
wordle

Release 0.2.1

Dominic Davis-Foster

Jan 14, 2021

CONTENTS

1	wordle	3
2	wordle.frequency	9
3	wordle.utils	11
4	Examples	13
4.1	Python Source File	13
4.2	C Source File	14
4.3	Folium git repository	14
5	Overview	17
6	Coding style	19
7	Automated tests	21
8	Type Annotations	23
9	Build documentation locally	25
10	Downloading source code	27
10.1	Building from source	27
	Python Module Index	29
	Index	31

Create a wordcloud for a Git repository.

Can also create wordclouds from directories of source files or a single source file.

Docs	
Tests	
PyPI	
Activity	
QA	
Other	

from PyPI

from GitHub

```
$ python3 -m pip install wordle --user
```

```
$ python3 -m pip install git+https://github.com/domdfcoding/wordle@master --user
```


Create wordclouds from git repositories, directories and source files.

Classes:

<code>Wordle([font_path, width, height, ...])</code>	Generate word clouds from source code.
--	--

Functions:

<code>export_wordcloud(word_cloud, outfile)</code>	Export a wordcloud to a file.
--	-------------------------------

```
class Wordle(font_path=None, width=400, height=200, prefer_horizontal=0.9, mask=None,
              contour_width=0, contour_color='black', scale=1, min_font_size=4, font_step=1,
              max_words=200, background_color='black', max_font_size=None, mode='RGB', rela-
              tive_scaling='auto', color_func=None, regexp=None, collocations=True, colormap=None,
              repeat=False, include_numbers=False, min_word_length=0, random_state=None)
```

Bases: WordCloud

Generate word clouds from source code.

Parameters

- **font_path** (`Optional[str]`) – Font path to the font that will be used (OTF or TTF). Defaults to DroidSansMono path on a Linux machine. If you are on another OS or don't have this font, you need to adjust this path. Default `None`.
- **width** (`int`) – The width of the canvas. Default 400.
- **height** (`int`) – The height of the canvas. Default 200.
- **prefer_horizontal** (`float`) – The ratio of times to try horizontal fitting as opposed to vertical. If `prefer_horizontal < 1`, the algorithm will try rotating the word if it doesn't fit. (There is currently no built-in way to get only vertical words.) Default 0.9.
- **mask** (`Optional[ndarray]`) – If not `None`, gives a binary mask on where to draw words. If mask is not `None`, width and height will be ignored and the shape of mask will be used instead. All white (`#FF` or `#FFFFFF`) entries will be considered “masked out” while other entries will be free to draw on. Default `None`.
- **contour_width** (`float`) – If mask is not `None` and `contour_width > 0`, draw the mask contour. Default 0.
- **contour_color** (`str`) – Mask contour color. Default `'black'`.
- **scale** (`float`) – Scaling between computation and drawing. For large word-cloud images, using scale instead of larger canvas size is significantly faster, but might lead to a coarser fit for the words. Default 1.

- **min_font_size** (`int`) – Smallest font size to use. Will stop when there is no more room in this size. Default 4.
- **font_step** (`int`) – Step size for the font. `font_step > 1` might speed up computation but give a worse fit. Default 1.
- **max_words** (`int`) – The maximum number of words. Default 200.
- **background_color** (`str`) – Background color for the word cloud image. Default 'black'.
- **max_font_size** (`Optional[int]`) – Maximum font size for the largest word. If `None` the height of the image is used. Default `None`.
- **mode** (`str`) – Transparent background will be generated when mode is “RGBA” and background_color is `None`. Default 'RGB'.
- **relative_scaling** (`Union[str, float]`) – Importance of relative word frequencies for font-size. With `relative_scaling=0`, only word-ranks are considered. With `relative_scaling=1`, a word that is twice as frequent will have twice the size. If you want to consider the word frequencies and not only their rank, `relative_scaling` around .5 often looks good. If 'auto' it will be set to 0.5 unless `repeat` is true, in which case it will be set to 0. Default 'auto'.
- **color_func** (`Optional[Callable]`) – Callable with parameters `word`, `font_size`, `position`, `orientation`, `font_path`, `random_state` which returns a PIL color for each word. Overwrites “colormap”. See `colormap` for specifying a matplotlib colormap instead. To create a word cloud with a single color, use `color_func=lambda *args, **kwargs: "white"`. The single color can also be specified using RGB code. For example `color_func=lambda *args, **kwargs: (255, 0, 0)` sets the color to red. Default `None`.
- **regex** (`Optional[str]`) – Regular expression to split the input text into tokens in `process_text`. If `None` is specified, `r"\w[\w']+"` is used. Ignored if using `generate_from_frequencies`. Default `None`.
- **collocations** (`bool`) – Whether to include collocations (bigrams) of two words. Ignored if using `generate_from_frequencies`. Default `True`.
- **colormap** (`Union[None, str, Colormap]`) – Matplotlib colormap to randomly draw colors from for each word. Ignored if “color_func” is specified. Default “viridis”.
- **repeat** (`bool`) – Whether to repeat words and phrases until `max_words` or `min_font_size` is reached. Default `False`.
- **include_numbers** (`bool`) – Whether to include numbers as phrases or not. Default `False`.
- **min_word_length** (`int`) – Minimum number of letters a word must have to be included. Default 0.
- **random_state** (`Union[RandomState, int, None]`) – Seed for the randomness that determines the colour and position of words. Default `None`.

