Next create the `index.html` file we referenced in `app.py`. This is the HTML file where you will lay out your webpage.

```
# Macs and Linux:
$ touch templates/index.html
# Windows:
$ start notepad++ templates/index.html
```

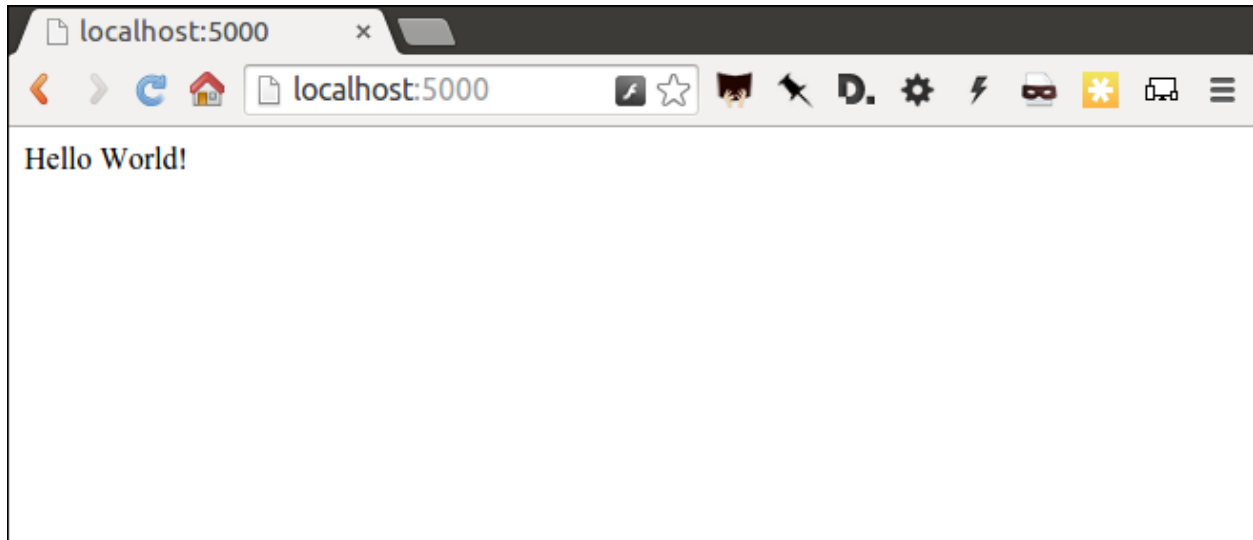
Open it up in your text editor and write something clever.

```
Hello World!
```

Now restart your Flask server.

```
$ python app.py
```

Head back to your browser and visit `localhost:5000` again. You should see the contents of your template displayed on the page.



We're approaching the end of this act, so it's time to save your work by returning to the command line and committing these changes to your Git repository.

---

**Note:** To get the terminal back up, you will either need to quit out of `app.py` by hitting CTRL-C, or open a second terminal and do additional work there. If you elect to open a second terminal, which is recommended, make sure to check into the virtualenv by repeating the `. bin/activate` part of *Act 1: Hello Git*. If you choose to quit out of `app.py`, you will need to turn it back on later by calling `python app.py` where appropriate.

As we progress through this lesson, you will need to continually do this to switch between the server and terminal. We no longer be instructing to do it each time from here on.

---

I bet you remember how from above. But here's a reminder.

```
$ git add . # Using "." is a trick that will quickly stage *all* files you've changed.  
$ git commit -m "Flask app.py and first template"
```

Push it up to GitHub and check out the changes there.

```
$ git push origin master
```

Congratulations, you've made a real web page with Flask. Now to put something useful in it.

## CHAPTER 6

---

### Act 3: Hello HTML

---

Start over in your `templates/index.html` file with a bare-bones HTML document.

```
<!doctype html>
<html lang="en">
  <head></head>
  <body>
    <h1>Deaths during the L.A. riots</h1>
  </body>
</html>
```

Commit the changes to your repository, if only for practice.

```
$ git add templates/index.html
$ git commit -m "Real HTML"
$ git push origin master
```

Make a directory to store our data file.

```
$ mkdir static
```

Download the comma-delimited file that will be the backbone of our application and save it there as `la-riots-deaths.csv`. Add it to your git repository.

```
$ git add static
$ git commit -m "Added CSV source data"
$ git push origin master
```

Next we will open up `app.py` in your text editor and create a function that uses Python's `csv` module to access the data.

First, create the new function and give it the path to your CSV file.

```
import csv
from flask import Flask
from flask import render_template
```

```
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Open up the file path for reading with Python using the built-in `open` function.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'rb')

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Pass it into the csv module's `DictReader`, to be parsed and returned as a list of dictionaries.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'rb')
    csv_obj = csv.DictReader(csv_file)

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

---

**Note:** Don't know what a dictionary is? That's okay. You can read more about them [here](#) but the minimum you need to know now is that they are Python's way of handling each row in your CSV. The columns there, like `id` or `gender`, are translated into "keys" on dictionary objects that you can access like `row['id']`.

---

A quirk of CSV objects is that once they're used they disappear. There's a good reason related to efficiency and memory limitations and all that but we won't bother with that here. Just take our word and use Python's built-in `list` function to convert this one to a permanent list.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'rb')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Close the function by returning the csv list.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'rb')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Next have your index function pull the CSV data using your new code and pass it on the top the template, where it will be named `object_list`.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'r')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
```

```

    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)

```

Make sure to save `app.py`. Then return to the `index.html` template. There you can dump out the `object_list` data using Flask's templating language [Jinja](#).

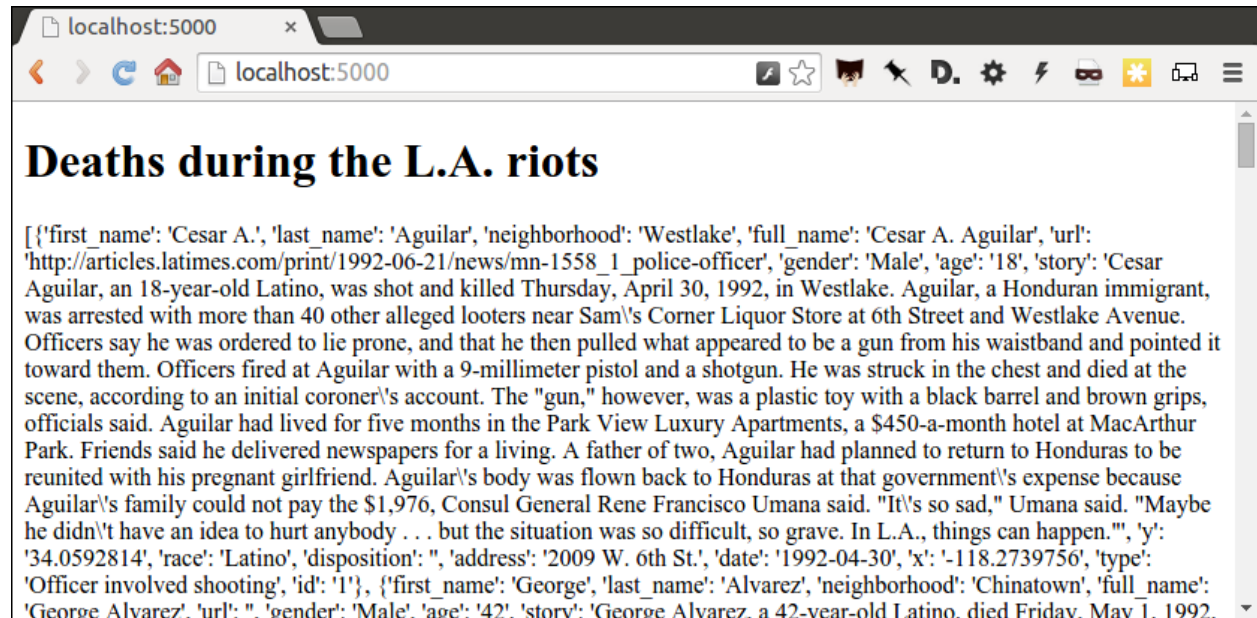
```

<!doctype html>
<html lang="en">
  <head></head>
  <body>
    <h1>Deaths during the L.A. riots</h1>
    {{ object_list }}
  </body>
</html>

```

If it isn't already running, return the command line, restart your test server and visit `localhost:5000` again.

```
$ python app.py
```



Now we'll use Jinja to sculpt the data in `index.html` to create an [HTML table](#) that lists all the names. Flask's templating language allows us to loop through the data list and print out a row for each record.

```

<!doctype html>
<html lang="en">
  <head></head>
  <body>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>

```

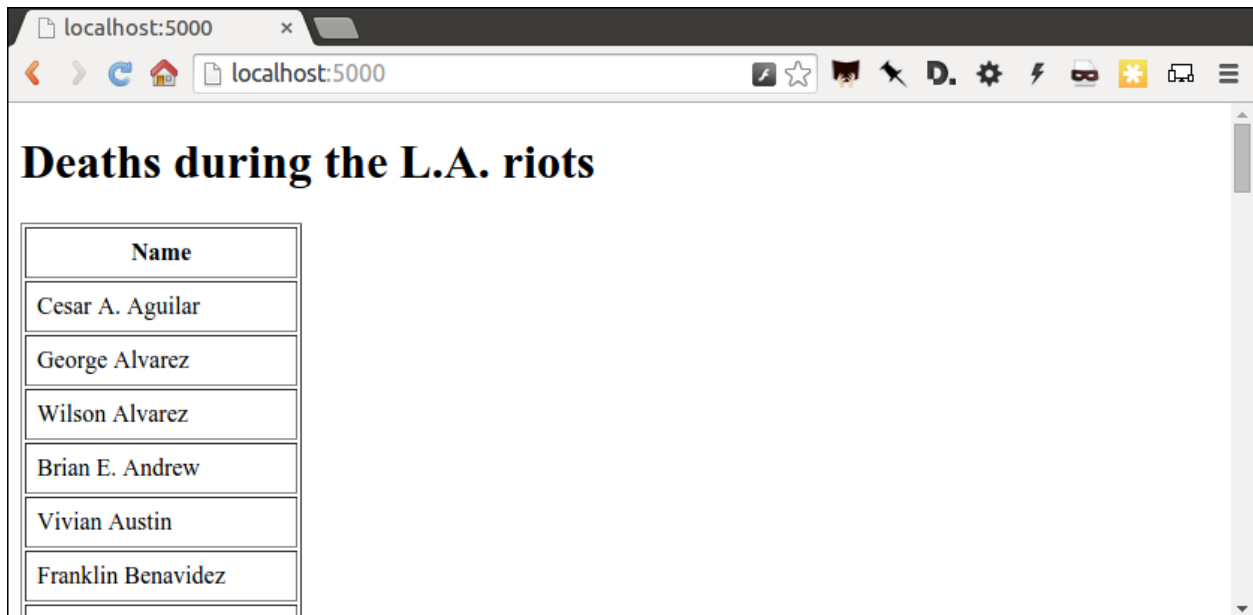


```

        <tr>
            <th>Name</th>
        </tr>
        {% for obj in object_list %}
        <tr>
            <td>{{ obj.full_name }}</td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>

```

Pause to reload your browser page.



