Python - Random Module

The random module is a built-in module to generate the pseudo-random variables. It can be used perform some action randomly such as to get a random number, selecting a random elements from a list, shuffle elements randomly, etc.

Generate Random Floats

The random.random() method returns a random float number between 0.0 to 1.0. The function doesn't need any arguments.

Example: random()

>> import random

>>> random.random()

0.645173684807533

Generate Random Integers

The random.randint() method returns a random integer between the specified integers.

Example: randint()

 Copy

>>> import random

>>> random.randint(1, 100)

95

>>> random.randint(1, 100)

49

Generate Random Numbers within Range

The random.randrange() method returns a randomly selected element from the range created by the start, stop and step arguments. The value of start is 0 by default. Similarly, the value of step is 1 by default.

Example:

 Copy

>>> random.randrange(1, 10)

2

>>> random.randrange(1, 10, 2)

5

>>> random.randrange(0, 101, 10)

80

Select Random Elements

The random.choice() method returns a randomly selected element from a non-empty sequence. An empty sequence as argument raises an IndexError.

Example:

 Copy

>>> import random

>>> random.choice('computer')

't'

>>> random.choice([12,23,45,67,65,43])

45

>>> random.choice((12,23,45,67,65,43))

67

Shuffle Elements Randomly

The random.shuffle() method randomly reorders the elements in a [list](https://www.tutorialsteacher.com/python/python-list).

Example:

 Copy

>>> numbers=[12,23,45,67,65,43]

>>> random.shuffle(numbers)

>>> numbers

[23, 12, 43, 65, 67, 45]

>>> random.shuffle(numbers)

>>> numbers

[23, 43, 65, 45, 12, 67]

# Python - sys Module

The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. You will learn some of the important features of this module here.

## sys.argv

sys.argv returns a list of command line arguments passed to a Python script. The item at index 0 in this list is always the name of the script. The rest of the arguments are stored at the subsequent indices.

Here is a Python script (test.py) consuming two arguments from the command line.

test.py

import sys

print("You entered: ",sys.argv[1], sys.argv[2], sys.argv[3])

This script is executed from command line as follows:

C:\python36> python test.py Python C# Java  
You entered: Python C# Java

Above, sys.argv[1] contains the first argument 'Python', sys.argv[2] contains the second argument 'Python', and sys.argv[3] contains the third argument 'Java'. sys.argv[0] contains the script file name test.py.

## sys.exit

This causes the script to exit back to either the Python console or the command prompt. This is generally used to safely exit from the program in case of generation of an exception.

## sys.maxsize

Returns the largest integer a variable can take.

Example: sys.maxsize

>>> import sys

>>>sys.maxsize

9223372036854775807

## sys.path

This is an environment variable that is a search path for all Python modules.

Example: sys.path

>>> import sys

>>>sys.path

['', 'C:\\python36\\Lib\\idlelib', 'C:\\python36\\python36.zip',

'C:\\python36\\DLLs', 'C:\\python36\\lib', 'C:\\python36',

'C:\\Users\\acer\\AppData\\Roaming\\Python\\Python36\\site-packages',

'C:\\python36\\lib\\site-packages']

## sys.version

This attribute displays a string containing the version number of the current Python interpreter.

Example: sys.version

>>> import sys

>>>sys.version

'3.7.0 (v3.7.0:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)]'

# Python - Built-in Modules

The [Python interactive shell](https://www.tutorialsteacher.com/python/python-interective-shell) has a number of built-in functions. They are loaded automatically as a shell starts and are always available, such as [print()](https://www.tutorialsteacher.com/python/print-method) and [input()](https://www.tutorialsteacher.com/python/input-function) for I/O, number conversion functions [int()](https://www.tutorialsteacher.com/python/int-method" \t "_blank), [float()](https://www.tutorialsteacher.com/python/float-method), [complex()](https://www.tutorialsteacher.com/python/complex-method), data type conversions [list()](https://www.tutorialsteacher.com/python/list-method), [tuple()](https://www.tutorialsteacher.com/python/tuple-method), [set()](https://www.tutorialsteacher.com/python/set-method), etc.