Note: Larger canvases will make the code significantly slower. If you need a large word cloud, try a lower canvas size, and set the scale parameter. The algorithm might give more weight to the ranking of the words than their actual frequencies, depending on the `max_font_size` and the scaling heuristic.

Methods:

<code>__array__()</code>	Returns the wordcloud image as numpy array.
<code>generate_from_directory(directory[, ...])</code>	Create a word_cloud from a directory of source code files.
<code>generate_from_file(filename[, outfile, ...])</code>	Create a word_cloud from a source code file.
<code>generate_from_git(git_url[, outfile, sha, ...])</code>	Create a word_cloud from a directory of source code files.
<code>recolor([random_state, color_func, colormap])</code>	Recolour the existing layout.
<code>to_array()</code>	Returns the wordcloud image as numpy array.
<code>to_file(filename)</code>	Export the wordle to a file.
<code>to_image()</code>	Returns the wordcloud as an image.
<code>to_svg(*[, embed_font, ...])</code>	Export the wordle to an SVG.

Attributes:

<code>color_func</code>	Callable with parameters word, font_size, position, orientation, font_path, random_state which returns a PIL color for each word.
-------------------------	---

`__array__()`

Returns the wordcloud image as numpy array.

Return type `ndarray`

`color_func`

Type: `Callable`

Callable with parameters word, font_size, position, orientation, font_path, random_state which returns a PIL color for each word.

`generate_from_directory` (*directory*, *outfile=None*, *, *exclude_words=()*, *exclude_dirs=()*, *max_font_size=None*)

Create a word_cloud from a directory of source code files.

Parameters

- **`directory`** (`Union[str, Path, PathLike]`) – The directory to process
- **`outfile`** (`Union[str, Path, PathLike, None]`) – The file to save the wordle as. Supported formats are PNG, JPEG and SVG. If `None` the wordle is not saved. Default `None`.
- **`exclude_words`** (`Sequence[str]`) – An optional list of words to exclude. Default `()`.
- **`exclude_dirs`** (`Sequence[Union[str, Path, PathLike]]`) – An optional list of directories to exclude. Each entry is treated as a regular expression to match at the beginning of the relative path. Default `()`.
- **`max_font_size`** (`Optional[int]`) – Use this font-size instead of `max_font_size`. Default `None`.

Changed in version 0.2.1: `exclude_words`, `exclude_dirs`, `max_font_size` are now keyword-only.

Return type `Wordle`

`generate_from_file` (*filename*, *outfile=None*, *, *exclude_words=()*, *max_font_size=None*)

Create a word_cloud from a source code file.

Parameters

- **filename** (`Union[str, Path, PathLike]`) – The file to process
- **outfile** (`Union[str, Path, PathLike, None]`) – The file to save the wordle as. Supported formats are PNG, JPEG and SVG. If `None` the wordle is not saved. Default `None`.
- **exclude_words** (`Sequence[str]`) – An optional list of words to exclude. Default `()`.
- **max_font_size** (`Optional[int]`) – Use this font-size instead of `max_font_size`. Default `None`.