Next expand the table to include a lot more data.

```

<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
            {% for obj in object_list %}
            <tr>
                <td>{{ obj.full_name }}</td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>

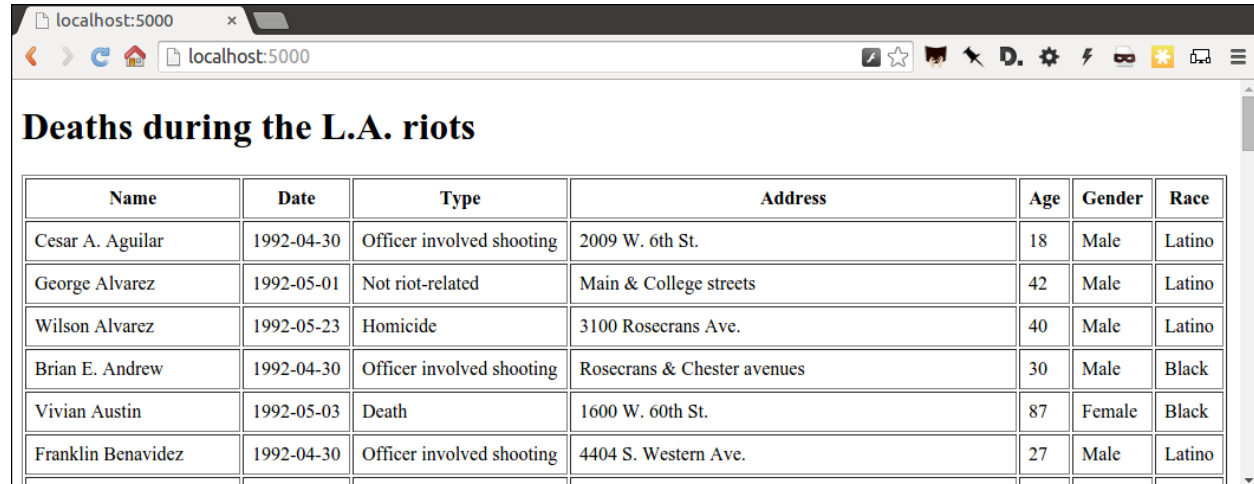
```

```

        <td>{{ obj.address }}</td>
        <td>{{ obj.age }}</td>
        <td>{{ obj.gender }}</td>
        <td>{{ obj.race }}</td>
    </tr>
    {% endfor %}
</table>
</body>
</html>

```

Reload your page in the browser again to see the change.



The screenshot shows a web browser window with the address bar at localhost:5000. The page title is "Deaths during the L.A. riots". Below the title is a table with 7 columns: Name, Date, Type, Address, Age, Gender, and Race. The table contains 7 rows of data.

Name	Date	Type	Address	Age	Gender	Race
Cesar A. Aguilar	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
George Alvarez	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
Wilson Alvarez	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino
Brian E. Andrew	1992-04-30	Officer involved shooting	Rosecrans & Chester avenues	30	Male	Black
Vivian Austin	1992-05-03	Death	1600 W. 60th St.	87	Female	Black
Franklin Benavidez	1992-04-30	Officer involved shooting	4404 S. Western Ave.	27	Male	Latino

Then commit your work.

```

$ git add .
$ git commit -m "Created basic table"
$ git push origin master

```

Next we're going to create a unique "detail" page dedicated to each person. Start by returning to `app.py` in your text editor and adding the URL and template that will help make this happen.

```

import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'r')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

@app.route('/<row_id>/')
def detail(row_id):

```

```

template = 'detail.html'
return render_template(template, row_id=row_id)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)

```

**Note:** Notice a key difference between the URL route for the index and the one we just added. This time, both the URL route and function accept an argument, named `row_id`. Our goal is for the number passed into the URL to go into the function where it can be used to pull the record with the corresponding `id` from the CSV. Once we have our hands on it, we can pass it on to the template to render its unique page.

Create a new file in your templates directory called `detail.html` for it to connect with.

```

# Macs and Linux:
$ touch templates/detail.html
# Windows:
$ start notepad++ templates/detail.html

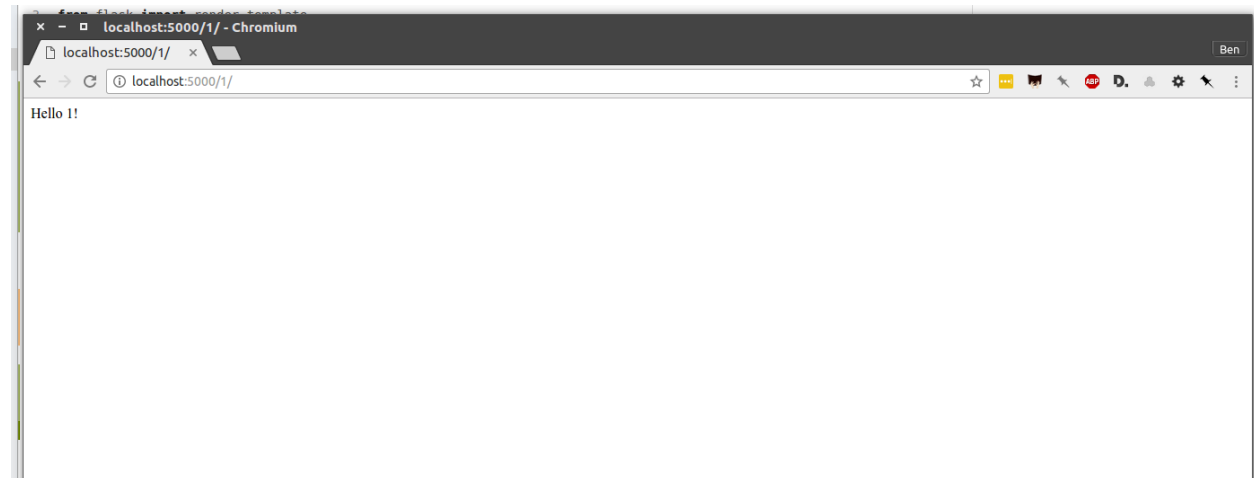
```

Put something simple in it with your text editor. We'll use the same templating language as above to print out the row id for each page.

```
Hello {{ row_id }}!
```

Then, if it's not running, restart your test server and use your browser to visit `localhost:5000/1/`, `localhost:5000/200/` or any other number.

```
$ python app.py
```



To customize the page for each person, we will need to connect the `row_id` in the URL with the `id` column in the CSV data file.

First, return to `app.py` and pull the CSV data into the detail view.

```

import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():

```

```
csv_path = './static/la-riots-deaths.csv'
csv_file = open(csv_path, 'r')
csv_obj = csv.DictReader(csv_file)
csv_list = list(csv_obj)
return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

@app.route('/<row_id>/')
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    return render_template(template, row_id=row_id)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Then have the detail function loop through the CSV data list, testing each row's id field against the row\_id provided by the URL. When you find a match, pass that row out to the template for rendering with the name object.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'r')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

@app.route('/<row_id>/')
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    for row in object_list:
        if row['id'] == row_id:
            return render_template(template, object=row)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

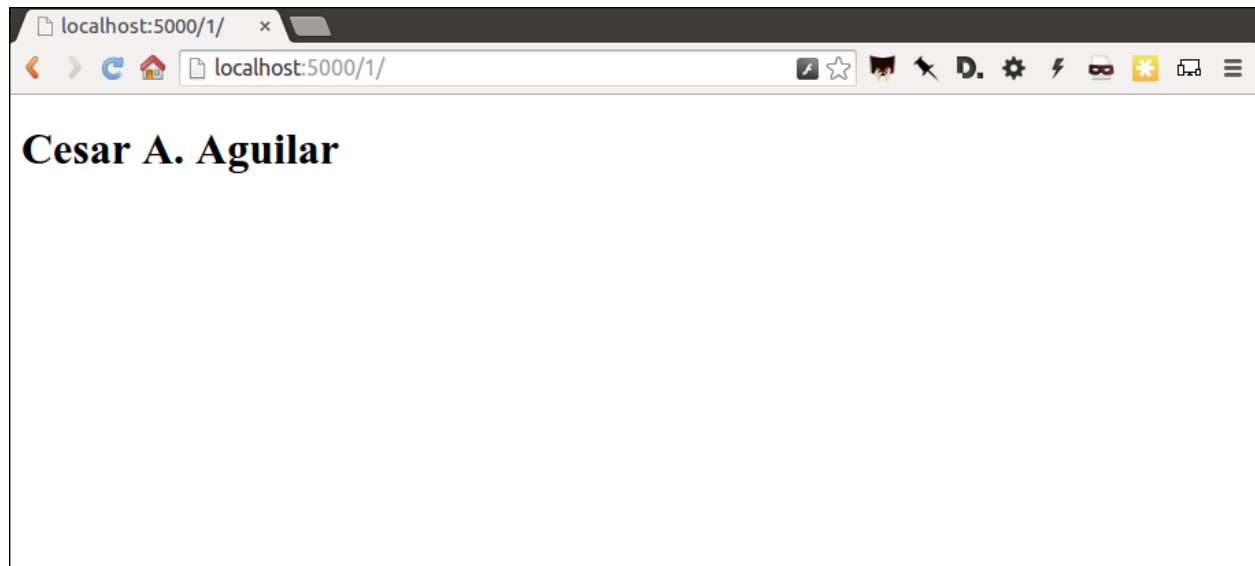
Now clear detail.html and make a new HTML document with a headline drawn from the data we've passed in from the dictionary.

```
<!doctype html>
<html lang="en">
```

```
<head></head>
<body>
  <h1>{{ object.full_name }}</h1>
</body>
</html>
```

Restart your test server and take a look at `http://localhost:5000/1/` again.

```
$ python app.py
```



Return to `index.html` and add a hyperlink to each detail page to the table.