In addition to built-in functions, a large number of pre-defined functions are also available as a part of libraries bundled with Python distributions. These functions are defined in [modules](https://www.tutorialsteacher.com/python/python-module) are called built-in modules.

Built-in modules are written in C and integrated with the Python shell. Each built-in module contains resources for certain system-specific functionalities such as OS management, disk IO, etc. The standard library also contains many Python scripts (with the .py extension) containing useful utilities.

To display a list of all available modules, use the following command in the Python console:

>>> help('modules')

>>> help('modules')

IPython \_weakrefset heapq secrets

\_\_future\_\_ \_winapi hmac select

\_abc abc html selectors

\_ast aifc http setuptools

\_asyncio antigravity idlelib shelve

\_bisect argparse imaplib shlex

\_blake2 array imghdr shutil

\_bootlocale ast imp signal

\_bz2 asynchat importlib simplegeneric

\_codecs asyncio ind site

\_codecs\_cn asyncore inspect six

\_codecs\_hk atexit io smtpd

\_codecs\_iso2022 audioop ipaddress smtplib

\_codecs\_jp autoreload ipython\_genutils sndhdr

\_codecs\_kr backcall itertools socket

\_codecs\_tw base64 jedi socketserver

\_collections bdb json sqlite3

\_collections\_abc binascii keyword sre\_compile

\_compat\_pickle binhex lib2to3 sre\_constants

\_compression bisect linecache sre\_parse

\_contextvars builtins locale ssl

\_csv bz2 logging stat

\_ctypes cProfile lzma statistics

\_ctypes\_test calendar macpath storemagic

\_datetime cgi mailbox string

\_decimal cgitb mailcap stringprep

\_distutils\_findvs chunk marshal struct

\_dummy\_thread cmath math subprocess

\_elementtree cmd mimetypes sunau

\_functools code mmap symbol

\_hashlib codecs modulefinder sympyprinting

\_heapq codeop msilib symtable

\_imp collections msvcrt sys

\_io colorama multiprocessing sysconfig

\_json colorsys netrc tabnanny

\_locale compileall nntplib tarfile

\_lsprof concurrent nt telnetlib

\_lzma configparser ntpath tempfile

\_markupbase contextlib nturl2path test

\_md5 contextvars numbers tests

\_msi copy opcode textwrap

\_multibytecodec copyreg operator this

\_multiprocessing crypt optparse threading

\_opcode csv os time

\_operator ctypes parser timeit

\_osx\_support curses parso tkinter

\_overlapped cythonmagic pathlib token

\_pickle dataclasses pdb tokenize

\_py\_abc datetime pickle trace

\_pydecimal dbm pickleshare traceback

\_pyio decimal pickletools tracemalloc

\_queue decorator pip traitlets

\_random difflib pipes tty

\_sha1 dis pkg\_resources turtle

\_sha256 distutils pkgutil turtledemo

\_sha3 doctest platform types

\_sha512 dummy\_threading plistlib typing

\_signal easy\_install poplib unicodedata

\_sitebuiltins email posixpath unittest

\_socket encodings pprint urllib

\_sqlite3 ensurepip profile uu

\_sre enum prompt\_toolkit uuid

\_ssl errno pstats venv

\_stat faulthandler pty warnings

\_string filecmp py\_compile wave

\_strptime fileinput pyclbr wcwidth

\_struct fnmatch pydoc weakref

\_symtable formatter pydoc\_data webbrowser

\_testbuffer fractions pyexpat winreg

\_testcapi ftplib pygments winsound

\_testconsole functools queue wsgiref

\_testimportmultiple gc quopri xdrlib

\_testmultiphase genericpath random xml

\_thread getopt re xmlrpc

\_threading\_local getpass reprlib xxsubtype

\_tkinter gettext rlcompleter zipapp

\_tracemalloc glob rmagic zipfile

\_warnings gzip runpy zipimport

\_weakref hashlib sched zlib

Enter any module name to get more help. Or, type "modules spam" to search

for modules whose name or summary contain the string "spam".

# Python - OS Module

It is possible to automatically perform many operating system tasks. The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

You first need to import the os module to interact with the underlying operating system. So, import it using the import os statement before using its functions.

## Getting Current Working Directory

The getcwd() function confirms returns the current working directory.

