randword

Release 2.11.1

Dec 16, 2022

Contents:

1 Usage **1**

2 *randword* Package **5**

2.1 `randword` Package 5

2.2 `rand_word` Module 5

2.3 `rand_name` Module 6

2.4 `rand_place` Module 6

2.5 `rand_letter` Module 7

2.6 `rand_other` Module 7

3 License **9**

4 Indices and tables **11**

Python Module Index **13**

Index **15**

CHAPTER 1

Usage

Random words:

```
>>> from randword import word

>>> word()
'concession'

>>> word(include_pos=['adj'])
'accentual'

>>> word(include_pos=['adj', 'verb'])
'immaterialize'

>>> word(exclude_pos=['adj', 'adv', 'noun', 'pron', 'verb'])
'even if'

>>> word(min_word_len=20)
'magnetic line of force'

>>> word(max_word_len=3)
'use'

>>> word(min_word_len=4, max_word_len=5)
'Sepia'

>>> word(word_len=5)
'buggy'

>>> word(starts_with="ly")
'lymphopoiesis'

>>> word(ends_with="en")
'ten'
```

(continues on next page)

(continued from previous page)

```
>>> word(starts_with="un", ends_with="e")
'untouchable'

>>> word(pattern="ten")
'finiteness'

>>> word(starts_with="e", ends_with="n", pattern="non")
'enigma canon'

>>> word(count=3)
['Mozambican', 'demythologization', 'incontestable']

>>> word(3, include_pos=['adj'])
['discriminable', 'excrescent', 'noncivilized']

>>> word(3, ['adj', 'verb'])
['Ptolemaic', 'masonic', 'tangled']

>>> word(4, exclude_pos=['adj', 'adv', 'noun', 'pron', 'verb'])
['beneath', 'now that', 'upon', 'yup']

>>> word(2, min_word_len=20)
['plasma thromboplastin antecedent',
'United States House of Representatives']

>>> word(count=5, max_word_len=3)
['say', 'Ofo', 'rag', 'act', 'N']

>>> word(3, min_word_len=4, max_word_len=5)
['alga', 'butch', 'nark']

>>> word(2, word_len=7)
['kinesis', 'outcrop']

>>> word(3, starts_with="ly")
['lysogeny', 'lymphoblastic leukemia', 'lyceum']

>>> word(3, ends_with="en")
['genus Pecten', 'Dinesen', 'Eigen']

>>> word(3, starts_with="un", ends_with="e")
['unchaste', 'undersize', 'unprotective']

>>> word(3, pattern="ten")
['lichtenoid eczema', 'potential unit', 'minuteness']

>>> word(count=2, starts_with="e", ends_with="n", pattern="non")
['enigma canon', 'epiphenomenon']
```

Random names:

```
>>> from randword import name, surname, fullname


>>> name()
'Ethelred'
```

(continues on next page)

(continued from previous page)

```
>>> name(gender='m')
'Elden'

>>> name(gender='f')
'Julee'

>>> name(count=4)
['Claudie', 'Trisha', 'Griffith', 'Annamarie']

>>> name(4, 'm')
['Helmuth', 'Collins', 'Ulrich', 'Zebedee']


>>> surname()
'Quicksall'

>>> surname(4)
['Shahan', 'Eickhoff', 'Akamiro', 'Giovanelli']


>>> fullname()
'Charmane Bitzel'

>>> fullname(gender='m')
'Nevin Mcnaught'

>>> fullname(gender='f')
'Sophia Comans'

>>> fullname(count=2)
['Annetta Tiso', 'Babette Velazquez']

>>> fullname(2, 'm')
['Thaxter Vanhofwegen', 'Timmie Coray']
```

Random sequences, letters and digits:

```
>>> from randword import sequence, letter, digits


>>> sequence()
'8OOBn9XN'

>>> sequence(5)
['hcre1hlC', 'jXTIqVAU', '6BwH7sUM', '2nAvHVh8', '6OANP6dO']

>>> sequence(5, 3)
['Tdv', '8Q0', 'HKG', 'K7X', 'Rwi']


>>> letter()
'Q'

>>> letter(10)
['D', 'M', 'N', 'j', 'h', 't', 'L', 'H', 'X', 'p']
```

(continues on next page)

(continued from previous page)

```
>>> digit()
'8'

>>> digit(10)
['1', '3', '6', '7', '5', '9', '4', '8', '2', '0']
```

Random places:

```
>>> from randword import country, city


>>> country()
'Romania'

>>> country(4)
['Lithuania', 'Ethiopia', 'Romania', 'Cyprus']


>>> city()
'Charlotte'

>>> city(4)
['Scottsdale', 'Jefferson', 'Vero Beach', 'Gainesville']
```

Some other random stuff:

```
>>> from randword import magic_8ball, flip_coin


>>> magic_8ball()
Ask me a question:
  Will the weather be good tomorrow?
Thinking...
  Cannot predict now.

Would you like to ask another question? [Y/N] n
Come back if you have questions.


>>> flip_coin()
False
>>> flip_coin()
True
```

CHAPTER 2

randword Package

2.1 randword Package

The Python module for generating random English words.