```
<!doctype html>
<html lang="en">
  <head></head>
  <body>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Race</th>
      </tr>
      {% for obj in object_list %}
        <tr>
          <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
          <td>{{ obj.date }}</td>
          <td>{{ obj.type }}</td>
          <td>{{ obj.address }}</td>
          <td>{{ obj.age }}</td>
          <td>{{ obj.gender }}</td>
          <td>{{ obj.race }}</td>
        </tr>
      {% endfor %}
```

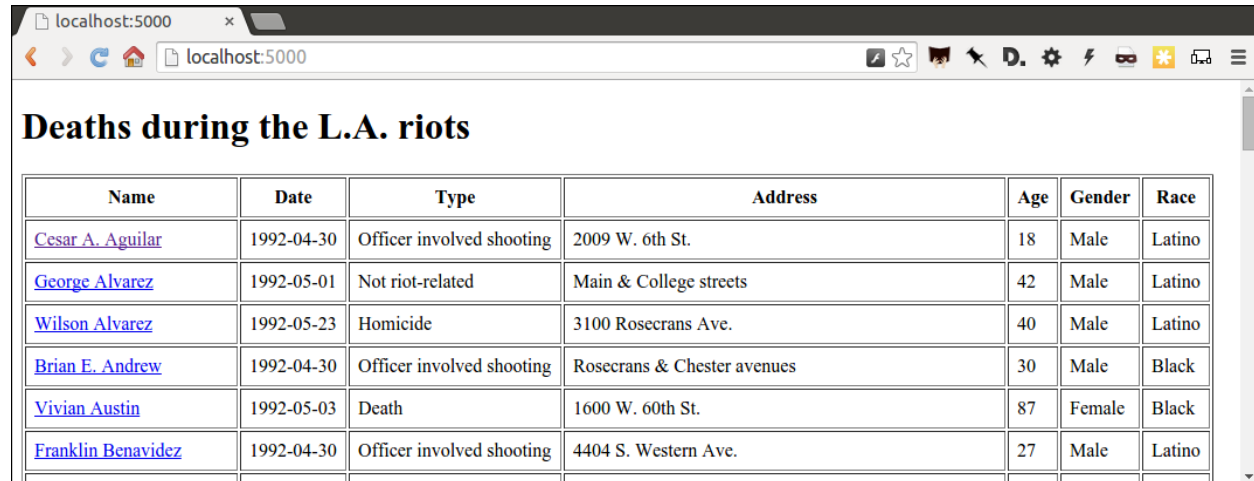
```

    </table>
  </body>
</html>

```

Restart your test server and take a look at `http://localhost:5000/`.

```
$ python app.py
```



The screenshot shows a web browser window with the address bar at `localhost:5000`. The page title is "Deaths during the L.A. riots". Below the title is a table with 7 columns: Name, Date, Type, Address, Age, Gender, and Race. The table contains 7 rows of data, each with a blue hyperlink in the Name column.

Name	Date	Type	Address	Age	Gender	Race
<a href="#">Cesar A. Aguilar</a>	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
<a href="#">George Alvarez</a>	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
<a href="#">Wilson Alvarez</a>	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino
<a href="#">Brian E. Andrew</a>	1992-04-30	Officer involved shooting	Rosecrans & Chester avenues	30	Male	Black
<a href="#">Vivian Austin</a>	1992-05-03	Death	1600 W. 60th St.	87	Female	Black
<a href="#">Franklin Benavidez</a>	1992-04-30	Officer involved shooting	4404 S. Western Ave.	27	Male	Latino

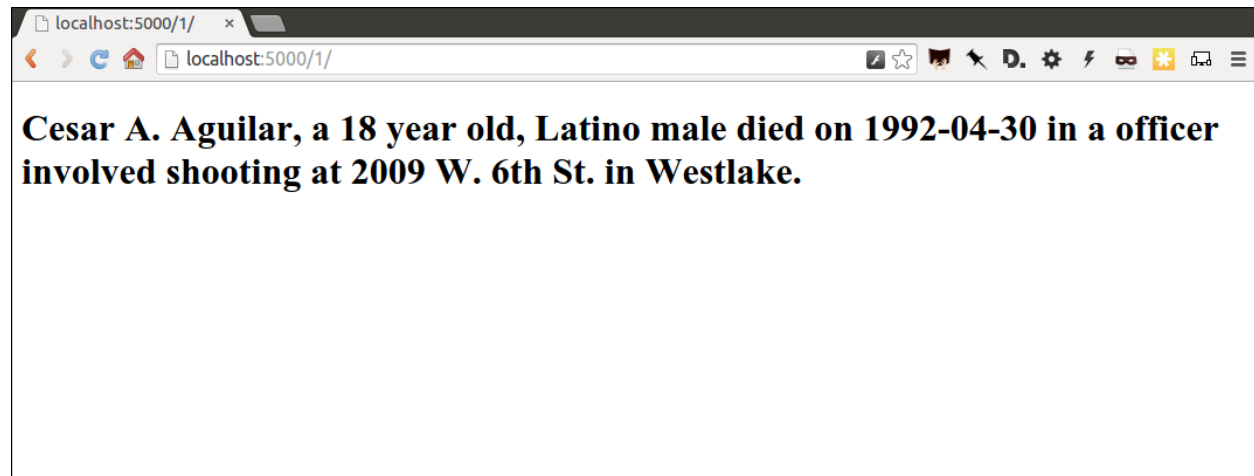
In `detail.html` you can use the rest of the data fields to write a sentence about the victim.

```

<!doctype html>
<html lang="en">
  <head></head>
  <body>
    <h1>
      {{ object.full_name }}, a {{ object.age }} year old,
      {{ object.race }} {{ object.gender|lower }} died on {{ object.date }}
      in a {{ object.type|lower }} at {{ object.address }} in {{ object.
↪neighborhood }}.
    </h1>
  </body>
</html>

```

Reload `localhost:5000/1/` to see it.

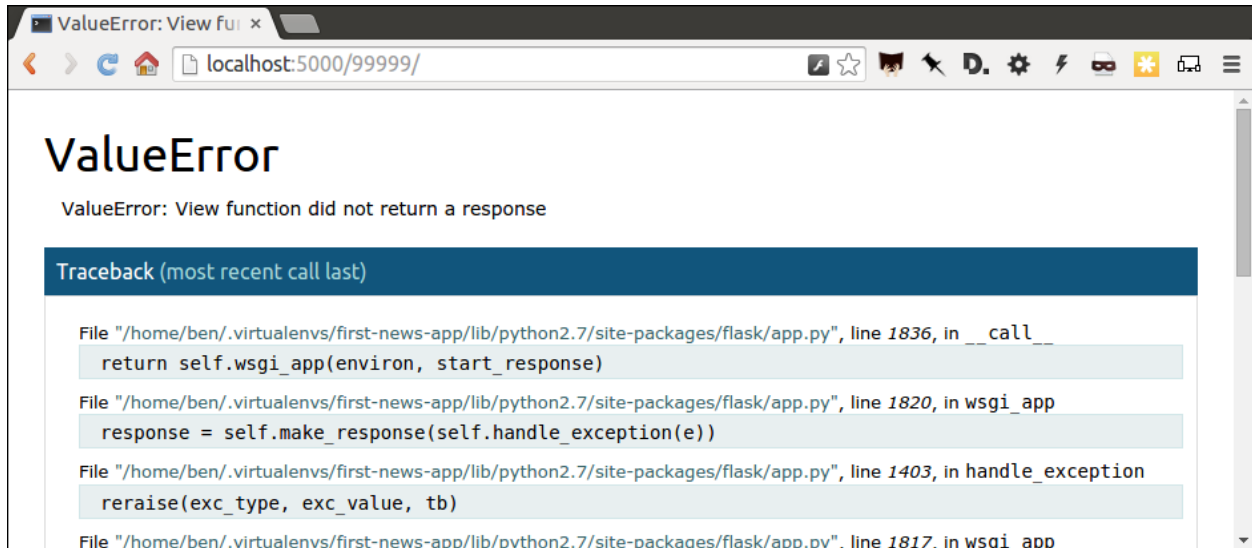


The screenshot shows a web browser window with the address bar at `localhost:5000/1/`. The page displays a single sentence in bold black text: "Cesar A. Aguilar, a 18 year old, Latino male died on 1992-04-30 in a officer involved shooting at 2009 W. 6th St. in Westlake."

Then once again commit your work.

```
$ git add .
$ git commit -m "Created a detail page about each victim."
$ git push origin master
```

One last thing before we move on. What if somebody visits an URL for an `id` that doesn't exist, like `localhost:5000/99999/`? Right now Flask throws an ugly error.



The polite thing to do is return what is called a `404 response code`. To do that with Flask, you only need to import a function called `abort` and run it after our loop finishes without finding a match.

```
import csv
from flask import Flask
from flask import abort
from flask import render_template
app = Flask(__name__)

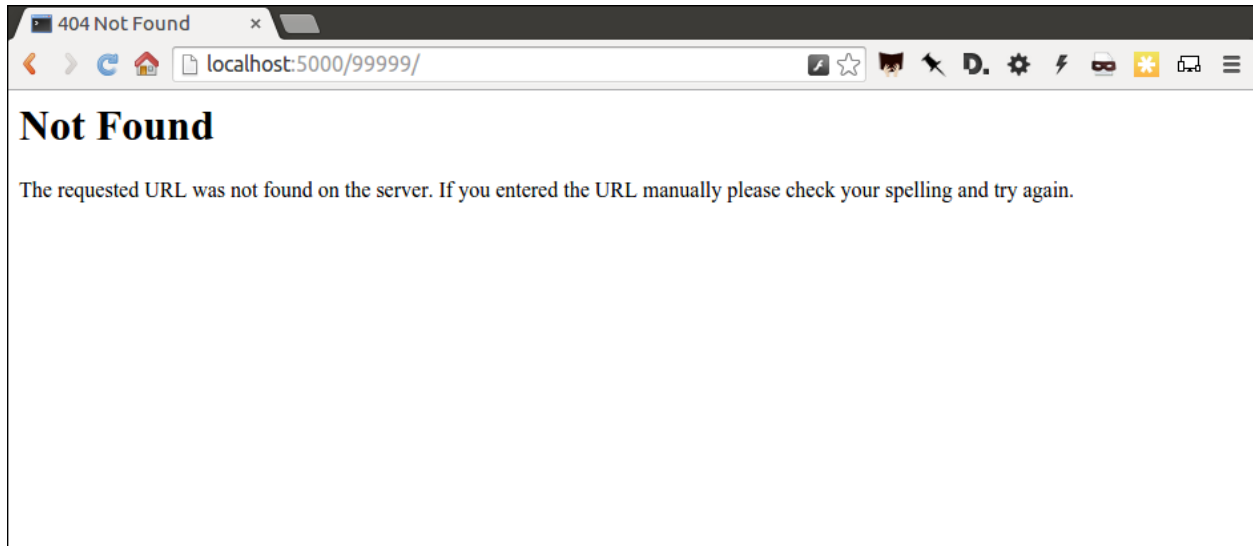
def get_csv():
    csv_path = './static/la-riots-deaths.csv'
    csv_file = open(csv_path, 'r')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

@app.route('/<row_id>/')
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    for row in object_list:
        if row['id'] == row_id:
            return render_template(template, object=row)
    abort(404)
```

```
if __name__ == '__main__':  
    app.run(debug=True, use_reloader=True)
```

Reload your bad URL and you'll see the change.





# CHAPTER 7

## Act 4: Hello JavaScript

Now we will use the [Leaflet](#) JavaScript library to create a map on each detail page showing where the victim died. Start by importing it in your page.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css"
↪ " />
    <script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
  </head>
  <body>
    <h1>
      {{ object.full_name }}, a {{ object.age }} year old,
      {{ object.race }} {{ object.gender|lower }} died on {{ object.date }}
      in a {{ object.type|lower }} at {{ object.address }} in {{ object.
↪neighborhood }}.
    </h1>
  </body>
</html>
```

Open up `detail.html` and make a map there, focus on just that victim.