Changed in version 0.2.1: `exclude_words`, `max_font_size` are now keyword-only.

Return type *Wordle*

generate_from_git (`git_url`, `outfile=None`, `*`, `sha=None`, `depth=None`, `exclude_words=()`, `exclude_dirs=()`, `max_font_size=None`)

Create a word_cloud from a directory of source code files.

Parameters

- **git_url** (`str`) – The url of the git repository to process
- **outfile** (`Union[str, Path, PathLike, None]`) – The file to save the wordle as. Supported formats are PNG, JPEG and SVG. If `None` the wordle is not saved. Default `None`.
- **sha** (`Optional[str]`) – An optional SHA hash of a commit to checkout. Default `None`.
- **depth** (`Optional[int]`) – An optional depth to clone at. If `None` and `sha` is `None` the depth is 1. If `None` and `sha` is given the depth is unlimited. Default `None`.
- **exclude_words** (`Sequence[str]`) – An optional list of words to exclude. Default `()`.
- **exclude_dirs** (`Sequence[Union[str, Path, PathLike]]`) – An optional list of directories to exclude. Default `()`.
- **max_font_size** (`Optional[int]`) – Use this font-size instead of `self.max_font_size`. Default `None`.

Changed in version 0.2.1: `exclude_words`, `exclude_dirs`, `max_font_size` are now keyword-only. Added the `sha` and `depth` keyword-only arguments.

- **Return type** *Wordle*

recolor (`random_state=None`, `color_func=None`, `colormap=None`)

Recolour the existing layout.

Applying a new coloring is much faster than regenerating the whole wordle.

Parameters

- **random_state** (`Union[RandomState, int, None]`) – If not `None`, a fixed random state is used. If an `int` is given, this is used as seed for a `random.Random` state. Default `None`.
- **color_func** (`Optional[Callable]`) – Function to generate new color from word count, font size, position and orientation. If `None`, `color_func` is used. Default `None`.

- **colormap** (`Union[None, str, Colormap]`) – Use this colormap to generate new colors. Ignored if `color_func` is specified. If `None`, `color_func` or `color_map` is used. Default `None`.

Return type `Wordle`

Returns `self`

to_array ()

Returns the wordcloud image as numpy array.

to_file (*filename*)

Export the wordle to a file.

Parameters **filename** (`Union[str, Path, PathLike]`) – The file to save as.

Returns `self`

to_image ()

Returns the wordcloud as an image.

to_svg (*, *embed_font=False, optimize_embedded_font=True, embed_image=False*)

Export the wordle to an SVG.

Parameters

- **embed_font** (`bool`) – Whether to include font inside resulting SVG file. Default `False`.
- **optimize_embedded_font** (`bool`) – Whether to be aggressive when embedding a font, to reduce size. In particular, hinting tables are dropped, which may introduce slight changes to character shapes (w.r.t. *to_image* baseline). Default `True`.
- **embed_image** (`bool`) – Whether to include rasterized image inside resulting SVG file. Useful for debugging. Default `False`.

Return type `str`

Returns The content of the SVG image.

export_wordcloud (*word_cloud, outfile*)

Export a wordcloud to a file.

Parameters

- **word_cloud** (`WordCloud`)
- **outfile** (`Union[str, Path, PathLike]`) – The file to export the wordcloud to.

WORDLE . FREQUENCY

Functions to determine word token frequency for wordclouds.

New in version 0.2.0.

Functions:

<code>frequency_from_directory(directory[, ...])</code>	Returns a dictionary mapping the words in files in <code>directory</code> to their frequencies.
<code>frequency_from_file(filename[, exclude_words])</code>	ex- Returns a dictionary mapping the words in the file to their frequencies.
<code>frequency_from_git(git_url[, sha, depth, ...])</code>	Returns a dictionary mapping the words in files in <code>directory</code> to their frequencies.
<code>get_tokens(filename)</code>	Returns a <code>collections.Counter</code> of the tokens in a file.

frequency_from_directory (*directory*, *exclude_words*=(), *exclude_dirs*=())

Returns a dictionary mapping the words in files in `directory` to their frequencies.

Parameters

- **directory** (`Union[str, Path, PathLike]`) – The directory to process
- **exclude_words** (`Sequence[str]`) – An optional list of words to exclude. Default `()`.
- **exclude_dirs** (`Sequence[Union[str, Path, PathLike]]`) – An optional list of directories to exclude. Default `()`.

