

Install Sphinx

Windows, Mac, Linux

<https://www.sphinx-doc.org/en/1.3.1/install.html>

Sphinx - get started

- Change directory to code repository
 - \$ sphinx-quickstart
- Separate source and build directories (y/n) [n]: **y**
- Enter project details
- You can make your html page by typing
 - \$ make html
- Done

Home Page

- Your home page is “index.rst” found in the “source” directory
- Restructured Text (reST) format guide:
 - https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html

RST Basics

Title

Only two rules:

- If **under and overline** are used, their **length must be identical**
- The length of the underline must be at least as long as the title itself

subtitle

You can use any of these characters for the under/overline. However, it is better to stick to the same convention throughout a project. For example:

- # with overline, for parts
- * with overline, for chapters
- =, for sections
- , for subsections
- ^, for subsubsections
- “, for paragraphs

subsubtitle

Implicit Links to Titles

index.rst

=====

Main topic

=====

Blah blah blah blah

We put an internal reference to topic1 like this ``Topic1`_`

And here is another link ``Topic2`_` that points to topic2

Topic1

Something something

Topic2

+++++

Something something

Appears as:

Main topic

Blah blah blah blah

We put an internal reference to topic1 like this

[Topic1](#)

And here is another link [Topic2](#) that points to topic2

Topic1

Something something

Topic2

Something something

Toctree

```
.. toctree::  
    :maxdepth: 2  
    :caption: Contents:
```

```
file1  
file2  
file3
```

- Multiple files need to be added to the toctree
- You can also modify toctree behaviour with:
 - **maxdepth**
 - **numbered**
 - **titlesonly**
 - **glob**
 - **hidden**
- More info [here](#)

Links to other files

File1.rst

File2.rst

Appears as:

```
.. _interesting_topic:  
====  
topic  
=====
```

Blah blah blah blah

You can find more interesting
topics at interesting_topic_

Or

You can find more interesting
topics at :ref: `interesting_topic`

Or

You can find more interesting
topics at :ref: `here`
<interesting_topic>

You can find more interesting
topics at [topic](#)

You can find more interesting
topics at [here](#)

Adding code path to sphinx

Add the path to codes bu uncommenting the following lines in conf.py, found in source.

```
import os
import sys
#the code in this example is found one directory
#above the source
sys.path.insert(0, os.path.abspath('../'))
sys.setrecursionlimit(1500)
```


Get module string doc

In conf.py add extension
extensions = ['sphinx.ext.autodoc']

module1.py

```
"""Random variable generators

integers
-----
    uniform within range

sequences
-----
    pick random element
    pick random sample
    pick weighted random sample
    generate random permutation

distributions on the real line:
-----
    uniform
    triangular
    normal (Gaussian)
    lognormal
    negative exponential
    gamma
    beta
    pareto
    Weibull

distributions on the circle (angles 0 to 2pi)
-----
    circular uniform
    von Mises

General notes on the underlying Mersenne Twister core generator:

* The period is 2**19937-1.
* It is one of the most extensively tested generators in existence.
* The random() method is implemented in C, executes in a single Python step,
  and is, therefore, threadsafe.

"""

from warnings import warn as _warn
from types import MethodType as _MethodType, BuiltinMethodType as _BuiltinMethodType
from math import log as _log, exp as _exp, pi as _pi, e as _e, ceil as _ceil
from math import sqrt as _sqrt, acos as _acos, cos as _cos, sin as _sin
from os import urandom as _urandom
from _collections_abc import Set as _Set, Sequence as _Sequence
from hashlib import sha512 as _sha512
import itertools as _itertools
import bisect as _bisect
import os as _os

__all__ = ["Random", "seed", "random", "uniform", "randint", "choice", "sample",
           "randrange", "shuffle", "normalvariate", "lognormvariate",
           "expovariate", "vonmisesvariate", "gammavariate", "triangular",
           "gauss", "betavariate", "paretovariate", "weibullvariate",
           "getstate", "setstate", "getrandbits", "choices",
           "SystemRandom"]

NV_MAGICCONST = 4 * _exp(-0.5)/_sqrt(2.0)
TWOPI = 2.0*_pi
LOG4 = _log(4.0)
SG_MAGICCONST = 1.0 + _log(4.5)
BPF = 53 # Number of bits in a float
RECIP_BPF = 2**-BPF

# Translated by Guido van Rossum from C source provided by
```

module.rst

```
Module1 Title
=====
Something written before the doc

.. automodule:: module1
```

Appers as

Previous topic

Here is the Doc

This Page

Show Source

Quick search

Go

Module1 Title

Something written before the doc

Random variable generators.

uniform within range

pick random element pick random sample pick weighted random sample generate random permutation

uniform triangular normal (Gaussian) lognormal negative exponential gamma beta pareto Weibull

circular uniform von Mises

General notes on the underlying Mersenne Twister core generator:

- The period is 2**19937-1.
- It is one of the most extensively tested generators in existence.
- The random() method is implemented in C, executes in a single Python step, and is, therefore, threadsafe.