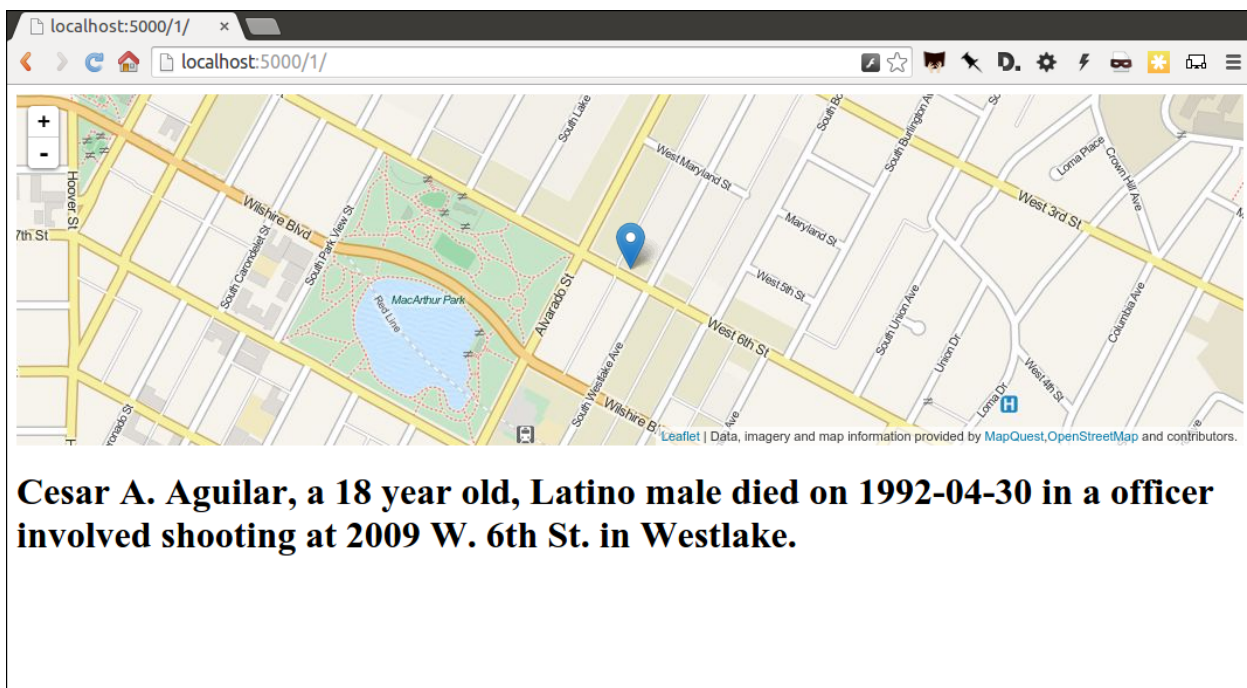
```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css"
↪ " />
    <script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
  </head>
  <body>
    <div id="map" style="width:100%; height:300px;"></div>
    <h1>
      {{ object.full_name }}, a {{ object.age }} year old,
      {{ object.race }} {{ object.gender|lower }} died on {{ object.date }}
      in a {{ object.type|lower }} at {{ object.address }} in {{ object.
↪neighborhood }}.
```

```

</h1>
<script type="text/javascript">
  var map = L.map('map').setView([{{ object.y }}, {{ object.x }}], 16);
  var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 18,
    attribution: 'Data, imagery and map information provided by <a href=
    "http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
  });
  map.addLayer(osmLayer);
  var marker = L.marker([{{ object.y }}, {{ object.x }}]).addTo(map);
</script>
</body>
</html>

```

Reload a detail page, like the one at `localhost:5000/1/`.



Commit that.

```

$ git add .
$ git commit -m "Made a map on the detail page"
$ git push origin master

```

Next we will work to make a map with every victim in `index.html` in one view.

Create an HTML element to hold the map and use Leaflet to boot it up and center on Los Angeles.

```

<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css"
    />
    <script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
  </head>

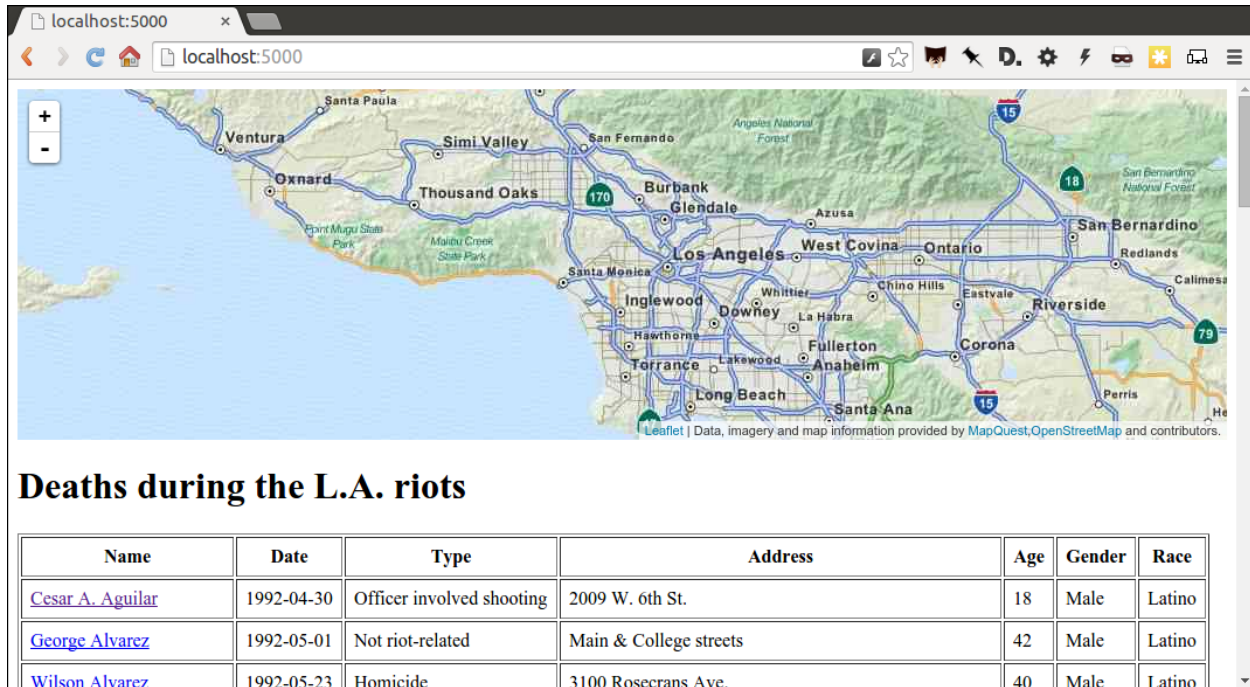
```

```

<body>
  <div id="map" style="width:100%; height:300px;"></div>
  <h1>Deaths during the L.A. riots</h1>
  <table border=1 cellpadding=7>
    <tr>
      <th>Name</th>
      <th>Date</th>
      <th>Type</th>
      <th>Address</th>
      <th>Age</th>
      <th>Gender</th>
      <th>Race</th>
    </tr>
    {% for obj in object_list %}
      <tr>
        <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
        <td>{{ obj.date }}</td>
        <td>{{ obj.type }}</td>
        <td>{{ obj.address }}</td>
        <td>{{ obj.age }}</td>
        <td>{{ obj.gender }}</td>
        <td>{{ obj.race }}</td>
      </tr>
    {% endfor %}
  </table>
  <script type="text/javascript">
    var map = L.map('map').setView([34.055, -118.35], 9);
    var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
    ↪{y}.png', {
      maxZoom: 18,
      attribution: 'Data, imagery and map information provided by <a href=
    ↪"http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a>.'
    });
    map.addLayer(osmLayer);
  </script>
</body>
</html>

```

Reload the root URL of your site at `localhost:5000`.



The screenshot shows a web browser at localhost:5000. The top half displays a map of Southern California, including cities like Ventura, Santa Paula, Simi Valley, San Fernando, Burbank, Glendale, Azusa, Los Angeles, West Covina, Ontario, San Bernardino, Redlands, Calimesa, Riverside, Corona, Fullerton, La Habra, Whittier, Inglewood, Downey, Santa Monica, Torrance, Lakewood, Long Beach, and Santa Ana. The bottom half features a table titled "Deaths during the L.A. riots" with the following data:

Name	Date	Type	Address	Age	Gender	Race
<a href="#">Cesar A. Aguilar</a>	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
<a href="#">George Alvarez</a>	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
<a href="#">Wilson Alvarez</a>	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino

Loop through the CSV data and format it as a [GeoJSON](#) object, which Leaflet can easily load.