New in version 0.2.0.

Return type `Counter`

frequency_from_file (*filename*, *exclude_words*=())

Returns a dictionary mapping the words in the file to their frequencies.

Parameters

- **filename** (`Union[str, Path, PathLike]`) – The file to process
- **exclude_words** (`Sequence[str]`) – An optional list of words to exclude. Default `()`.

New in version 0.2.0.

See also:

func:~.get_tokens

Return type `Counter`

frequency_from_git (*git_url*, *sha=None*, *depth=None*, *exclude_words=()*, *exclude_dirs=()*)

Returns a dictionary mapping the words in files in *directory* to their frequencies.

Parameters

- **git_url** (*str*) – The url of the git repository to process
- **sha** (*Optional[str]*) – An optional SHA hash of a commit to checkout. Default *None*.
- **depth** (*Optional[int]*) – An optional depth to clone at. If *None* and *sha* is *None* the depth is 1. If *None* and *sha* is given the depth is unlimited. Default *None*.
- **exclude_words** (*Sequence[str]*) – An optional list of words to exclude. Default *()*.
- **exclude_dirs** (*Sequence[Union[str, Path, PathLike]]*) – An optional list of directories to exclude. Default *()*.

New in version 0.2.0.

Return type *Counter*

get_tokens (*filename*)

Returns a *collections.Counter* of the tokens in a file.

Parameters **filename** (*Union[str, Path, PathLike]*) – The file to parse.

Return type *Counter[str]*

Returns A count of words etc. in the file.

WORDLE . UTILS

Utility functions.

New in version 0.2.0.

Functions:

`clone_into_tmpdir`(`git_url`, `tmpdir`[, `sha`, `depth`]) Clone the git repository at `git_url` into `tmpdir`.

`clone_into_tmpdir` (`git_url`, `tmpdir`, `sha=None`, `depth=None`)

Clone the git repository at `git_url` into `tmpdir`.

Parameters

- **`git_url`** (`str`) – The url of the git repository to process
- **`tmpdir`** (`Union[str, Path, PathLike]`)
- **`sha`** (`Optional[str]`) – An optional SHA hash of a commit to checkout. Default `None`.
- **`depth`** (`Optional[int]`) – An optional depth to clone at. If `None` and `sha` is `None` the depth is 1. If `None` and `sha` is given the depth is unlimited. Default `None`.

New in version 0.2.0.

