Python

Release

October 08, 2016

Contents

version 1.0.5

author Luca De Vitis <luca@monkeython.com>

contact http://www.monkeython.com

CHAPTER 1

Overview

The purpose of this package is to generate random (plausible) text sentences and paargraphs based on a dictionary and a sample text. By default this package will generate Lorem Ipsum style text, but you can customize the generator to effectively load any dictionary and any sample text you like.

This package has en extensive docstring documentation, so you can read more on the online documentation or in the python interactive shell as well.

CHAPTER 2

History

Most of the code of this module is taken from lorem-ipsum-generator by James Hales. James stated that his package served his purpose and he was not interested in further development, so I took it over.

CHAPTER 3

Changes

1.0.4

- Added MANIFEST.in
- Removed dependencies on distribute
- Applied pep8 and pylint suggested cleanup

1.0.3

- Fix issue #5

1.0.2

- Now is a package: fixes datafiles distribution.

1.0.1

- Added support for python 2.5

1.0.0

- Added unittests.
- Added documentation.
- Added stats to text generators methods in Generator
- Added generator methods in Generator, for multiple text generations
- Added stats-less text generators fuctions to module

0.1.0

- First release.

CHAPTER 4

API

4.1 Basic usage

This package provides a text generator class and some utility functions that can simply return the text you desire. There are 2 sets of functions:

- Those with **generate_** prefix that return the desired text and some stats
- Those with **get_** that return the desired text without the stats

On the average, you probably want to import the **get_** prefixed functions and just get the text:

```
>>> from loremipsum import get_sentences
>>>
>>> sentences_list = get_sentences(5)
>>> len(sentences_list)
5
>>>
```

If you fancy some statistics, you want to import the **generate_** prefixed functions:

```
>>> from loremipsum import generate_paragraph
>>>
>>> sentences_count, words_count, paragraph = generate_paragraph()
```

If you need generate text based on your own sample text and/or dictionary, you want to import the **Generator** class:

```
>>> from loremipsum import Generator
>>>
>>> with open('data/sample.txt', 'r') as sample_txt
>>>     sample = sample_txt.read()
>>> with open('data/dictionary.txt', 'r') as dictionary_txt
>>>     dictionary = dictionary_txt.read().split()
>>>
>>> g = Generator(sample, dictionary)
>>> sentence = g.get_sentence()
>>>
```

4.2 Exceptions

exception `SampleError`

The sample text must contain one or more empty-line delimited paragraphs, and each paragraph must contain

one or more period, question mark, or exclamation mark delimited sentences.

exception `DictionaryError`
The dictionary must be a list of one or more words.

4.3 Functions

`generate_sentence`(*start_with_lorem=False*)
Utility function to generate a single random sentence with stats.

Parameters `start_with_lorem` (*bool*) – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

Returns a tuple with amount of sentences, words and the text

Return type tuple(int, int, str)

`generate_sentences`(*amount*, *start_with_lorem=False*)
Generator function that yields specified amount of random sentences with stats.

Parameters

- `start_with_lorem` – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.
- `amount` (*int*) – amount of sentences to generate.

Returns a tuple with amount of sentences, words and the text

Return type tuple(int, int, str)

`generate_paragraph`(*start_with_lorem=False*)
Utility function to generate a single random paragraph with stats.

Parameters `start_with_lorem` – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

Returns a tuple with amount of sentences, words and the text

Return type tuple(int, int, str)

`generate_paragraphs`(*amount*, *start_with_lorem=False*)
Generator function that yields specified amount of random paragraphs with stats.

Parameters `start_with_lorem` – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

Returns a tuple with amount of sentences, words and the text

Return type tuple(int, int, str)

`get_sentence`(*start_with_lorem=False*)
Utility function to get a single random sentence.

Parameters `start_with_lorem` – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

Returns a random sentence

Return type str

`get_sentences`(*amount*, *start_with_lorem=False*)
Utility function to get specified amount of random sentences.

Parameters

- **start_with_lorem** – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.
- **amount** (*int*) – amount of sentences to get.

Returns a list of random sentences.

Return type list

get_paragraph(*start_with_lorem=False*)
Utility function to get a single random paragraph.

Parameters **start_with_lorem** – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

Returns a random paragrpah

Return type str

get_paragraphs(*amount*, *start_with_lorem=False*)
Utility function to get specified amount of random paragraphs.

Parameters **start_with_lorem** – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

Returns a list of random paragraphs

Return type list

4.4 The Generator class

class Generator(*sample=None*, *dictionary=None*)
Generates random strings of "lorem ipsum" text.

Markov chains are used to generate the random text based on the analysis of a sample text. In the analysis, only paragraph, sentence and word lengths, and some basic punctuation matter – the actual words are ignored. A provided list of words is then used to generate the random text, so that it will have a similar distribution of paragraph, sentence and word lengths.

Parameters

- **sample** (*str*) – a string containing the sample text
- **dictionary** (*list*) – a string containing a list of words

generate_paragraph(*start_with_lorem=False*)
Generates a single lorem ipsum paragraph, of random length.

Parameters **start_with_lorem** (*bool*) – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

generate_paragraphs(*amount*, *start_with_lorem=False*)
Generator method that yields paragraphs, of random length.

Parameters **start_with_lorem** (*bool*) – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

generate_sentence(*start_with_lorem=False*)
Generates a single sentence, of random length.

Parameters **start_with_lorem** (*bool*) – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

generate_sentences(*amount*, *start_with_lorem=False*)
Generator method that yields sentences, of random length.

Parameters **start_with_lorem** (*bool*) – if True, then the text will begin with the standard "Lorem ipsum..." first sentence.

reset_statistics()
Resets the values of *sentence_mean*, *sentence_sigma*, *paragraph_mean*, and *paragraph_sigma* to their values as calculated from the sample text.

dictionary
A dictionary of words that generated sentences are made of, grouped by words length.

Parameters **words** (*list*) – list of words

Return type dict

Raises *DictionaryError* if no valid words in dictionary

paragraph_mean
A non-negative value determining the mean paragraph length (in sentences) of generated sentences. Is changed to match the sample text when the sample text is updated.

Return type int

Raises ValueError if value is lesser then 0

paragraph_sigma
A non-negative value determining the standard deviation of paragraph lengths (in sentences) of generated sentences. Is changed to match the sample text when the sample text is updated.

Return type int

Raises ValueError if value is lesser then 0

sample
The sample text that generated sentences are based on.

Sentences are generated so that they will have a similar distribution of word, sentence and paragraph lengths and punctuation.

Sample text should be a string consisting of a number of paragraphs, each separated by empty lines. Each paragraph should consist of a number of sentences, separated by periods, exclamation marks and/or question marks. Sentences consist of words, separated by white space.

Parameters **sample** (*str*) – the sample text

Return type str

Raises *SampleError* if no words in sample text

sentence_mean
A non-negative value determining the mean sentence length (in words) of generated sentences. Is changed to match the sample text when the sample text is updated.

Return type int

Raises ValueError if value is lesser then 0

sentence_sigma
A non-negative value determining the standard deviation of sentence lengths (in words) of generated sentences. Is changed to match the sample text when the sample text is updated.

Return type int

Raises `ValueError` if value is lesser then 0

words
The plain list of words in the dictionary.

4.5 Indices and tables

- genindex
- modindex
- search

l

D

G

L

P

R

S

W