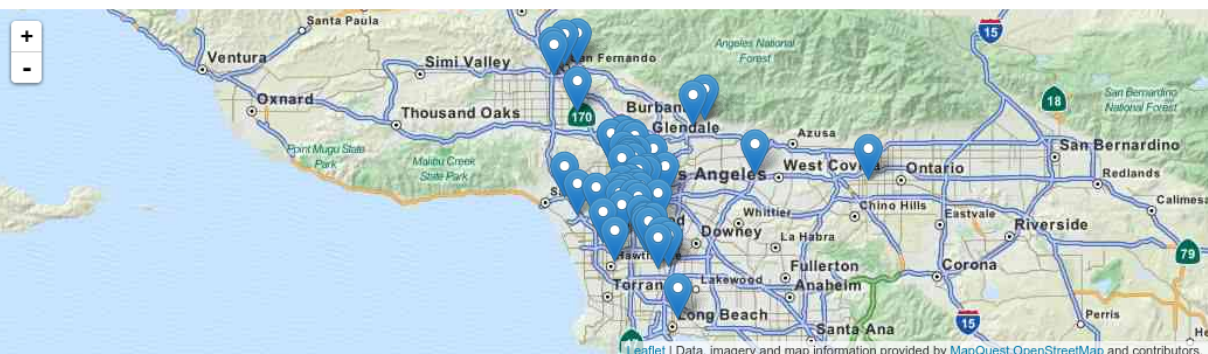
```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css"
    ↪" />
    <script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
  </head>
  <body>
    <div id="map" style="width:100%; height:300px;"></div>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Race</th>
      </tr>
      {% for obj in object_list %}
        <tr>
          <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
          <td>{{ obj.date }}</td>
          <td>{{ obj.type }}</td>
          <td>{{ obj.address }}</td>
          <td>{{ obj.age }}</td>
          <td>{{ obj.gender }}</td>
          <td>{{ obj.race }}</td>
        </tr>
      {% endfor %}
    </table>
```

```

<script type="text/javascript">
  var map = L.map('map').setView([34.055, -118.35], 9);
  var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 18,
    attribution: 'Data, imagery and map information provided by <a href=
    "http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
  });
  map.addLayer(osmLayer);
  var data = {
    "type": "FeatureCollection",
    "features": [
      {% for obj in object_list %}
      {
        "type": "Feature",
        "properties": {
          "full_name": "{{ obj.full_name }}",
          "id": "{{ obj.id }}"
        },
        "geometry": {
          "type": "Point",
          "coordinates": [{{ obj.x }}, {{ obj.y }}]
        }
      }
      {% if not loop.last %},{% endif %}
      {% endfor %}
    ]
  };
  var dataLayer = L.geoJson(data);
  map.addLayer(dataLayer);
</script>
</body>
</html>

```

Reload the page.



The screenshot shows a web browser window at localhost:5000. The main content is a map of Los Angeles and surrounding areas, with numerous blue location pins indicating specific points of interest. Below the map, there is a table titled "Deaths during the L.A. riots".

Name	Date	Type	Address	Age	Gender	Race
<a href="#">Cesar A. Aguilar</a>	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
<a href="#">George Alvarez</a>	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
<a href="#">Wilson Alvarez</a>	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino

Add a popup on the map pins that shows the name of the victim.

```

<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css"
    ↪ " />
    <script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
  </head>
  <body>
    <div id="map" style="width:100%; height:300px;"></div>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Race</th>
      </tr>
      {% for obj in object_list %}
        <tr>
          <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
          <td>{{ obj.date }}</td>
          <td>{{ obj.type }}</td>
          <td>{{ obj.address }}</td>
          <td>{{ obj.age }}</td>
          <td>{{ obj.gender }}</td>
          <td>{{ obj.race }}</td>
        </tr>
      {% endfor %}
    </table>
    <script type="text/javascript">
      var map = L.map('map').setView([34.055, -118.35], 9);
      var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        ↪ maxZoom: 18,
        attribution: 'Data, imagery and map information provided by <a href=
        ↪ "http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
      });
      map.addLayer(osmLayer);
      var data = {
        "type": "FeatureCollection",
        "features": [
          {% for obj in object_list %}
          {
            "type": "Feature",
            "properties": {
              "full_name": "{{ obj.full_name }}",
              "id": "{{ obj.id }}"
            },
            "geometry": {
              "type": "Point",
              "coordinates": [{{ obj.x }}, {{ obj.y }}]
            }
          }
          {% if not loop.last %},{% endif %}
        ]
      }
    </script>
  </body>
</html>

```

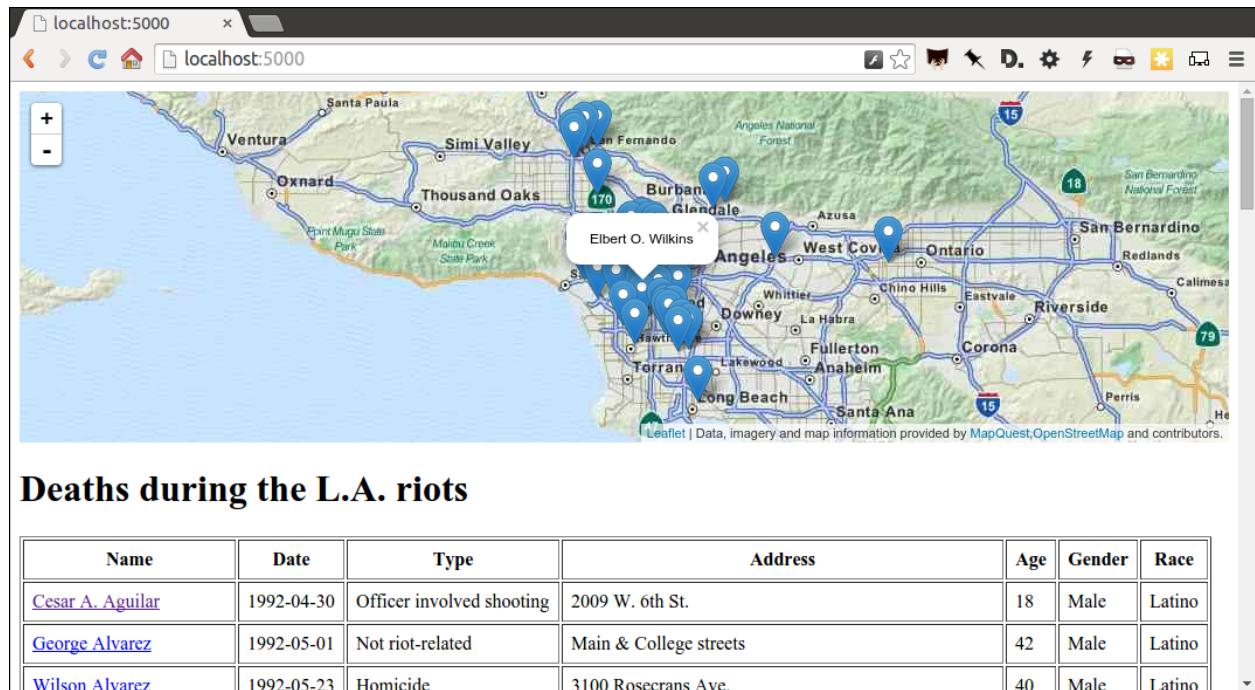


```

    ]
  };
  var dataLayer = L.geoJson(data, {
    onEachFeature: function(feature, layer) {
      layer.bindPopup(feature.properties.full_name);
    }
  });
  map.addLayer(dataLayer);
</script>
</body>
</html>

```

Reload the page and click a pin.



The screenshot shows a web browser at localhost:5000 displaying a map of Southern California. Numerous blue pins are placed on the map, representing locations of deaths during the L.A. riots. A popup for 'Elbert O. Wilkins' is visible. Below the map, the title 'Deaths during the L.A. riots' is followed by a table with the following data:

Name	Date	Type	Address	Age	Gender	Race
<a href="#">Cesar A. Aguilar</a>	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
<a href="#">George Alvarez</a>	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
<a href="#">Wilson Alvarez</a>	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino

Now wrap the name in a hyperlink to that person's detail page.

```

<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css"
    ↪" />
    <script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
  </head>
  <body>
    <div id="map" style="width:100%; height:300px;"></div>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>

```

```

        <th>Race</th>
    </tr>
    {% for obj in object_list %}
        <tr>
            <td><a href="{ { obj.id } }/">{{ obj.full_name }}</a></td>
            <td>{{ obj.date }}</td>
            <td>{{ obj.type }}</td>
            <td>{{ obj.address }}</td>
            <td>{{ obj.age }}</td>
            <td>{{ obj.gender }}</td>
            <td>{{ obj.race }}</td>
        </tr>
    {% endfor %}
</table>
<script type="text/javascript">
    var map = L.map('map').setView([34.055, -118.35], 9);
    var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
    {y}.png', {
        maxZoom: 18,
        attribution: 'Data, imagery and map information provided by <a href=
    "http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
    });
    map.addLayer(osmLayer);
    var data = {
        "type": "FeatureCollection",
        "features": [
            {% for obj in object_list %}
            {
                "type": "Feature",
                "properties": {
                    "full_name": "{ { obj.full_name } }",
                    "id": "{ { obj.id } }"
                },
                "geometry": {
                    "type": "Point",
                    "coordinates": [ { { obj.x } }, { { obj.y } } ]
                }
            }
            {% if not loop.last %},{% endif %}
            {% endfor %}
        ]
    };
    var dataLayer = L.geoJson(data, {
        onEachFeature: function(feature, layer) {
            layer.bindPopup(
                '<a href="' + feature.properties.id + '/'>' +
                feature.properties.full_name +
                '</a>'
            );
        }
    });
    map.addLayer(dataLayer);
</script>
</body>
</html>

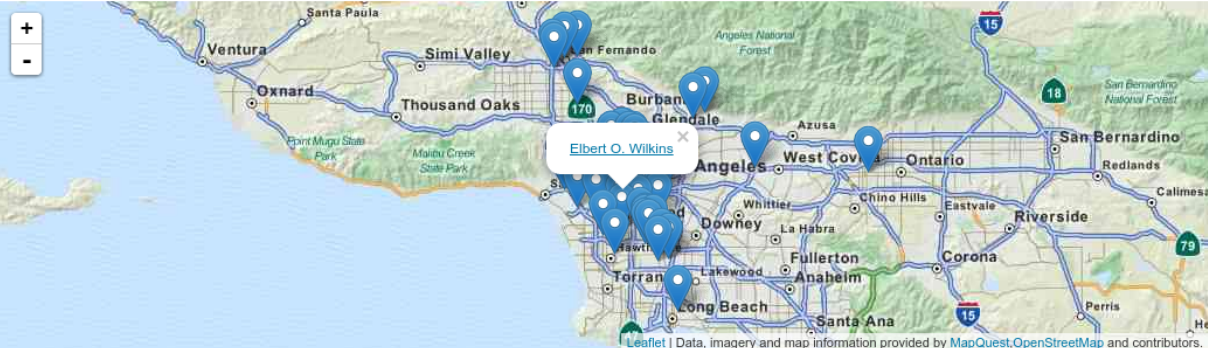
```

Reload again and click a pin.



localhost:5000

localhost:5000



Deaths during the L.A. riots

Name	Date	Type	Address	Age	Gender	Race
<a href="#">Cesar A. Aguilar</a>	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
<a href="#">George Alvarez</a>	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
<a href="#">Wilson Alvarez</a>	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino

Commit your map.

```
$ git add .  
$ git commit -m "Made a map on the index page"  
$ git push origin master
```



## CHAPTER 8

---

### Act 5: Hello Internet

---

In this final act, we will publish your application to the Internet using [Frozen Flask](#), a Python library that saves every page you've made with Flask as a flat file that can be uploaded to the web. This is an alternative publishing method that does not require you configure and host an full-fledged Internet server.

