fabulist Documentation

Release 1.2.0

Martin Wendt

Jan 01, 2018

Contents

Generate random strings that make sense.

Project https://github.com/mar10/fabulist/

Version 1.2, Jan 01, 2018

CHAPTER 1

Installation

Requirements: Python 2.7+ or 3 is required.

Releases are hosted on PyPI and can be installed using pip:

```
$ pip install fabulist
```

CHAPTER 2

User Guide

All random strings are generated by methods of the `Fabulist` class. Have a look at the `Reference Guide` for details.

2.1 Some Examples

This code:

```
from fabulist import Fabulist
fab = Fabulist()

# Print a random noun:
print(fab.get_word("noun"):

# Print a random adjective, tagged 'positive':
print(fab.get_word("adj", "#positive"):

# Print a random name, prefix with Mr./Mrs. and allow a middle-initial:
print(fab.get_name("mr:middle"):

# Print three random fragments, prefix with a/an:
for s in fab.generate_quotes("Look, $(noun:an:#animal)!", 3):
    print(s)
```

will produce something like this:

```
obligation
kind
Mrs. Julia P. Hughes
Look, a tiger
Look, an eagle!
Look, a bug!
```

2.2 More Examples

The following output was generated by this demo code:

```
bash-3.2$ python -m tests.demo
get_verb():
-  counsel
-  oblige
-  discover
Friendly warnings:
-  Don't sell with my alarm or I'll nip your boxes.
-  Don't stretch with my obligation or I'll pacify your guarantees.
-  Don't sneak with my guide or I'll thump your deads.
Software release names:
-  detailed-bear
-  unfortunate-rabbit
-  weak-otter
Compare:
-  One gather may be far, but two gathers are farther.
-  One exchange may be tidy, but two exchanges are tidier.
-  One smile may be delayed, but two smiles are delayeder.
Names:
- My name is Mr. Isaac James
- My name is Mr. Colin Powell
- My name is Mr. Jacob K. Wilson
Introduction:
- Friends call me Harry, you can call me Mr. Harry A. May.
  May I introduce you to my wife Mrs. Rose May?
- Friends call me Keith, you can call me Mr. Keith Taylor.
  May I introduce you to my wife Mrs. Madeleine Taylor?
- Friends call me Dominic, you can call me Mr. Dominic W. Ross.
  May I introduce you to my wife Mrs. Lily Ross?
Compliments:
-  You have very reliable performances.
-  You have very intelligent leagues.
-  You have very good fingers.
Blessings:
-  May your scripts express joyfulliest.
-  May your motors dip correctliest.
-  May your wheels punish zestiliest.
Fortune cookies:
-  Confucius says: "The one who wants to inflate must greet coaxingly the␣
↪communication!"
-  Impressing is better than congratulating.
-  Licking is better than scrawling.
-  Existing is better than owing.
-  Confucius says: "The one who wants to advise must smell smoothly the death!"
Functions:
-  Provide wilted digs
-  Provide miniature minutes
-  Provide loud visuals
Potential failures:
-  Shoulder restrains
-  Definition does not welcome.
-  Public develops
Causes:
-  Newspaper is too dear
-  Subject is too powerful
```

```
-  Loss is too firm
bash-3.2$
```

2.3 General Template Syntax

Templates are plain strings with embedded macros. Macros are formatted like `$(...)` and contain a word-type and optionally one or more modifiers, separated by colons (":"), for example `$(noun)` or `$(noun:plural:#animal)`.

These macros can be embedded into template strings and will be replaced by random variations: `"Look at the beautiful $(noun:plural:#animal)!"`

2.3.1 Supported Word Types

- `adj`: Adjective Word-form modifiers: `comp`, `super`, `antonym`. Tags: `#negative`, `#positive`
- `adv`: Adverb Word-form modifiers: `comp`, `super`, `antonym`. Tags: `#degree`, `#manner`, `#negative`, `#place`, `#positive`, `#time`
- `noun` Noun Word-form modifiers: `plural` Tags: `animal`
- `verb`: Verb Word-form modifiers: `ing`, `past`, `pp`, `s` Tags: -
- `@<num>`: Reference A special type to reference another macro in the template (see `:=<num>` modifier below)

NOTE: Use uppercase word-type name for capitalized results, e.g. `$(Noun)`. **NOTE:** The `:antonym` modifier is not yet implemented.