Example: Get Current Working Directory

>>> import os

>>> os.getcwd()

'C:\\Python37'

## Creating a Directory

We can create a new directory using the os.mkdir() function, as shown below.

Example: Create a Physical Directory

>>> import os

>>> os.mkdir("C:\MyPythonProject")

A new directory corresponding to the path in the string argument of the function will be created. If you open the C:\ drive, then you will see the MyPythonProject folder has been created.By default, if you don't specify the whole path in the mkdir() function, it will create the specified directory in the current working directory or drive. The following will create MyPythonProject in the C:\Python37 directory.

Example: Create a Physical Directory

>>> import os

>>> os.getcwd()

'C:\Python37'

>>> os.mkdir("MyPythonProject")

## Changing the Current Working Directory

We must first change the current working directory to a newly created one before doing any operations in it. This is done using the chdir() function. The following change current working directory to C:\MyPythonProject.

Example: Change Working Directory

>>> import os

>>> os.chdir("C:\MyPythonProject") # changing current workign directory

>>> os.getcwd()

'C:\MyPythonProject'

You can change the current working directory to a drive. The following makes the C:\ drive as the current working directory.

Example: Change Directory to Drive

>>> os.chdir("C:\\")

>>> os.getcwd()

'C:\\'

In order to set the current directory to the parent directory use ".." as the argument in the chdir() function.

Example: Change CWD to Parent

>>> os.chdir("C:\\MyPythonProject")

>>> os.getcwd()

'C:\\MyPythonProject'

>>> os.chdir("..")

>>> os.getcwd()

'C:\\'

ADVERTISEMENT

## Removing a Directory

The rmdir() function in the OS module removes the specified directory either with an absolute or relative path. Note that, for a directory to be removed, it should be empty.

Example: Remove Directory

>>> import os

>>> os.rmdir("C:\\MyPythonProject")

However, you can not remove the current working directory. To remove it, you must change the current working directory, as shown below.

Example: Remove Directory

>>> import os

>>> os.getcwd()

'C:\\MyPythonProject'

>>> os.rmdir("C:\\MyPythonProject")

PermissionError: [WinError 32] The process cannot access the file because it is being used by another process: 'd:\\MyPythonProject'

>>> os.chdir("..")

>>> os.rmdir("MyPythonProject")

Above, the MyPythonProject will not be removed because it is the current directory. We changed the current working directory to the parent directory using os.chdir("..") and then remove it using the rmdir() function.

## List Files and Sub-directories

The listdir() function returns the list of all files and directories in the specified directory.

Example: List Directories

>>> import os

>>> os.listdir("c:\python37")

['DLLs', 'Doc', 'fantasy-1.py', 'fantasy.db', 'fantasy.py', 'frame.py',

'gridexample.py', 'include', 'Lib', 'libs', 'LICENSE.txt', 'listbox.py', 'NEWS.txt',

'place.py', 'players.db', 'python.exe', 'python3.dll', 'python36.dll', 'pythonw.exe',

'sclst.py', 'Scripts', 'tcl', 'test.py', 'Tools', 'tooltip.py', 'vcruntime140.dll',

'virat.jpg', 'virat.py']

If we don't specify any directory, then list of files and directories in the current working directory will be returned.

Example: List Directories of CWD

>>> import os

>>>os.listdir()

['.config', '.dotnet', 'python']

# Python - Math Module

Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module.

Pi is a well-known mathematical constant, which is defined as the ratio of the circumference to the diameter of a circle and its value is 3.141592653589793.

Example: Getting Pi Value

>>> import math

>>>math.pi

3.141592653589793

Another well-known mathematical constant defined in the math module is **e**. It is called **Euler's number** and it is a base of the natural logarithm. Its value is 2.718281828459045.

Example: e Value

>>> import math

>>> math.e

2.718281828459045

The math module contains functions for calculating various trigonometric ratios for a given angle. The functions (sin, cos, tan, etc.) need the angle in radians as an argument. We, on the other hand, are used to express the angle in degrees. The math module presents two angle conversion functions: degrees() and radians(), to convert the angle from degrees to radians and vice versa. For example, the following statements convert the angle of 30 degrees to radians and back (Note: π radians is equivalent to 180 degrees).

