
Concept of Library

The "Python library" contains several different kinds of components.

- **It contains data types** that would normally be considered part of the "core" of a language, such as numbers and lists.

  For these types, the Python language core defines **the form of literals** and **places some constraints on their semantics**, but does not fully define the semantics. (On the other hand, the language core does define syntactic properties like the spelling and priorities of operators.)
- **Python contains a large *library* of standard functions** which can be used for common programming tasks (You can also create your own).
- **The functions in the library are contained in separate *modules***, similar to the ones you have been writing and saving in the editor so far. **In order to use a particular module, you must explicitly *import* it. This gives you access to the functions it contains**.

Modules in Python are simply Python files with a .py extension. The name of the module will be the name of the file. A Python module can have a set of functions, classes or variables defined and implemented.

Math module

It provides access to the mathematical functions.

Syntax

```
import math
```

- `ceil(...)`

  `ceil(x)`

  Return the ceiling of x as an Integral. This is the smallest integer >= x.

  Examples:

```
>>> math.ceil(10.3456)
11
>>> math.ceil(10.9999)
```

```
>>> math.ceil(-10.9999)
-10
```

- floor(...)

  floor(x)

  Return the floor of x as an Integral. This is the largest integer <= x.

Examples:

```
>>> math.floor(10.9999)
10
>>> math.floor(10.3456)
10
>>> math.floor(-10.3456)
-11
```

- ```
  >>> math.e
  2.718281828459045
  ```

- exp(...)

  exp(x)

  Return e raised to the power of x.

Example:

```
>>>math.exp(10)
22026.465794806718
```

- factorial(…)

  factorial(x) -> Integral

  Find x!. Raise a ValueError if x is negative or non-integral.

  Examples:

```
>>> math.factorial(5)
120
>>> math.factorial(-5)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: factorial() not defined for negative values
```

- sqrt(...)

  sqrt(x)

  Return the square root of x.
```

Examples:

```
>>> math.sqrt(4)
    2.0
 >>> math.sqrt(-4)
    Traceback (most recent call last):
     File "<stdin>", line 1, in <module>
    ValueError: math domain error
```

- `>>> math.pi`

```
3.141592653589793
```

- `trunc(...)`

  `trunc(x:Real)` -> Integral

  Truncates x to the nearest Integral toward 0.

Examples:

```
>>> math.trunc(10.003)
     10
>>> math.trunc(10.903)
     10
>>> math.trunc(-10.903)
     -10
```

- `gcd(...)`

  `gcd(x, y)` -> int

  greatest common divisor of x and y

Example:

```
>>> math.gcd(5,10)
     5
```

- `fabs(...)`

  `fabs(x)`

  Return the absolute value of the float x.

Example:

```
>>> math.fabs(-29)
      29.0
>>> math.fabs(-29.2)
     29.2
```

Random module

This module implements pseudo-random number generators for various distributions.

import random

- `random(...)` method of random.Random instance

  `random()` -> x in the interval [0, 1).

  Return the next random floating point number in the range [0.0, 1.0).

  Examples:

```
>>> random.random()
      0.8443234927516827
>>> random.random()
      0.8640152877209692
```

- `randint(a, b)` method of random.Random instance

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

  Examples:

```
>>> random.randint(1,4)
      1
>>> random.randint(1,4)
      1
>>> random.randint(1,4)
      3
>>> random.randint(1,4)
      4
```

- `randrange(start, stop=None, step=1 )` method of random.Random instance

  Choose a random item from range(start, stop[, step]).

  This fixes the problem with randint() which includes the endpoint.

  Examples:

```
>>> random.randrange(1,4)
      1
>>> random.randrange(1,4)
      3
>>> random.randrange(1,4,2)
```

```
1
>>> random.randrange(1,4,2)
3
```

- `uniform(a, b)`

Return a random floating point number *N* such that a <= N <= b for a <= b and b <= N <= a for b < a. The end-point value b may or may not be included in the range depending on floating-point rounding.

Examples

```
>>> random.uniform(1,4)
1.2477198867940338
>>> random.uniform(1,4)
3.3577021948021972
```

- `seed(a=None)`

Initialize the random number generator.

If a is omitted or None, the current system time is used. If randomness sources are provided by the operating system, they are used instead of the system time . If a is an int, it is used directly.

Examples

```
>>> random.seed(10)
>>> random.seed()
```

- `shuffle(x)` method of random.Random instance

Shuffle list x in place, and return None.

Examples

```
>>> a=[1,2,3,4]
>>> random.shuffle(a)
>>> a
[3, 1, 2, 4]
```

- `sample(population, k)`

Return a *k* length list of unique elements chosen from the population sequence or set. Used for random sampling without replacement.

Examples

```
>>> a=[1,2,3,4]
>>> random.sample(a,2)
```

```
[1, 4]
>>> random.sample(a,3)
[4, 1, 3]
>>> random.sample(a,7)
Traceback (most recent call last):
 raise ValueError("Sample larger than population or is
     negative")
ValueError: Sample larger than population or is negative
```

- choice(*seq*)

  Return a random element from the non-empty sequence *seq*. If *seq* is empty, raises IndexError.

  Examples

```
>>> a=[10,20,30,40,50]
>>> a
[10, 20, 30, 40, 50]
>>> random.choice(a)
10
>>> random.choice(a)
50
>>> a=[]
>>> random.choice(a)
Traceback (most recent call last):
raise IndexError('Cannot choose from an empty sequence')
from None
IndexError: Cannot choose from an empty sequence
```

You may also read the following :

In Builtins module: pow(),round()