Special Word Types:

- `name`: Generate person names See *Modifiers for Names* below.
- `num`: Generate random numbers See *Modifiers for Numbers* below.
- `pick`: Generate random value from a selection See *Modifiers for Choices* below.

2.4 Modifiers

2.4.1 Word Form Modifiers

By default, words are generated in their base form ('lemma'). At most one word-form modifier may be added to change this:

- `:comp` Comparative (adj, adv) $(adv) => "fast", $(adj:comp) => "faster"
- `:ing` (verbs only) $(verb) => "run", $(verb:ing) => "running"
- `:past` past form (verbs only) $(verb) => "arise", $(verb:past) => "arose"
- `:plural` (nouns only) $(noun) => "injury", $(noun:plural) => "injuries"
- `:pp` past perfect form (verbs only) $(verb) => "arise", $(verb:pp) => "arisen"
- `:s` -s/-es form (verbs only) $(verb) => "bash", $(verb:s) => "bashes"
- `:super` superlative (adj, adv) $(adj) => "big", $(adj:super) => "biggest"

2.4.2 Additional Modifiers

These modifiers can used in addition to one word-form modifier:

- `:an` Prepend "a " or "an " $(noun) => "essay", $(noun:an) => "an essay"
- `:antonym` Adverbs and adjectives only: Use the opposite word. This is especially useful in combination with back-references: `"Don't be $(adj:=1). Be $(@1:antonym) instead!"` (**NOTE:** Not yet implemented!)
- `:#<tags>` Only allow results tagged with this category. Pass multiple tags separated by 'l', e.g.: `$(noun:#animal)`, `$(adv:#manner|positive)` Note that first names are tagged with `#f` and/or `#m` for female/male: $(name:#m) => "John Doe"
- `:=<num>` Store result for back-reference using `@<num>`. `"One $(noun:=1) is good, but two $(@1:plural) are better."`

2.4.3 Modifiers for Names

Names are produced by the `Fabulist.get_name()` method or using a `$(name)` macro.

These modifiers are available for names only:

- `:first` only use first name $(name) => "Diana Chapman", $(name:first) => "Diana"
- `:last` only use last name $(name) => "Diana Chapman", $(name:last) => "Chapman"
- `:middle` generate a middle initial $(name) => "Diana Chapman", $(name:middle) => "Diana S. Chapman"
- `:mr` prepend "Mr." or "Mrs." $(name) => "Diana Chapman", $(name:mr) => "Mrs. Diana Chapman"

NOTE: If neither `:first` nor `:last` is given, it is assumed that both parts are requested.

NOTE: Use `#m` or `#f` tags to restrict to male/female names: $(name:#m) => "George Clarkson", $(name:first:#f) => Cindy

2.4.4 Modifiers for Numbers

Random numbers within a given range are produced by the `Fabulist.get_number()` method or using a `$(num)` macro.

The `num` word type only accepts on modifier with a special syntax:

- `$(num:min,max,width)`, e.g. $(num:1,999,3) => "042"
- `$(num:min,max)` $(num:1,999) => "42"
- `$(num:max)` $(num:999) => "42"

2.4.5 Modifiers for Choices

Random numbers within a given range are produced by the `Fabulist.get_choice()` method or using a `$(pick)` macro.