First, use `pip` to install Frozen Flask from the command line.

```
$ pip install Frozen-Flask
```

Create a new file called `freeze.py` where we will configure which pages it should convert into flat files.

```
# Mac and Linux:
$ touch freeze.py
# Windows:
$ start notepad++ freeze.py
```

Use your text editor to write a basic Frozen Flask configuration.

```
from flask_frozen import Freezer
from app import app
freezer = Freezer(app)

if __name__ == '__main__':
    freezer.freeze()
```

Now run it from the command line, which will create a new directory called `build` filled with a set of flattened files.

```
$ python freeze.py
```

Use your browser to open up one of the local files in `build`, rather than visit the dynamically generated pages we created at `localhost`.

You will notice that the default Frozen Flask configuration only flattened out `index.html`, and not all your detail pages our template could generate using the data file.

To flatten those, again edit `freeze.py` to give it the instructions it needs to make a page for every record in the source CSV.

```
from flask_frozen import Freezer
from app import app, get_csv
freezer = Freezer(app)

@freezer.register_generator
def detail():
    for row in get_csv():
        yield {'row_id': row['id']}

if __name__ == '__main__':
    freezer.freeze()
```

Run it again from the command line and notice all the additional pages it made in the `build` directory. Try opening one in your browser.

```
$ python freeze.py
```

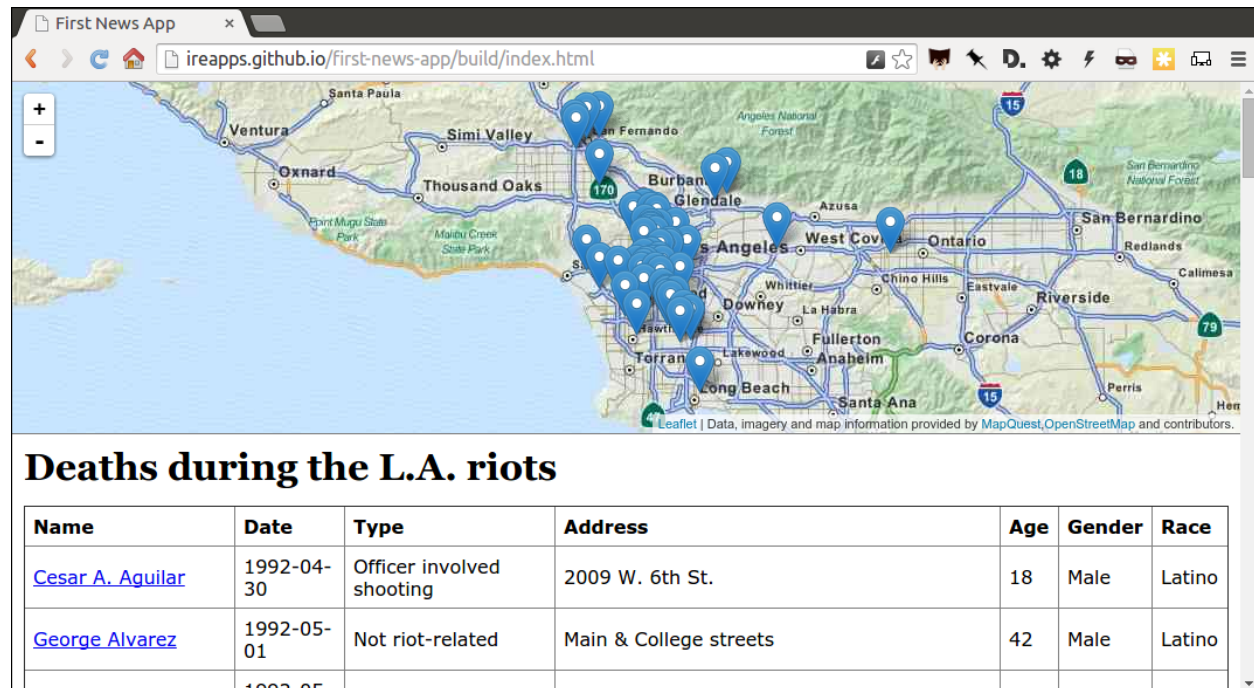
Commit all of the flat pages to the repository.

```
$ git add .
$ git commit -m "Froze my app"
$ git push origin master
```

Finally, we will publish these static files to the web using [GitHub's Pages](#) feature. All it requires is that we create a new branch in our repository called `gh-pages` and push our files up to GitHub there.

```
$ git checkout -b gh-pages # Create the new branch
$ git merge master # Pull in all the code from the master branch
$ git push origin gh-pages # Push up to GitHub from your new branch
```

Now wait a minute or two, then visit `http://<yourusername>.github.io/first-news-app/build/index.html` to cross the finish line.



The screenshot shows a web browser window with the address `ireapps.github.io/first-news-app/build/index.html`. The main content is a map of the Los Angeles area with numerous blue location pins. Below the map, there is a section titled "Deaths during the L.A. riots" which contains a table with the following data:

Name	Date	Type	Address	Age	Gender	Race
<a href="#">Cesar A. Aguilar</a>	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
<a href="#">George Alvarez</a>	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
	1992-05-					

---

**Note:** If your page does not appear, make sure that you have verified your email address with GitHub. It is required before the site will allow publishing pages. And keep in mind there are many other options for publishing flat files, like [Amazon's S3 service](#).

---

So you've built a site. But it's kind of janky looking. The next chapter, which we won't have time for in class, will show you how to dress it up to look like the [demonstration site](#).



## CHAPTER 9

---

### Epilogue: Hello CSS

---

Before you get started, move back to the master branch of your repository.

```
$ git checkout master
```

The first step is to create a stylesheet in the static directory where **CSS** code that controls the design of the page can be stored.

```
# Macs or Linux:
$ touch static/style.css
# In Windows fire it up in your text editor right away:
$ start notepad++ static/style.css
```

Add the style tag to the top of `index.html` so it imported on the page. Flask's built-in `url_for` method will create the URL for us.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.
↪7.7/leaflet.css" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/leaflet.js">
↪</script>
  </head>
  <body>
    <div id="map" style="width:100%; height:300px;"></div>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>
```

```
        <th>Race</th>
    </tr>
    {% for obj in object_list %}
        <tr>
            <td><a href="{ {{ obj.id }} }/">{{ obj.full_name }}</a></td>
            <td>{{ obj.date }}</td>
            <td>{{ obj.type }}</td>
            <td>{{ obj.address }}</td>
            <td>{{ obj.age }}</td>
            <td>{{ obj.gender }}</td>
            <td>{{ obj.race }}</td>
        </tr>
    {% endfor %}
</table>
<script type="text/javascript">
    var map = L.map('map').setView([34.055, -118.35], 9);
    var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
↪{y}.png', {
        maxZoom: 18,
        attribution: 'Data, imagery and map information provided by <a href=
↪"http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
    });
    map.addLayer(osmLayer);
    var data = {
        "type": "FeatureCollection",
        "features": [
            {% for obj in object_list %}
            {
                "type": "Feature",
                "properties": {
                    "full_name": "{ {{ obj.full_name }}",
                    "id": "{ {{ obj.id }}"
                },
                "geometry": {
                    "type": "Point",
                    "coordinates": [{{ obj.x }}, {{ obj.y }}]
                }
            }
            {% if not loop.last %},{% endif %}
        ]
    };
    var dataLayer = L.geoJson(data, {
        onEachFeature: function(feature, layer) {
            layer.bindPopup(
                '<a href="' + feature.properties.id + '/'>' +
                feature.properties.full_name +
                '</a>'
            );
        }
    });
    map.addLayer(dataLayer);
</script>
</body>
</html>
```

Before we start styling the page, let's do a little reorganization of the HTML to make a little more like a news site.

First, download this [IRE logo](#) and throw in the `static` directory. We'll add that as an image in a new navigation bar



at the top of the site, then zip up the headline and move it above the map with with a new byline.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.
↪7.7/leaflet.css" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/leaflet.js">
↪</script>
  </head>
  <body>
    <nav>
      <a href="http://first-news-app.readthedocs.org/">
        
      </a>
    </nav>
    <header>
      <h1>These are the 60 people who died during the L.A. riots</h1>
      <div class="byline">
        By <a href="http://first-news-app.readthedocs.org/">The First News_
↪App Tutorial</a>
      </div>
    </header>
    <div id="map" style="width:100%; height:300px;"></div>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Race</th>
      </tr>
      {% for obj in object_list %}
        <tr>
          <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
          <td>{{ obj.date }}</td>
          <td>{{ obj.type }}</td>
          <td>{{ obj.address }}</td>
          <td>{{ obj.age }}</td>
          <td>{{ obj.gender }}</td>
          <td>{{ obj.race }}</td>
        </tr>
      {% endfor %}
    </table>
    <script type="text/javascript">
      var map = L.map('map').setView([34.055, -118.35], 9);
      var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
↪{y}.png', {
        maxZoom: 18,
        attribution: 'Data, imagery and map information provided by <a href=
↪"http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
      });
      map.addLayer(osmLayer);
      var data = {
        "type": "FeatureCollection",
        "features": [
```