Return type `Path`

EXAMPLES

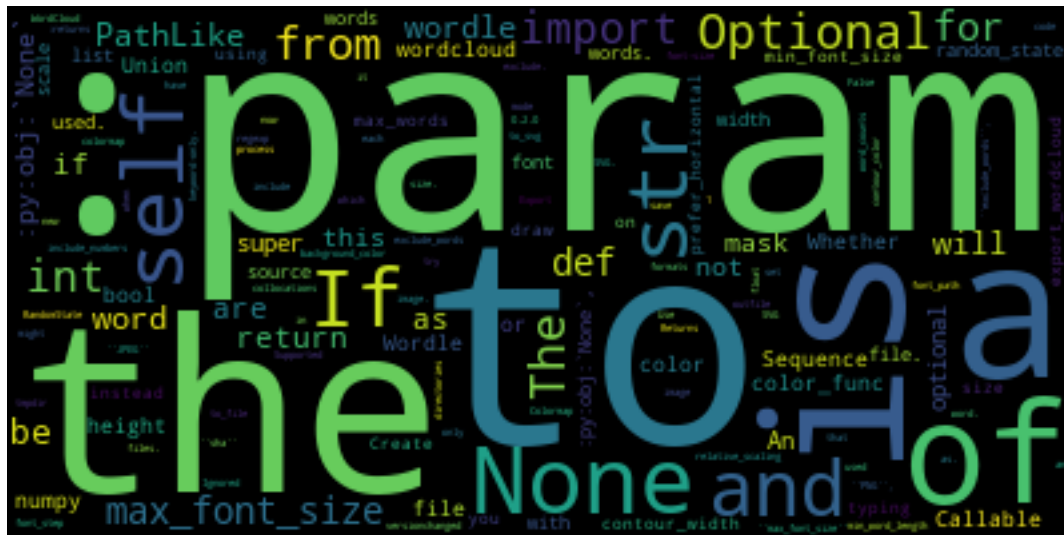
```
"""
Create a wordcloud from a single Python source file
"""

# stdlib
import pathlib

# this package
from wordle import Wordle, export_wordcloud

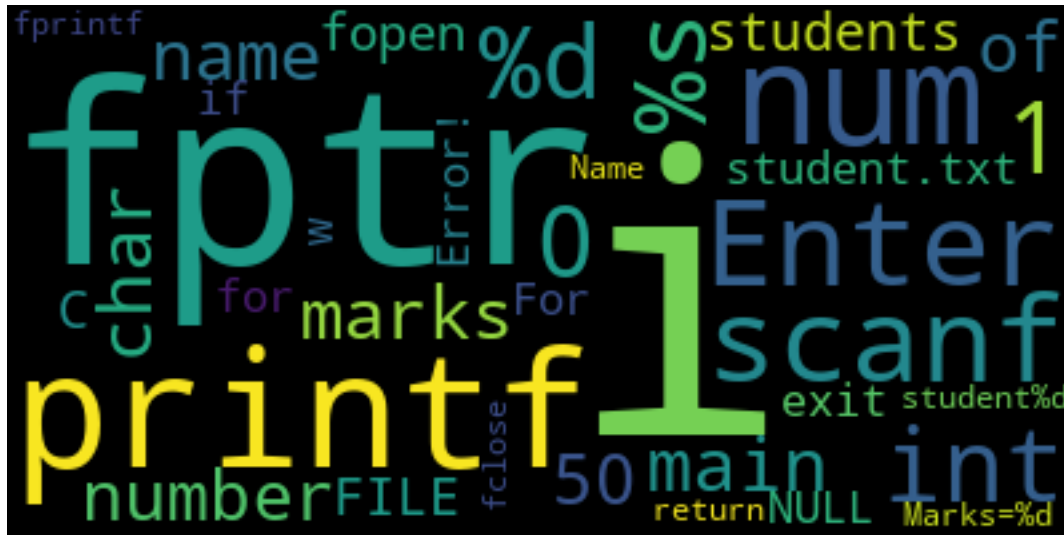
filename = pathlib.Path('.').absolute().parent / "wordle/__init__.py"

w = Wordle(random_state=5678)
w.generate_from_file(filename, outfile="python_wordcloud.svg")
export_wordcloud(w, outfile="python_wordcloud.png")
```



4.2 C Source File

```
1 """
2 Create a wordcloud from a single C source file
3 """
4
5 # this package
6 from wordle import Wordle, export_wordcloud
7
8 w = Wordle(random_state=5678)
9 w.generate_from_file("example.c", outfile="c_wordcloud.svg")
10 export_wordcloud(w, outfile="c_wordcloud.png")
```



4.3 Folium git repository

```
1 """
2 Create a wordcloud from the Folium git repository.
3
4 https://github.com/python-visualization/folium
5 """
6
7 # this package
8 from wordle import Wordle, export_wordcloud
9
10 w = Wordle(random_state=5678)
11 w.generate_from_git("https://github.com/python-visualization/folium", outfile="folium_
12 ↪wordcloud.svg")
13 export_wordcloud(w, outfile="folium_wordcloud.png")
```


OVERVIEW

wordle uses `tox` to automate testing and packaging, and `pre-commit` to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```


CODING STYLE

`yapf-isort` is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run yapf-isort -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```


AUTOMATED TESTS

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6, run:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```


TYPE ANNOTATIONS

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```


BUILD DOCUMENTATION LOCALLY

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```


DOWNLOADING SOURCE CODE

The `wordle` source code is available on GitHub, and can be accessed from the following URL: <https://github.com/domdfcoding/wordle>

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/wordle"  
> Cloning into 'wordle'...  
> remote: Enumerating objects: 47, done.  
> remote: Counting objects: 100% (47/47), done.  
> remote: Compressing objects: 100% (41/41), done.  
> remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126  
> Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.  
> Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a ‘zip’ file by clicking:

Clone or download → Download Zip

10.1 Building from source

The recommended way to build `wordle` is to use `tox`:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use `pep517.build` or another **PEP 517**-compatible build tool.