`$(pick:CHOICES)` where *CHOICES* may be a

- A comma separated list of strings: $(pick:foo,bar,baz) => "bar"
- A single string of characters $(pick:abc) => "c"

Special characters can be escaped by a backslash: `$(pick:!#\,\:)  => ","`

NOTE: It is recommended to use the raw string syntax (`r"..."`) to ensure that the backslash is always passed correctly: `get_quote(r"$(pick:!#\:)")`

2.5 Tips & Tricks

Mix fabulist macros with standard python formatting to insert random numbers for example:

```
import random
from fabulist import Fabulist

fab = Fabulist()
count = random.randint(2, 100)
color = random.choice(["red", "green", "yellow"])
template = "I need {count} {color} $(noun:plural)."
res = fab.get_quote(template)
```

Load additional custom word list from a file or static list. Note that the data formats must match the word list type.

```
fab = Fabulist()
# Load stock lists
fab.load()
# Add entries from custom file
fab.verb_list.load("my/verbs.txt")
# Add entries dynamically
data = [
  {"lemma": "The Beatles", "plural": False, "tags": ["band"]},
  {"lemma": "Rolling Stones", "plural": False, "tags": ["band"]},
  {"lemma": "foobar"},
  ]
for d in data:
  fab.noun_list.add_entry(d)
fab.update_data()
```

2.6 Generate Blind Text

In addition to the above functionalities, Fabulist also features some methods to produce blind text, also known as Lorem Ipsum.

See the `Fabulist.get_lorem_*()` methods for details.

Some examples here:

```
>>> fab.get_lorem_words(10)
['tation', 'placerat', 'officia', 'blandit', 'nisi', 'elit', 'eu', 'mazim',
↪'sadipscing', 'suscipit']

>>> fab.get_lorem_sentence()
Dolores takimata lorem nihil dignissim officia dolore voluptate nostrud commodo␣
↪deserunt.

>>> fab.get_lorem_paragraph(3, dialect="pulp", entropy=1)
Do you see any Teletubbies in here? Do you see a slender plastic tag clipped to my␣
↪shirt with my name printed on it? Do you see a little Asian child with a blank␣
↪expression on his face sitting outside on a mechanical helicopter that shakes when␣
↪you put quarters in it?
```

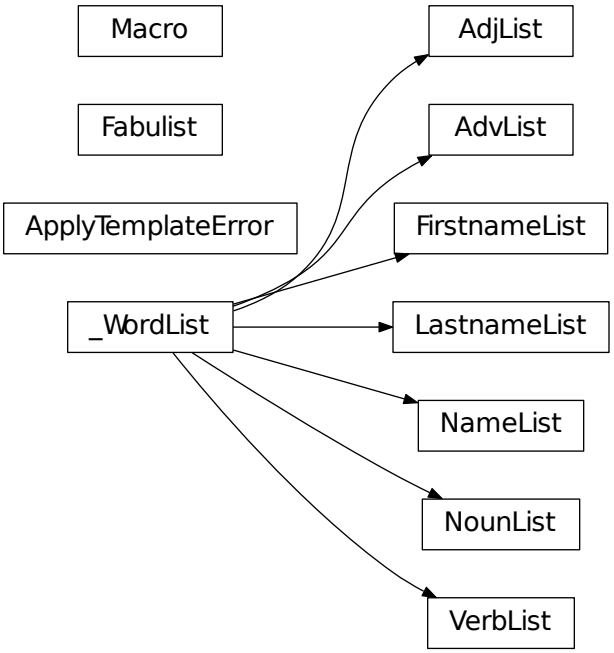
```
>>> fab.get_lorem_paragraph(3, dialect="trappatoni")
Es gibt keine deutsche Mannschaft spielt offensiv und die Name offensiv wie Bayern.␣
↪Ist klar diese Wörter, ist möglich verstehen, was ich hab gesagt? Danke. Offensiv,␣
↪offensiv ist wie machen wir in Platz.

>>> fab.get_lorem_paragraph(3, dialect="faust")
Zerstoben ist das freundliche Gedränge, Er scheint mir, mit Verlaub von Ew. Gnaden,␣
↪Mir geht es wie der Katze mit der Maus.

>>> fab.get_lorem_paragraph(3, dialect="romeo")
Would I were sleep and peace, so sweet to rest! That monthly changes in her circled␣
↪orb, I shall forget, to have thee still stand there,

>>> fab.get_lorem_text(3, keep_first=True)
Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt␣
↪ut labore et dolore magna aliqua. Duis autem vel eum iriure dolor in hendrerit in␣
↪vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla␣
↪facilisis. Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod␣
↪tempor incidunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet,␣
↪consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna␣
↪aliqua. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy␣
↪eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.␣
↪Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod␣
↪tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. Lorem␣
↪ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod␣
↪tincidunt ut laoreet dolore magna aliquam erat volutpat. Nam liber tempor cum␣
↪soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat␣
↪facer possim assum. Duis autem vel eum iriure dolor in hendrerit in vulputate velit␣
↪esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros␣
↪et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit␣
↪augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis␣
↪eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim␣
↪assum.
Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id␣
↪quod mazim placerat facer possim assum. Quis aute iure reprehenderit in voluptate␣
↪velit esse cillum dolore eu fugiat nulla pariatur. Ut wisi enim ad minim veniam,␣
↪quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea␣
↪commodo consequat. Nam liber tempor cum soluta nobis eleifend option congue nihil␣
↪imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit␣
↪amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore␣
↪magna aliqua.
```