Example: Math Radians and Degrees

>>> import math

>>> math.radians(30)

0.5235987755982988

>>> math.degrees(math.pi/6)

29.999999999999996

The following statements show sin, cos and tan ratios for the angle of 30 degrees (0.5235987755982988 radians):

Example: sin, cos, tan Calculation

>>> import math

>>> math.sin(0.5235987755982988)

0.49999999999999994

>>> math.cos(0.5235987755982988)

0.8660254037844387

>>> math.tan(0.5235987755982988)

0.5773502691896257

You may recall that sin(30)=0.5, cos(30)=32 (which is 0.8660254037844387) and tan(30)= 13 (which is 0.5773502691896257).

## math.log()

The math.log() method returns the natural logarithm of a given number. The natural logarithm is calculated to the base e.

Example: log

>>> import math

>>>math.log(10)

2.302585092994046

## math.log10()

The math.log10() method returns the base-10 logarithm of the given number. It is called the standard logarithm.

Example: log10

>>> import math

>>>math.log10(10)

1.0

## math.exp()

The math.exp() method returns a float number after raising e to the power of the given number. In other words, exp(x) gives e\*\*x.

Example: Exponent

>>> import math

>>>math.exp(10)

22026.465794806718

This can be verified by the exponent operator.

Example: Exponent Operator \*\*

>>> import math

>>>math.e\*\*10

22026.465794806703

## math.pow()

The math.pow() method receives two float arguments, raises the first to the second and returns the result. In other words, pow(4,4) is equivalent to 4\*\*4.

Example: Power

>>> import math

>>> math.pow(2,4)

16.0

>>> 2\*\*4

16

## math.sqrt()

The math.sqrt() method returns the square root of a given number.

Example: Square Root

>>> import math

>>> math.sqrt(100)

10.0

>>> math.sqrt(3)

1.7320508075688772

The following two functions are called representation functions. The **ceil()** function approximates the given number to the smallest integer, greater than or equal to the given floating point number. The floor() function returns the largest integer less than or equal to the given number.

Example: Ceil and Floor

>>> import math

>>> math.ceil(4.5867)

5

>>> math.floor(4.5687)

4

# Python - Collections Module

The collections module provides alternatives to built-in container data types such as list, tuple and dict.

## namedtuple()

The namedtuple() function returns a tuple-like object with named fields. These field attributes are accessible by lookup as well as by index.

General usage of this function is:

Signature:

collections.namedtuple(type\_name, field-list)

The following statement declares a student class having name, age and marks as fields.

Example: Declare a Named Tuple

>>> import collections

>>> student = collections.namedtuple('student', [name, age, marks])

To create a new object of this namedtuple, do the following:

Example: Create Object of Named Tuple

>>> s1 = student("Imran", 21, 98)

The values of the field can be accessible by attribute lookup:

Example: Access Named Tuple

>>> s1.name

'Imran'

Or by index:

Example: Access Named Tuple

 Copy

>>>s1[0]

'Imran'

## OrderedDict()

The OrderedDict() function is similar to a normal dictionary object in Python. However, it remembers the order of the keys in which they were first inserted.

Example: Ordered Dictionary

 Copy

import collections

d1 = collections.OrderedDict()

d1['A'] = 65

d1['C'] = 67

d1['B'] = 66

d1['D'] = 68

for k,v in d1.items():

print (k,v)

Output

A 65

C 67

B 66

D 68

Upon traversing the dictionary, pairs will appear in the order of their insertion.

## deque()

A deque object support appends and pops from either ends of a list. It is more memory efficient than a normal list object. In a normal list object, the removal of any item causes all items to the right to be shifted towards left by one index. Hence, it is very slow.

Example: Deque

 Copy

>>> q=collections.deque([10,20,30,40])

>>> q.appendleft(0)

>>> q

deque([0, 10, 20, 30, 40])

>>> q.append(50)

>>> q

deque([0, 10, 20, 30, 40, 50])

>>>q.pop()

50

>>> q

deque([0, 10, 20, 30, 40])

>>> q.popleft()

0

>>> q

deque([10, 20, 30, 40])