View the [Function Index](#) or browse the [Source Code](#).

[Browse the GitHub Repository](#)

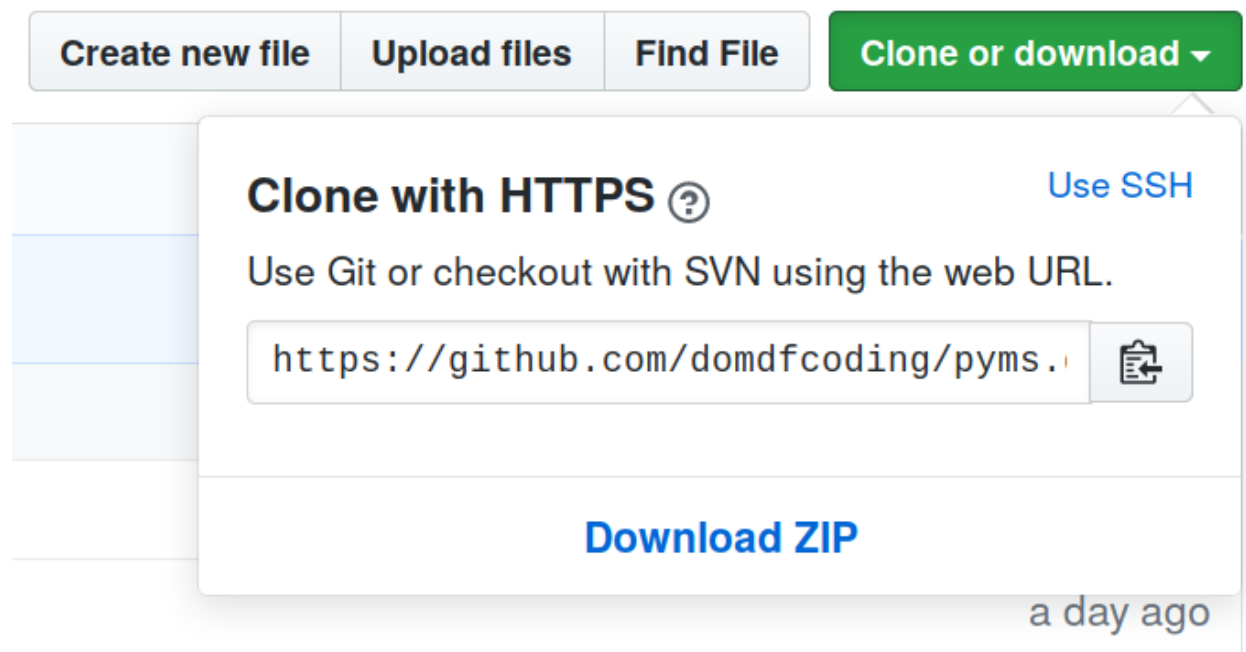


Fig. 1: Downloading a 'zip' file of the source code

PYTHON MODULE INDEX

W

`wordle`, [3](#)
`wordle.frequency`, [9](#)
`wordle.utils`, [11](#)

Symbols

`__array__()` (*Wordle method*), 5

C

`clone_into_tmpdir()` (*in module wordle.utils*), 11
`color_func` (*Wordle attribute*), 5

E

`export_wordcloud()` (*in module wordle*), 7

F

`frequency_from_directory()` (*in module wordle.frequency*), 9
`frequency_from_file()` (*in module wordle.frequency*), 9
`frequency_from_git()` (*in module wordle.frequency*), 9

G

`generate_from_directory()` (*Wordle method*), 5
`generate_from_file()` (*Wordle method*), 5
`generate_from_git()` (*Wordle method*), 6
`get_tokens()` (*in module wordle.frequency*), 10

M

module
 wordle, 3
 wordle.frequency, 9
 wordle.utils, 11

P

Python Enhancement Proposals
 PEP 517, 27

R

`recolor()` (*Wordle method*), 6

T

`to_array()` (*Wordle method*), 7
`to_file()` (*Wordle method*), 7
`to_image()` (*Wordle method*), 7

`to_svg()` (*Wordle method*), 7

W

wordle
 module, 3
`Wordle` (*class in wordle*), 3
`wordle.frequency`
 module, 9
`wordle.utils`
 module, 11