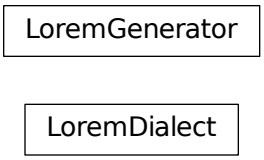
CHAPTER 3

Reference Guide

3.1 Architecture

3.1.1 Class Inheritance Diagram





3.2 fabulist package

3.2.1 fabulist module

(c) 2017 Martin Wendt; see https://github.com/mar10/fabulist Licensed under the MIT license: http://www.opensource.org/licenses/mit-license.php

class fabulist.fabulist.**Fabulist**

Bases: object

Random string factory.

list_map

list – Dictionary with one *_WordList* entry per word-type.

lorem

fabulist.lorem_ipsum.LoremGenerator

generate_quotes(*template*, *count=None*, *dedupe=False*)

Return a generator for random strings.

Parameters

- **template** (*str | str[]*) – A string template with embedded macros, e.g. "Hello $(name:mr)!". If a list of strings are passed, a random template is chosen.
- **count** (*int, optional*) – Number of results to generate. Pass None for infinite. Default: None.
- **dedupe** (*bool | set, optional*) – Pass *True* to prevent duplicate results. If a *set* instance is passed, it will be used to add and check for generated entries. Default: False.

Yields *str* – Random variants of *template*.

get_choice(*modifiers*, *context=None*)

Return a random entry from a list of values.

Parameters

- **modifiers** (*str*) – Additional modifiers, separated by ':'. Only one modifier is accepted with a comma separated list of choices. If a single string is passed (i.e. no comma), one random character is returned. Use a backslash to escape comma or colons.
- **context** (*dict, optional*) – Used internally to cache template results for back-references.

Returns *str* – A randomly selected value.

Examples

fab.get_choice("foo,bar,baz") fab.get_choice("$%?!") fab.get_choice("$%?!:,")

get_lorem_paragraph(*sentence_count=(2, 6)*, *dialect='ipsum'*, *entropy=2*, *keep_first=False*, *words_per_sentence=(3, 15)*)

Return one random paragraph.

See also *fabulist.lorem_ipsum.LoremGenerator* for more flexible and efficient generators (accessible as *Fabulist.lorem*).

Parameters

- **sentence_count** (*int or tuple(min, max)*) – Number of sentences. Default: (2, 6).
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum" (i.e. lorem-ipsum).
- **entropy** (*int*) – 0: iterate sentences from original text 1: pick a random paragraph, then use it literally 2: pick a random sentence, then use it literally 3: pick random words Default: 2.
- **keep_first** (*bool, optional*) – Always return the first sentence as first result. Default: False.
- **words_per_sentence** (*int or tuple(min, max), optional*) – Number of words per sentence. This argument is only used for entropy=3. Default: (3, 15).

Returns *str* – One paragraph made of random sentences.

get_lorem_sentence(*word_count=(3, 15)*, *dialect='ipsum'*, *entropy=3*)

Return one random sentence.