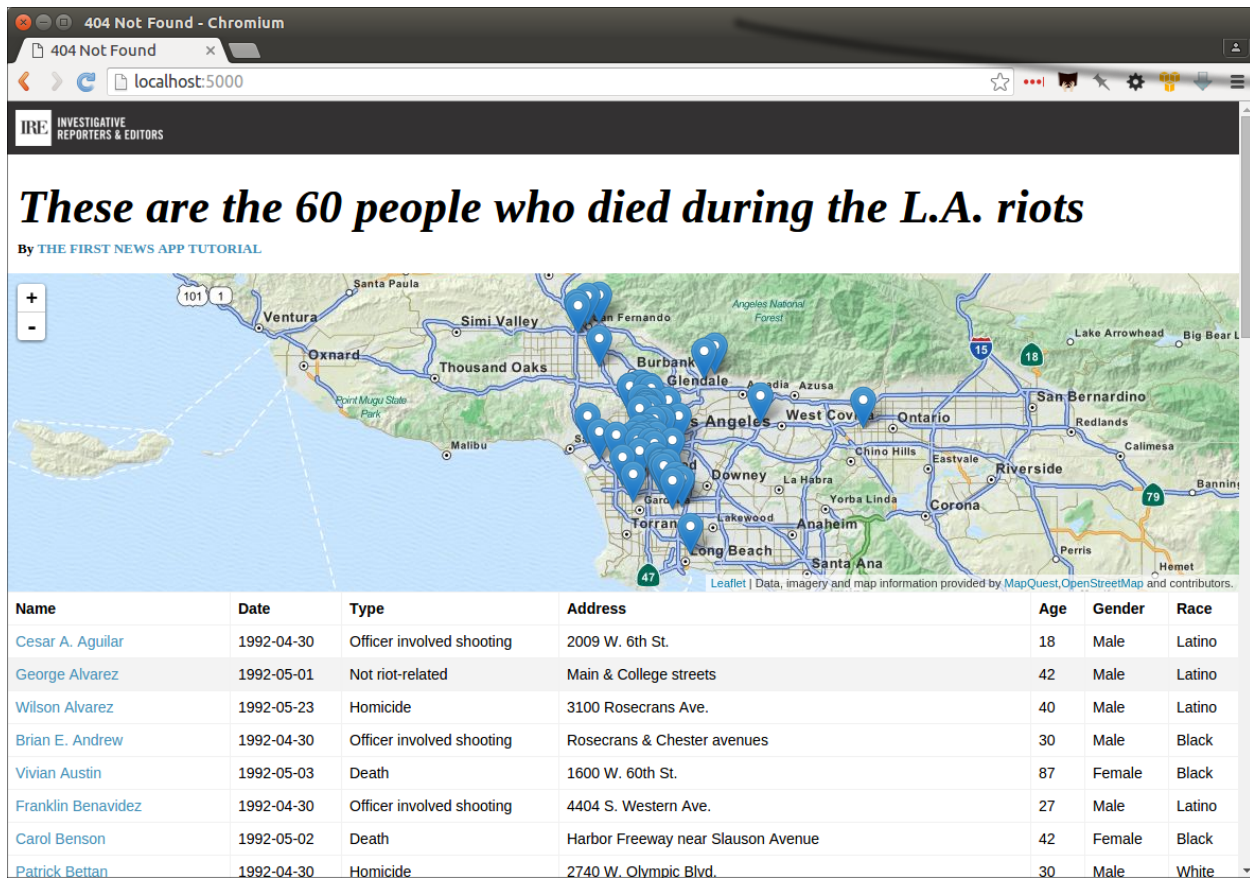
```
        {% for obj in object_list %}
        {
            "type": "Feature",
            "properties": {
                "full_name": "{{ obj.full_name }}",
                "id": "{{ obj.id }}"
            },
            "geometry": {
                "type": "Point",
                "coordinates": [{{ obj.x }}, {{ obj.y }}]
            }
        }
        {% if not loop.last %},{% endif %}
        {% endfor %}
    ]
};
var dataLayer = L.geoJson(data, {
    onEachFeature: function(feature, layer) {
        layer.bindPopup(
            '<a href="' + feature.properties.id + '/'>' +
                feature.properties.full_name +
            '</a>'
        );
    }
});
map.addLayer(dataLayer);
</script>
</body>
</html>
```

Now go into `style.css` and toss in some style we've prepared that will draw in a dark top bar, limit the width of the page and tighten up the rest of the page.

```
body {
    margin: 0 auto;
    padding: 0;
    font-family: Verdana, sans-serif;
    background-color: #F2EFEC;
    max-width: 1200px;
}
nav {
    background-color: #333132;
    width: 100%;
    height: 50px;
}
nav img {
    height: 34px;
    padding: 8px;
}
header {
    margin: 25px 10px 15px 10px;
    font-family: Times, Times New Roman, serif;
}
h1 {
    margin: 0;
    padding: 0;
    font-size: 44px;
    line-height: 50px;
    font-weight: bold;
}
```

```
    font-style: italic;
    text-shadow: 0.3px 0.3px 0px gray;
    letter-spacing: .01em;
}
.byline {
    margin: 6px 0 0 0;
    font-size: 13px;
    font-weight: bold;
}
.byline a {
    text-transform: uppercase;
}
table {
    border-collapse: collapse;
    margin: 0 0 20px 0;
    border-width: 0;
    width: 100%;
    font-size: 14px;
}
th {
    text-align: left;
}
tr, td, th {
    border-color: #f2f2f2;
}
tr:hover {
    background-color: #f3f3f3;
}
p {
    line-height: 140%;
}
a {
    color: #4591B8;
    text-decoration: none;
}
a:hover {
    text-decoration: underline;
}
```

Reload the page and you should see something a little more presentable.



**These are the 60 people who died during the L.A. riots**  
By THE FIRST NEWS APP TUTORIAL

Name	Date	Type	Address	Age	Gender	Race
Cesar A. Aguilar	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
George Alvarez	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
Wilson Alvarez	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino
Brian E. Andrew	1992-04-30	Officer involved shooting	Rosecrans & Chester avenues	30	Male	Black
Vivian Austin	1992-05-03	Death	1600 W. 60th St.	87	Female	Black
Franklin Benavidez	1992-04-30	Officer involved shooting	4404 S. Western Ave.	27	Male	Latino
Carol Benson	1992-05-02	Death	Harbor Freeway near Slauson Avenue	42	Female	Black
Patrick Bettan	1992-04-30	Homicide	2740 W. Olympic Blvd.	30	Male	White

The next step is to upgrade the styles to reshape the page on smaller devices like tablets and phones. This is done using a system known as **responsive design** and **CSS media queries** that set different style rules at different device sizes.

First the HTML page needs an extra tag to turn the system on.

```
<!doctype html>
<html lang="en">
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{ { url_for('static', filename='style.css') } }" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/leaflet.css" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/leaflet.js">
  </script>
  </head>
  <body>
    <nav>
      <a href="http://first-news-app.readthedocs.org/">
        
      </a>
    </nav>
    <header>
      <h1>These are the 60 people who died during the L.A. riots</h1>
      <div class="byline">
        By <a href="http://first-news-app.readthedocs.org/">The First News_
      </div>
    </header>
```

```

<div id="map" style="width:100%; height:300px;"></div>
<table border=1 cellpadding=7>
  <tr>
    <th>Name</th>
    <th>Date</th>
    <th>Type</th>
    <th>Address</th>
    <th>Age</th>
    <th>Gender</th>
    <th>Race</th>
  </tr>
  {% for obj in object_list %}
    <tr>
      <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
      <td>{{ obj.date }}</td>
      <td>{{ obj.type }}</td>
      <td>{{ obj.address }}</td>
      <td>{{ obj.age }}</td>
      <td>{{ obj.gender }}</td>
      <td>{{ obj.race }}</td>
    </tr>
  {% endfor %}
</table>
<script type="text/javascript">
  var map = L.map('map').setView([34.055, -118.35], 9);
  var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
↪{y}.png', {
    maxZoom: 18,
    attribution: 'Data, imagery and map information provided by <a href=
↪"http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
  });
  map.addLayer(osmLayer);
  var data = {
    "type": "FeatureCollection",
    "features": [
      {% for obj in object_list %}
      {
        "type": "Feature",
        "properties": {
          "full_name": "{{ obj.full_name }}",
          "id": "{{ obj.id }}"
        },
        "geometry": {
          "type": "Point",
          "coordinates": [{{ obj.x }}, {{ obj.y }}]
        }
      }
      {% if not loop.last %},{% endif %}
      {% endfor %}
    ]
  };
  var dataLayer = L.geoJson(data, {
    onEachFeature: function(feature, layer) {
      layer.bindPopup(
        '<a href="' + feature.properties.id + '/'>' +
          feature.properties.full_name +
          '</a>'
      );
    }
  });

```

```
    });  
    map.addLayer(dataLayer);  
  </script>  
</body>  
</html>
```

Now the `style.css` file should be expanded to include media queries that will drop columns from the table on smaller devices.

```
body {  
    margin: 0 auto;  
    padding: 0;  
    font-family: Verdana, sans-serif;  
    background-color: #F2EFEC;  
    max-width: 1200px;  
}  
nav {  
    background-color: #333132;  
    width: 100%;  
    height: 50px;  
}  
nav img {  
    height: 34px;  
    padding: 8px;  
}  
header {  
    margin: 25px 10px 15px 10px;  
    font-family: Times, Times New Roman, serif;  
}  
h1 {  
    margin: 0;  
    padding: 0;  
    font-size: 44px;  
    line-height: 50px;  
    font-weight: bold;  
    font-style: italic;  
    text-shadow: 0.3px 0.3px 0px gray;  
    letter-spacing: .01em;  
}  
.byline {  
    margin: 6px 0 0 0;  
    font-size: 13px;  
    font-weight: bold;  
}  
.byline a {  
    text-transform: uppercase;  
}  
table {  
    border-collapse: collapse;  
    margin: 0 0 20px 0;  
    border-width: 0;  
    width: 100%;  
    font-size: 14px;  
}  
th {  
    text-align: left;  
}  
tr, td, th {
```

```
        border-color: #f2f2f2;
    }
    tr:hover {
        background-color: #f3f3f3;
    }
    p {
        line-height: 140%;
    }
    a {
        color: #4591B8;
        text-decoration: none;
    }
    a:hover {
        text-decoration: underline;
    }
    @media (max-width: 979px) {
        tr th:nth-of-type(n+3),
        tr td:nth-of-type(n+3) {
            display: none;
        }
    }
    @media (max-width: 420px) {
        tr th:nth-of-type(n+2),
        tr td:nth-of-type(n+2) {
            display: none;
        }
    }
}
```

Reload the page and size down your browser to see how the page should appear when visited by a mobile phone.

localhost:5000 - Chromium

localhost:5000

localhost:5000

IRE

INVESTIGATIVE  
REPORTERS & EDITORS

# *These are the 60 people who died during the L.A. riots*

By [THE FIRST NEWS APP TUTORIAL](#)



A map of the Los Angeles metropolitan area showing the locations of 60 deaths during the 1992 riots. The map includes labels for cities like Simi Valley, Thousand Oaks, San Fernando, Burbank, Glendale, Los Angeles, Downey, Torrance, and Long Beach. The pins are concentrated in the central urban areas, particularly around Los Angeles and its immediate surroundings. The map is provided by Leaflet, using data from MapQuest, OpenStreetMap, and contributors.