Get class string doc module.rst

module1.py

```
NV_MAGICCONST = 4 * _exp(-0.5)/_sqrt(2.0)
TWOPI = 2.0*_pi
LOG4 = _log(4.0)
SG_MAGICCONST = 1.0 + _log(4.5)
BPF = 53          # Number of bits in a float
RECIP_BPF = 2**-BPF

# Translated by Guido van Rossum from C source provided by
# Adrian Baddeley.  Adapted by Raymond Hettinger for use with
# the Mersenne Twister  and os.urandom() core generators.

import _random

class Random(_random.Random):
    """Random number generator base class used by bound module functions.

    Used to instantiate instances of Random to get generators that don't
    share state.

    Class Random can also be subclassed if you want to use a different basic
    generator of your own devising: in that case, override the following
    methods:  random(), seed(), getstate(), and setstate().
    Optionally, implement a getrandbits() method so that randrange()
    can cover arbitrarily large ranges.

    """

    VERSION = 3          # used by getstate/setstate

    def __init__(self, x=None):
        """Initialize an instance.

        Optional argument x controls seeding, as for Random.seed().
        """

        self.seed(x)
        self.gauss_next = None

    def seed(self, a=None, version=2):
        """Initialize internal state from hashable object.

        None or no argument seeds from current time or from an operating
        system specific randomness source if available.

        If *a* is an int, all bits are used.
```

Appers as

```
Module1 Title
=====
Something written before the doc

.. automodule:: module1

Class Random
+++++++

Here is the documentation for class random:

.. autoclass:: module1.Random
```

Orbit 1.0.0 documentation » Module1 Title

Table of Contents

Module1 Title

- Class Random

Previous topic

Here is the Doc

This Page

Show Source

Quick search

Go

Module1 Title

Something written before the doc

Random variable generators.

uniform within range

pick random element pick random sample pick weighted random sample generate random permutation

uniform triangular normal (Gaussian) lognormal negative exponential gamma beta pareto Weibull

circular uniform von Mises

General notes on the underlying Mersenne Twister core generator:

- The period is 2**19937-1.
- It is one of the most extensively tested generators in existence.
- The random() method is implemented in C, executes in a single Python step, and is, therefore, threadsafe.

Class Random

Here is the documentation for class random:

```
class module1.Random(x=None)
    Random number generator base class used by bound module functions.

    Used to instantiate instances of Random to get generators that don't share state.

    Class Random can also be subclassed if you want to use a different basic generator of your own devising: in
    that case, override the following methods: random(), seed(), getstate(), and setstate().  Optionally, implement
    a getrandbits() method so that randrange() can cover arbitrarily large ranges.
```


Get class string doc + methods

module1.py

```
NV_MAGICCONST = 4 * _exp(-0.5)/_sqrt(2.0)
TWOPI = 2.0*_pi
LOG4 = _log(4.0)
SG_MAGICCONST = 1.0 + _log(4.5)
BPF = 53          # Number of bits in a float
RECIP_BPF = 2**-BPF
```

```
# Translated by Guido van Rossum from C source provided by
# Adrian Baddeley.  Adapted by Raymond Hettinger for use with
# the Mersenne Twister and os.urandom() core generators.
```

```
import _random
```

```
class Random(_random.Random):
    """Random number generator base class used by bound module functions.
```

Used to instantiate instances of Random to get generators that don't share state.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the following methods: `random()`, `seed()`, `getstate()`, and `setstate()`. Optionally, implement a `getrandbits()` method so that `randrange()` can cover arbitrarily large ranges.

```
VERSION = 3      # used by getstate/setstate
```

```
def __init__(self, x=None):
    """Initialize an instance.
```

```
Optional argument x controls seeding, as for Random.seed().
"""
```

```
self.seed(x)
self.gauss_next = None
```

```
def seed(self, a=None, version=2):
    """Initialize internal state from hashable object.
```

None or no argument seeds from current time or from an operating system specific randomness source if available.

If **a** is an int, all bits are used.

module.rst

Appears as

Class Random
+++++

Here is the documentation for class random:

```
.. autoclass:: module1.Random
   :members:
```

Orbit 1.0.0 documentation » Class Random

Previous topic

[Here is the Doc](#)

This Page

[Show Source](#)

Quick search

Go

Class Random

Here is the documentation for class random:

class module1.Random(*x=None*)

Random number generator base class used by bound module functions.

Used to instantiate instances of Random to get generators that don't share state.

Class Random can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the following methods: `random()`, `seed()`, `getstate()`, and `setstate()`. Optionally, implement a `getrandbits()` method so that `randrange()` can cover arbitrarily large ranges.

choice(*seq*)

Choose a random element from a non-empty sequence.

choices(*population, weights=None, *, cum_weights=None, k=1*)

Return a k sized list of population elements chosen with replacement.

If the relative weights or cumulative weights are not specified, the selections are made with equal probability.

getstate()

Return internal state; can be passed to `setstate()` later.

normalvariate(*mu, sigma*)

Normal distribution.

mu is the mean, and *sigma* is the standard deviation.

randint(*a, b*)

Return random integer in range [*a, b*], including both end points.

randrange(*start, stop=None, step=1, intclass=int*)

Get function string doc

function.rst

```
Function Example
+++++++

Get function doc strings from module2.py

.. currentmodule:: module2

The run function
+++++++

short description...

.. autofunction:: run

The runctx function
+++++++

Write something here

.. autofunction:: runctx
```

Appers as

| | |
|---|--|
| Table of Contents | Function Example |
| Function Example <ul style="list-style-type: none">The run functionThe runctx function | Get function doc strings from module2.py |
| Previous topic | The run function |
| Module1 Title | |
| This Page | short description... |
| Show Source | module2.run(statement, filename=None, sort=- 1) Run statement under profiler optionally saving results in filename |
| Quick search | This function takes a single argument that can be passed to the “exec” statement, and an optional file name. In all cases this routine attempts to “exec” its first argument and gather profiling statistics from the execution. If no file name is present, then this function automatically prints a simple profiling report, sorted by the standard name string (file/line/function-name) that is presented in each line. |
| | The runctx function |
| | Write something here |
| | module2.runctx(statement, globals, locals, filename=None, sort=- 1) Run statement under profiler, supplying your own globals and locals, optionally saving results in filename. |
| | statement and filename have the same semantics as profile.run |