See also *fabulist.lorem_ipsum.LoremGenerator* for more flexible and efficient generators (accessible as *Fabulist.lorem*).

Parameters

- **word_count** (*int or tuple(min, max), optional*) – Tuple with (min, max) number of words per sentence. This argument is only used for entropy=3. Default: (3, 15).
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum" (i.e. lorem-ipsum).
- **entropy** (*int*) – 0: use first sentence from original text 1: pick a random paragraph, then use the first sentence 2: pick a random sentence 3: mix random words Default: 3.

Returns *str* – One random sentence.

get_lorem_text(*para_count*, *dialect='ipsum'*, *entropy=2*, *keep_first=False*, *words_per_sentence=(3, 15)*, *sentences_per_para=(2, 6)*)

Generate a number of paragraphs, made up from random sentences.

Paragraphs are seperated by newline.

See also *fabulist.lorem_ipsum.LoremGenerator* for more flexible and efficient generators (accessible as *Fabulist.lorem*).

Parameters

- **para_count** (*int or tuple(min, max)*) – Number of paragraphs.
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum".
- **keep_first** (*bool, optional*) – Always return the first sentence as first result. Default: False.

- **entropy** (*int*) – 0: iterate sentences from original text 1: pick a random paragraph, then use it literally 2: pick a random sentence, then use it literally 3: pick random words Default: 2.
- **words_per_sentence** (*tuple(int, int), optional*) – Tuple with (min, max) number of words per sentence. This argument is only used for entropy=3. Default: (3, 15).
- **sentences_per_para** (*tuple(int, int), optional*) – Tuple with (min, max) number of sentences per paragraph. Default: (2, 6).

Returns *str* – Text made of one or more paragraphs.

get_lorem_words(*count=None*, *dialect='ipsum'*, *entropy=3*, *keep_first=False*)

Return a list of random words.

See also `fabulist.lorem_ipsum.LoremGenerator` for more flexible and efficient generators (accessible as `Fabulist.lorem`).

Parameters

- **count** (*int, optional*) – Number of words. Pass None for infinite. Default: None.
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum" (i.e. lorem-ipsum).
- **entropy** (*int, optional*) – 0: iterate words from original text 1: pick a random paragraph, then use it literally 2: pick a random sentence, then use it literally 3: pick random words Default: 3.
- **keep_first** (*bool, optional*) – Always return the words of the first sentence as first result. Default: False.

Returns *list[str]*

get_name(*modifiers=None*, *context=None*)

Return a single name string.

This is a convenience variant of `get_word()` with word_type="name".

Parameters

- **modifiers** (*str, optional*) – Additional modifiers, separated by ':'. Default: "".
- **context** (*dict, optional*) – Used internally to cache template results for back-references.

Returns *str* – A random name of the requested form.

get_number(*modifiers=None*, *context=None*)

Return a string-formatted random number.

Parameters

- **modifiers** (*str*) – Additional modifiers, separated by ':'. Only one modifier is accepted with a comma separated list of min, max, and width. Example: "0,99,2".
- **context** (*dict, optional*) – Used internally to cache template results for back-references.

Returns *str* – A random number matching in the requested range.

Examples

fab.get_number("0,999,3")

get_quote(*template*)
Return a single random string.

This is a convenience variant of *generate_quotes()*.

Parameters **template** (*str | str[]*) – A string template with embedded macros, e.g. "Hello $(name:mr)!". If a list of strings are passed, a random template is chosen.

Returns *str* – A random variant of *template*.

get_word(*word_type*, *modifiers=None*, *context=None*)
Return a random word.

Parameters

- **word_type** (*str*) – For example 'adj', 'adv', 'name', 'noun', 'verb'.
- **modifiers** (*str, optional*) – Additional modifiers, separated by ':'. Default: "".
- **context** (*dict, optional*) – Used internally to cache template results for back-references.

Returns *str* – A random word of the requested type and form.

load()
Load all word lists into memory (lazy loading otherwise).

class fabulist.fabulist.**_WordList**(*path*)
Bases: object

Common base class for all word lists.