2.2 rand_word Module

randword.rand_word.**word**(*count: Optional[int] = None*, *include_pos: Optional[List[str]] = None*, *exclude_pos: Optional[List[str]] = None*, *word_len: Optional[int] = None*, *min_word_len: int = 1*, *max_word_len: Optional[int] = None*, *starts_with: Optional[str] = None*, *ends_with: Optional[str] = None*, *pattern: Optional[str] = None*) → Union[str, List[str]]

Returns a random English word or a list of words

Abbreviation "pos" means "part of speech"

Parameters

- **count** (*int, optional*) – The number of words to be generated. Defaults to None.
- **include_pos** (*list of str, optional*) – List of parts of speech that will be included in the generation. Defaults to *None*
- **exclude_pos** (*list of str, optional*) – List of parts of speech that will be excluded in the generation. Defaults to *None*
- **word_len** (*int, optional*) – Specifies the length of a word. Ignores the *min_word_len* and *max_word_len* parameters. Defaults to *None*
- **min_word_len** (*int, optional*) – The minimum word length. Defaults to *1*
- **max_word_len** (*int, optional*) – The maximum word length. Defaults to *None*
- **starts_with** (*str, optional*) – The pattern with which the word begins. Defaults to *None*

- **ends_with** (*str, optional*) – The pattern with which the word ends. Defaults to *None*
- **pattern** (*str, optional*) – The pattern that should be contained in the word. Defaults to *None*

Returns A random English word if *count* is *None* or a list of them if *count* is not *None*

Return type Union[str, List[str]]

Raises `IndexError` – If the word was not found or if the desired number of words was not found

2.3 `rand_name` Module

`randword.rand_name.`**`fullname`**(*count: Optional[int] = None, gender: Optional[str] = None*) → Union[str, List[str]]

Returns a random fullname or a list of them

Parameters

- **count** (*int, optional*) – The number of fullnames to be generated. Defaults to *None*
- **gender** (*str*) – Specifies the fullname of which gender will be generated. Defaults to *None*

Returns A random fullname if *count* is *None* or a list of random fullnames if *count* is not *None*

Return type Union[str, List[str]]

`randword.rand_name.`**`name`**(*count: Optional[int] = None, gender: Optional[str] = None*) → Union[str, List[str]]

Returns a random first name or a list of them

Parameters

- **count** (*int, optional*) – The number of names to be generated. Defaults to *None*
- **gender** (*str, optional*) – Specifies the name of which gender will be generated. Defaults to *None*

Returns A random first name if *count* is *None* or a list of random first names if *count* is not *None*

Return type Union[str, List[str]]

`randword.rand_name.`**`surname`**(*count: Optional[int] = None*) → Union[str, List[str]]

Returns a random surname or a list of them

Parameters **count** (*int, optional*) – The number of surnames to be generated. Defaults to *None*

Returns A random surname if *count* is *None* or a list of surnames if *count* is not *None*

Return type Union[str, List[str]]

2.4 `rand_place` Module

`randword.rand_place.`**`city`**(*count: Optional[int] = None*) → Union[str, List[str]]

Returns a random city or a list of them

Parameters **count** (*int, optional*) – The number of cities to be generated. Defaults to *None*

Returns A random city if *count* is *None* or a list of cities if *count* is not *None*

Return type Union[str, List[str]]

`randword.rand_place.country(count: Optional[int] = None)` → Union[str, List[str]]

Returns a random country or a list of a random countries

Parameters **count** (*int, optional*) – The number of countries to be generated. Defaults to *None*

Returns A random country if *count* is *None* or a list of countries if *count* is not *None*

Return type Union[str, List[str]]

2.5 `rand_letter` Module

`randword.rand_letter.digit(count: Optional[int] = None)` → Union[str, List[str]]

Returns a random digit

Parameters **count** (*int, optional*) – The number of digits to be generated. Defaults to *None*

Returns A single digit if *count* is *None* or a list of digits if *count* is not *None*

Return type Union[str, List[str]]

`randword.rand_letter.letter(count: Optional[int] = None)` → Union[str, List[str]]

Returns a random ASCII letter or a list of them

Parameters **count** (*int, optional*) – The number of letters to be generated. Defaults to *None*

Returns An ASCII letter if *count* is *None* or a list of letters if *count* is not *None*

Return type Union[str, List[str]]

`randword.rand_letter.sequence(count: Optional[int] = None, length: int = 8)` → Union[str, List[str]]

Returns a random sequence consisting of ASCII symbols and digits or a list of a random sequences

Parameters

- **count** (*int, optional*) – The number of sequances to be generated. Defaults to *None*
- **length** (*int, optional*) – The length of the sequence. Defaults to *8*

Returns The sequence if *count* is *None* or a list of sequences if *count* is not *None*

Return type Union[str, List[str]]

2.6 `rand_other` Module

`randword.rand_other.flip_coin()` → int

`randword.rand_other.getrandbits(k)` → x. Generates an int with k random bits.

`randword.rand_other.magic_8ball()` → None

`randword.rand_other.random()` → x in the interval [0, 1).

CHAPTER 3

License

MIT License

Copyright (c) 2020 Artyom Bezmenov

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

randword, 5
randword.rand_letter, 7
randword.rand_name, 6
randword.rand_other, 7
randword.rand_place, 6
randword.rand_word, 5

Index

C

`city()` (*in module randword.rand_place*), 6
`country()` (*in module randword.rand_place*), 7

D

`digit()` (*in module randword.rand_letter*), 7

F

`flip_coin()` (*in module randword.rand_other*), 7
`fullname()` (*in module randword.rand_name*), 6

G

`getrandbits()` (*in module randword.rand_other*), 7

L

`letter()` (*in module randword.rand_letter*), 7

M

`magic_8ball()` (*in module randword.rand_other*), 7

N

`name()` (*in module randword.rand_name*), 6

R

`random()` (*in module randword.rand_other*), 7
`randword` (*module*), 5
`randword.rand_letter` (*module*), 7
`randword.rand_name` (*module*), 6
`randword.rand_other` (*module*), 7
`randword.rand_place` (*module*), 6
`randword.rand_word` (*module*), 5

S

`sequence()` (*in module randword.rand_letter*), 7
`surname()` (*in module randword.rand_name*), 6

W

`word()` (*in module randword.rand_word*), 5