Name

Cesar A. Aguilar

Leaflet | Data, imagery and map information provided by MapQuest, OpenStreetMap and contributors.



We can punch up the map markers by replacing the Leaflet default pins with custom designs from the [Mapbox's open-source Maki set](#).

Download [these two](#) black pin images and add them to your `static` directory.

Now expand our Leaflet JavaScript code to substitute these images for the defaults.

```
<!doctype html>
<html lang="en">
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.
↪7.7/leaflet.css" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/leaflet.js">
↪</script>
  </head>
  <body>
    <nav>
      <a href="http://first-news-app.readthedocs.org/">
        
      </a>
    </nav>
    <header>
      <h1>These are the 60 people who died during the L.A. riots</h1>
      <div class="byline">
        By <a href="http://first-news-app.readthedocs.org/">The First News_
↪App Tutorial</a>
      </div>
    </header>
    <div id="map" style="width:100%; height:300px;"></div>
    <table border=1 cellpadding=7>
      <tr>
        <th>Name</th>
        <th>Date</th>
        <th>Type</th>
        <th>Address</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Race</th>
      </tr>
      {% for obj in object_list %}
        <tr>
          <td><a href="{{ obj.id }}">{{ obj.full_name }}</a></td>
          <td>{{ obj.date }}</td>
          <td>{{ obj.type }}</td>
          <td>{{ obj.address }}</td>
          <td>{{ obj.age }}</td>
          <td>{{ obj.gender }}</td>
          <td>{{ obj.race }}</td>
        </tr>
      {% endfor %}
    </table>
    <script type="text/javascript">
      var map = L.map('map').setView([34.055, -118.35], 9);
      var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
↪{y}.png', {
        maxZoom: 18,
        attribution: 'Data, imagery and map information provided by <a href=
↪"http://www.openstreetmap.org/" target=" blank">OpenStreetMap</a> and contributors.'
```

```
});
map.addLayer(osmLayer);
var data = {
  "type": "FeatureCollection",
  "features": [
    {% for obj in object_list %}
    {
      "type": "Feature",
      "properties": {
        "full_name": "{{ obj.full_name }}",
        "id": "{{ obj.id }}"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [{{ obj.x }}, {{ obj.y }}]
      }
    }
    {% if not loop.last %},{% endif %}
  {% endfor %}
];
var blackIcon = L.Icon.extend({
  options: {
    iconUrl: "{{ url_for('static', filename='marker-24.png') }}",
    iconSize: [24, 24]
  }
});
var dataLayer = L.geoJson(data, {
  pointToLayer: function (feature, latlng) {
    return L.marker(latlng, {icon: new blackIcon()});
  },
  onEachFeature: function(feature, layer) {
    layer.bindPopup(
      '<a href="' + feature.properties.id + '>' +
        feature.properties.full_name +
      '</a>'
    );
  }
});
map.addLayer(dataLayer);
</script>
</body>
</html>
```

That will restyle the map to look like this.

First News App - Chromium

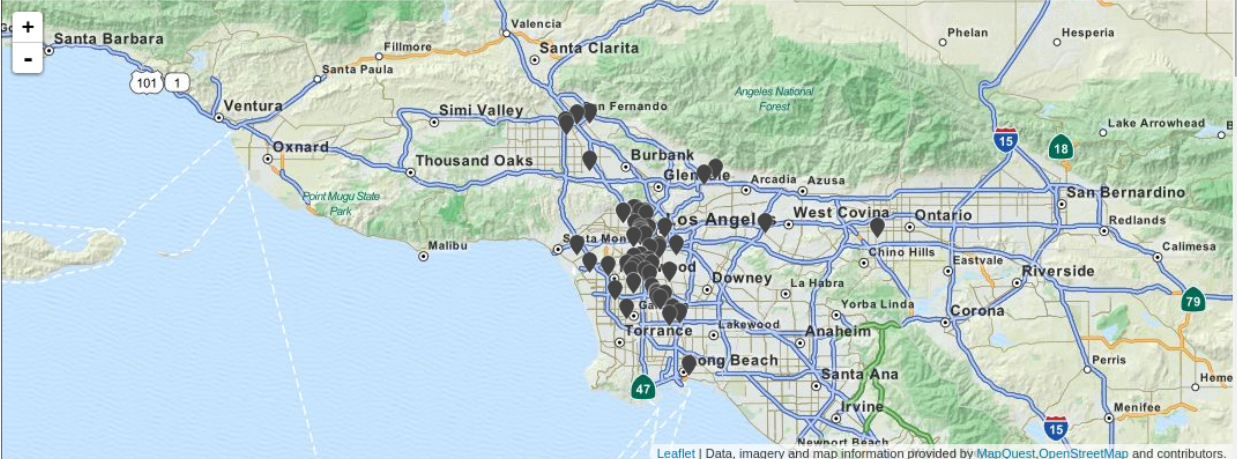
First News App

localhost:5000

IRE INVESTIGATIVE REPORTERS & EDITORS

## These are the 60 people who died during the L.A. riots

By THE FIRST NEWS APP TUTORIAL



Name	Date	Type	Address	Age	Gender	Race
Cesar A. Aguilar	1992-04-30	Officer involved shooting	2009 W. 6th St.	18	Male	Latino
George Alvarez	1992-05-01	Not riot-related	Main & College streets	42	Male	Latino
Wilson Alvarez	1992-05-23	Homicide	3100 Rosecrans Ave.	40	Male	Latino
Brian E. Andrew	1992-04-30	Officer involved shooting	Rosecrans & Chester avenues	30	Male	Black
Vivian Austin	1992-05-03	Death	1600 W. 60th St.	87	Female	Black
open.mapquest.co.uk	1992-04-30	Officer involved shooting	4404 S. Western Ave.	27	Male	Latino

Extending this new design to detail page is simply a matter of repeating the steps above.

```

<!doctype html>
<html lang="en">
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.
    ↪7.7/leaflet.css" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/leaflet.js">
    ↪</script>
  </head>
  <body>
    <nav>
      <a href="http://first-news-app.readthedocs.org/">
        
      </a>
    </nav>
    <header>
      <h1>{{ object.full_name }}, a {{ object.age }} year old, {{ object.
    ↪race|lower }} {{ object.gender|lower }} died on {{ object.date }}
      in a {{ object.type|lower }} at {{ object.address }} in {{ object.neighborhood }}.
    ↪</h1>

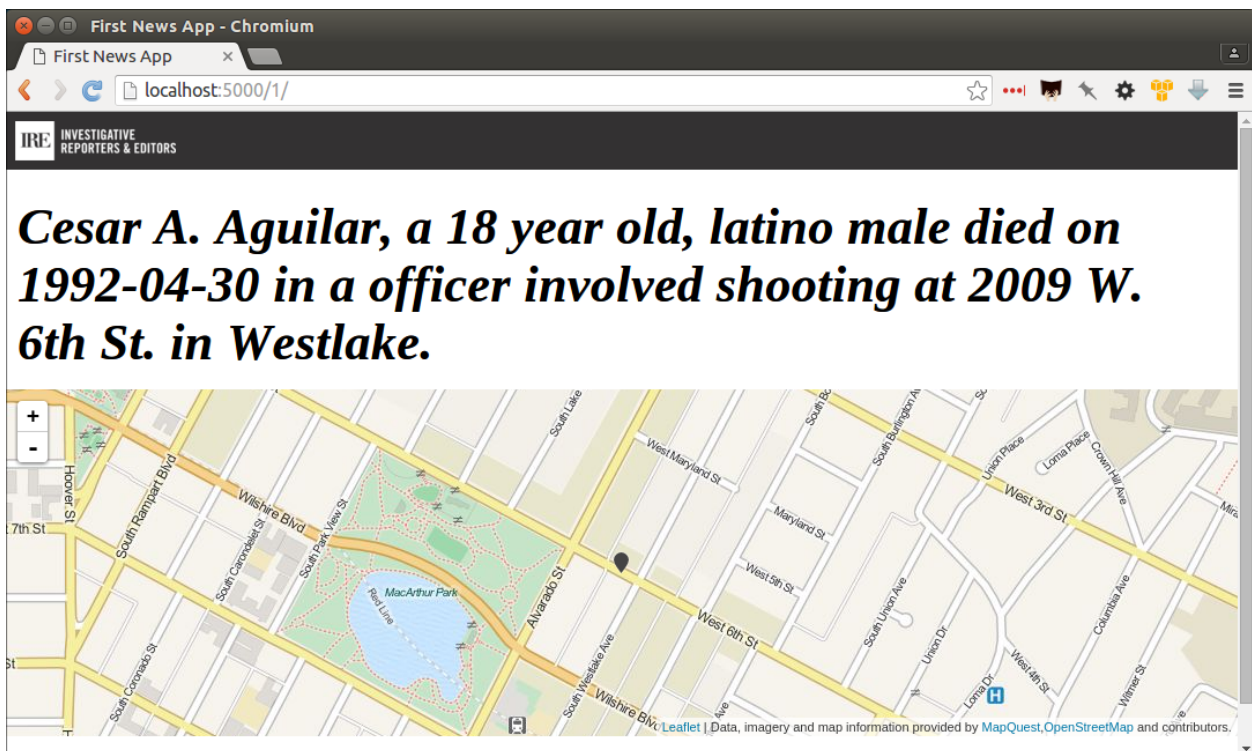
```

```

</header>
<div id="map" style="width:100%; height:300px;"></div>
<script type="text/javascript">
  var map = L.map('map').setView([{{ object.y }}, {{ object.x }}], 16);
  var osmLayer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/
↪{y}.png', {
    maxZoom: 18,
    attribution: 'Data, imagery and map information provided by <a href=
↪"http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a> and contributors.'
  });
  map.addLayer(osmLayer);
  var blackIcon = L.Icon.extend({
    options: {
      iconUrl: "{{ url_for('static', filename='marker-24.png') }}",
      iconSize: [24, 24]
    }
  });
  var marker = L.marker([{{ object.y }}, {{ object.x }}], {icon: new ↪
↪blackIcon()}).addTo(map);
</script>
</body>
</html>

```

That should shape up the page like this.



Now it is time to build out all the pages by running the freeze script that will save all of the pages again.

```
$ python freeze.py
```

Commit all of the flat pages to the repository.

```
$ git add .  
$ git commit -m "Froze my restyled app"  
$ git push origin master
```

Republish your work by going back to the `gh-pages` branch and pushing up the code.

```
$ git checkout gh-pages  
$ git merge master  
$ git push origin gh-pages
```

Now wait a minute or two, then visit `http://<yourusername>.github.io/first-news-app/build/index.html` to see the restyled application.