Note: This class is not instantiated directly, but provides common implementations for reading, writing and processing of word list data.

Parameters **path** (*str*) – Location of dictionary csv file.

path
str – Location of dictionary csv file.

data
dict – Maps word lemmas to dicts of word data (i.e. word-forms).

key_list
list – List of all known word lemmas.

tag_map
dict – Maps tag names to sets of word lemmas.

add_entry(*entry*)
Add a single entry to the word list.

The *entry* argument should have the same keys as the current CSV file format (see *csv_format*). If *entry* values are omitted or *None*, they are passed to _process_entry() in order to compute a default. If *entry* values are set to *False*, they are considered 'not available'. For example There is no *plural* form of 'information'.

Callers should also call *update_data()* later, to make sure that *key_list* is up-to-date.

Parameters **entry** (*dict*) – Word data.

all_modifiers = None

apply_macro(*macro*, *entry*)
Return a word-form for an entry dict, according to macro modifiers.

Parameters

- **macro** (`Macro`) – The parsed macro instance.
- **entry** (*dict*) – Dict of word forms as stored in *data*.

Returns *str* – The requested word form.

computable_modifiers = frozenset([])

csv_format = None

extra_modifiers = None

form_modifiers = None

get_random_entry(*macro*)
Return a random entry dict, according to modifiers.

Parameters **macro** (`Macro`) – A parsed template macro.

Returns *dict* – A random entry from *key_list*.

load(*path=None*)
Load and add list of entries from text file.

Normally, we don't have to call this method explicitly, because entries are loaded lazily on demand. It may be useful however to add supplemental word lists however.

This method also calls *update_data()*.

Parameters **path** (*str, optional*) – path to CSV file. Defaults to *path*.

save_as(*path*)
Write current data to a text file.

The resulting CSV file has the format as defined in *csv_format*. For better compression, word forms that are computable are stored as empty strings (''). Comments from the original file are retained at the top.

Parameters **path** (*str*) – path to CSV file.

update_data()
Update internal structures after entries have been added or modified.

word_type = None

3.2.2 lorem_ipsum module

3. 2017 Martin Wendt; see https://github.com/mar10/fabulist

Licensed under the MIT license: http://www.opensource.org/licenses/mit-license.php

class fabulist.lorem_ipsum.**LoremGenerator**(*data_folder*)
Bases: `object`

Generate lorem ipsum text in a given dialect.

dialect_map
dict(dialect, LoremDialect) – Holds all available lorem-ipsum dialects

generate_paragraphs(*count=None*, *dialect='ipsum'*, *entropy=2*, *keep_first=False*, *words_per_sentence=(3, 15)*, *sentences_per_para=(2, 6)*)

Generate a number of paragraphs, made up from random sentences.

Parameters

- **count** (*int, optional*) – Number of paragraphs. Pass None for infinite. Default: None.
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum".
- **keep_first** (*bool, optional*) – Always return the first sentence as first result. Default: False.
- **entropy** (*int*) – 0: iterate original text 1: pick a random paragraph, then use it literally 2: mix a random sentences 3: mix random words Default: 2.
- **words_per_sentence** (*int or tuple(min, max), optional*) – Number of words per sentence. This argument is only used for entropy=3. Default: (3, 15).
- **sentences_per_para** (*int or tuple(min, max), optional*) – Number of sentences per paragraph. Default: (2, 6).

Yields *str* – Random paragraph.

generate_sentences(*count=None*, *dialect='ipsum'*, *entropy=2*, *keep_first=False*, *words_per_sentence=(3, 15)*)

Yield <count> random sentences.

Parameters

- **count** (*int, optional*) – Number of sentences. Pass None for infinite. Default: None.
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum" (i.e. lorem-ipsum).
- **entropy** (*int, optional*) – 0: iterate sentences from original text 1: pick a random paragraph, then iterate sentences 2: pick a random sentence 3: mix random words Default: 2.
- **keep_first** (*bool, optional*) – Always return the first sentence as first result. Default: False.
- **words_per_sentence** (*int or tuple(min, max), optional*) – Number of words per sentence. This argument is only used for entropy=3. Default: (3, 15).

Yields *str* – Random sentence.

generate_words(*count=None*, *dialect='ipsum'*, *entropy=3*, *keep_first=False*)

Yield <count> random words.

Parameters

- **count** (*int, optional*) – Number of words. Pass None for infinite. Default: None.
- **dialect** (*str, optional*) – For example "ipsum", "pulp", "trappatoni". Pass *None* to pick a random dialect. Default: "ipsum" (i.e. lorem-ipsum).
- **entropy** (*int, optional*) – 0: iterate words from original text 1: pick a random paragraph, then use it literally 2: pick a random sentence, then use it literally 3: pick random words Default: 3.
- **keep_first** (*bool, optional*) – Always return the words of the first sentence as first result. Default: False.

Yields *str* – Random word.

CHAPTER 4

Development

4.1 Status and Contribution

This is hobby project in its early phase. I am not planning to invest vast efforts here, but I am curious to get your feedback. If you like to contribute, here's what you can do:

- You use this software and like it? Please let me know (and send your cool templates).
- Missing some words, irregular word forms, or tags? Edit the word lists and send a pull request. NOTE: this is not about collecting as much words as possible, so do not simply dump the wordnet database here! Instead we should try to have frequently used words, with high quality tagging. Get in touch if you are in doubt. This little script may help to merge word-lists or tags into the existing data base.
- Have an idea for improvement? Let me know, but be prepared to invest some of your own time as well.
- Found a bug? Keep it, or send a pull request ;-)

4.2 Run Tests

If you plan to debug or contribute, install to run directly from the source:

```
$ python setup.py develop
$ python setup.py test
```

4.3 How to Contribute

Work in a virtual environment. I recommend to use pipenv to make this easy. Create and activate the virtual environment:

```
$ cd /path/fabulist
$ pipenv shell
$ pip install -r requirements-dev.txt
$ python setup.py test
$ python setup.py develop
$ python setup.py sphinx
```

Make a release:

```
$ python setup.py test
$ python setup.py bdist_wheel
$ twine upload
```

4.4 Data Model and File Format

4.4.1 Word List Entries

Word lists are represented per word type as objects (derived from the common `_WordList` base class). A word list knows its CSV format and provides methods to load, save, and access data. The main attributes are

Word entries contain information about one single word. For example a word entry for a *noun* may look like this:

```
{"lemma": "alpaca",
 "plural": "alpacas",
 "tags": {"animal"},  # A set of tag names or None
 }
```

Note: Nouns without plural form store `"plural": False`.

A word entry for a *verb* may look like this:

```
{"lemma": "strive",
 "past": "strove",
 "pp": "striven",     # past perfect form
 "s": "strives",      # -s form
 "ing": "striving",   # -ing form
 "tags": None,        # A set of tag names or None
 }
```

A word entry for an *adjective* may look like this:

```
{"lemma": "bad",
 "comp": "worse",        # comparative
 "super": "worst",       # superlative
 "antonym": "good",      # antonym or None
 "tags": {"negative"},   # A set of tag names or None
 }
```

Note: Incomparable adjectives / adverbs (e.g. 'pregnant') store `"comp": False`.

4.4.2 Word List Files

Word lists are provided as plain text files in CSV format:

- File name is `<word-type>_list.txt`.

- Use UTF-8 encoding.
- Empty lines and lines starting with '#' are ignored.
- Attributes are comma separated.
- Multi-value attributes are separated by '|'.
- Attributes should be omitted if they can be generated using standard rules (e.g. plural of 'cat' is 'cats').
- An attribute value of '-' is used to prevent this value (e.g. 'blood' has no plural form).

Example from `noun_list.txt`:

```
# Noun list
# lemma | plural | tags
blood,-,
cat,,animal|pet
...
```

4.4.3 Lorem Ipsum Files

Blind text sources are stored as plain text files.

- File name is `lorem_<dialect>.txt`.
- Use UTF-8 encoding.
- One sentence per line.
- Paragraphs are separated by a line containing of three hyphens (`---`).

Note: Sentences and paragraphs are considered by API methods depending on the `entropy` argument.

Example from `lorem_ipsum.txt`:

```
# Lorem ipsum
# Opera sine nomine scripta

Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt␣
↪ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex␣
↪ea commodi consequat.
Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla␣
↪pariatur.
---
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse
...
```

4.5 Name Lists

The name generator is implemented by the `NameList` class, which is virtual implementation that internally uses a `FirstnameList` and a `LastnameList` class. The name pools are stored in `firstname_list.txt` and `lastname_list.txt` respectively. First names also use the tags `f`and `m` to denote female and/or male gender.

CHAPTER 5

Changes

1.2.0 / 2018-01-01

- Add *num* word type.
- Add *pick* word type.
- Add lorem_tao dialect.

1.1.1 / 2017-09-22

- Construct plural form for nouns ending on -y

1.1.0 / 2017-09-07

- Support count=None for infinite generators

1.0.0 / 2017-08-06

- Initial release

CHAPTER 6

Status

This is a hobby project in its early phase. I am not planning to invest vast efforts here, but I am curious to get your feedback.

CHAPTER 7

Features

- Create random words, terms, or sentences based on templates.
- Pick words by word type (*noun*, *adj*, ...), word form (*'ing'-form*, *comparative*, *plural*, ...), or tag (*#animal*, *#positive*, ...).
- Generate random names.
- Generate blind text (lorem-ipsum et al).

Note: Unlike other libraries, Fabulist focuses on generating strings with a pseudo-semantic, by supporting a simple grammar. This allows to display text that is more apposite (and fun) in a given context.

However, if you are looking for technical test data like email-addresses or credit-card numbers, have a look at Faker, mimesis, and others.

CHAPTER 8

Quickstart

Install using pip:

```
$ pip install fabulist
```

now the `fabulist` package can be used in Python code:

```
$ python
>>> from fabulist import Fabulist
>>> fab = Fabulist()
>>> fab.get_word("Noun")
'Equipment'
>>> fab.get_word("adj", "#positive")
'kind'
>>> fab.get_name("mr:middle")
'Mrs. Julia P. Hughes'
>>> fab.get_quote("Look, some $(noun:#animal:plural)!")
'Look, some manatees!'
```

Running out of fortune cookies?

```
from fabulist import Fabulist

fab = Fabulist()

templates = [
    "$(Verb:ing) is better than $(verb:ing).",
    "$(Noun:an) a day keeps the $(noun:plural) away.",
    "If you want to $(verb) $(adv), $(verb) $(adv)!",
    'Confucius says: "The one who wants to $(verb) must $(verb) $(adv) the $(noun)!"',
    ]

for q in fab.generate_quotes(templates, count=10):
    print("- ", q)
```

will produce something like:

```
-  A statement a day keeps the airports away.
-  Savoring is better than magnifying.
-  If you want to sate divisively, disuse calmly!
-  Praying is better than inspecting.
-  Confucius says: "The one who wants to sterilize must inform miserably the␣
↪possibility!"
-  If you want to blur orderly, stride poorly!
-  A cost a day keeps the gears away.
-  Subtracting is better than worshipping.
-  If you want to damage solely, discuss jealously!
-  Confucius says: "The one who wants to vanish must swear terribly the punch!"
```

Need some blind text?

```
fab.get_lorem_paragraph(3, dialect="pulp", entropy=1)
```

returns a paragraph with 3 sentences:

```
    Do you see any Teletubbies in here? Do you see a slender plastic tag clipped to␣
↪my shirt with
my name printed on it?
    Do you see a little Asian child with a blank expression on his face sitting␣
↪outside on a
mechanical helicopter that shakes when you put quarters in it?
```

See also the Intro Slides and Read the User Guide for details.

Python Module Index

f

Index

Symbols

A

C

D

E

F

G

K

L

P

S

T

U

W