

# Python Modules

Any text file with the `.py` extension containing Python code is basically a module. Different Python objects such as functions, classes, variables, constants, etc., defined in one module can be made available to an interpreter session or another Python script by using the `import` statement. Functions defined in built-in modules need to be imported before use. On similar lines, a custom module may have one or more user-defined Python objects in it. These objects can be imported in the interpreter session or another script.

If the programming algorithm requires defining a lot of functions and classes, they are logically organized in modules. One module stores classes, functions and other resources of similar relevance. Such a modular structure of the code makes it easy to understand, use and maintain.

## Creating a Module

Shown below is a Python script containing the definition of `sum()` function. It is saved as `calc.py`.

`calc.py`

```
def sum(x, y):  
    return x + y
```

## Importing a Module

We can now import this module and execute the `sum()` function in the [Python shell](#).

Example: Importing a Module

```
>>> import calc  
>>> calc.sum(5, 5)  
10
```

In the same way, to use the above `calc` module in another Python script, use the `import` statement.

Every module, either built-in or custom made, is an object of a module class.

Verify the type of different modules using the built-in `type()` function, as shown below.

#### Example: Module Type

```
>>> import math
>>> type(math)
<class 'module'>
>>> import calc
>>> type(calc)
<class 'module'>
```

## Renaming the Imported Module

Use the `as` keyword to rename the imported module as shown below.

#### Example:

```
>>> import math as cal
>>> cal.log(4)
1.3862943611198906
```

## from .. import statement

The above import statement will load all the resources of the module in the current working environment (also called namespace). It is possible to import specific objects from a module by using this syntax. For example, the following module `calc.py` has three functions in it.

#### calc.py

```
def sum(x,y):
    return x + y
def average(x, y):
    return (x + y)/2
def power(x, y):
    return x**y
```

Now, we can import one or more functions using the `from...import` statement. For example, the following code imports only two functions in the `test.py`.

### Example: Importing Module's Functions

```
>>> from functions import sum, average
>>> sum(10, 20)
30
>>> average(10, 20)
15
>>> power(2, 4)
```

The following example imports only one function - sum.

### Example: Importing Module's Function

```
>>> from functions import sum
>>> sum(10, 20)
30
>>> average(10, 20)
```

You can also import all of its functions using the `from...import *` syntax.

### Example: Import Everything from Module

```
>>> from functions import *
>>> sum(10, 20)
30
>>> average(10, 20)
15
>>> power(2, 2)
4
```

## Module Search Path

When the import statement is encountered either in an interactive session or in a script:

- First, the Python interpreter tries to locate the module in the current working directory.
- If not found, directories in the PYTHONPATH environment variable are searched.
- If still not found, it searches the installation default directory.

As the Python interpreter starts, it puts all the above locations in a list returned by the `sys.path` attribute.

## Example: Module Attributes

```
>>> 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']
```

If the required module is not present in any of the directories above, the message `ModuleNotFoundError` is thrown.

```
>>> import MyModule
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    ModuleNotFoundError: No module named 'MyModule'
```

## Reloading a Module

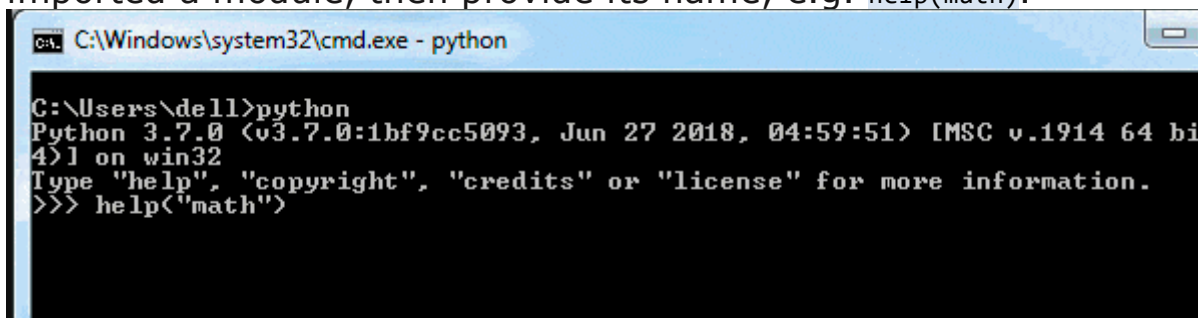
Suppose you have already imported a module and using it. However, the owner of the module added or modified some functionalities after you imported it. So, you can reload the module to get the latest module using the `reload()` function of the `imp` module, as shown below.

### Example: Reloading Module

```
>>> import imp
>>> imp.reload(calc)
<module 'calc' (built-in)>
```

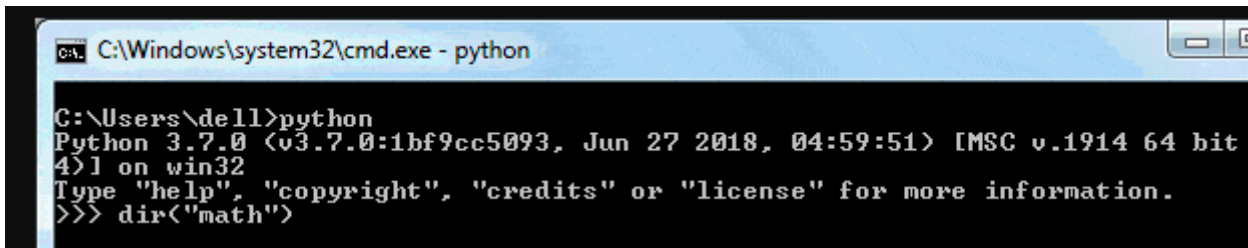
## Getting Help on Modules

Use the `help()` function to know the methods and properties of a module. For example, call the `help("math")` to know about the `math` module. If you already imported a module, then provide its name, e.g. `help(math)`.



```
C:\Windows\system32\cmd.exe - python
C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bi
4)1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help("math")
```

As shown above, you can see the method names and descriptions. It will not display pages of help ending with --More--. Press Enter to see more help. You can also use the [dir\(\)](#) function to know the names and attributes of a module.



```
C:\Windows\system32\cmd.exe - python
C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit
4)1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir("math")
```

## Python Module Attributes: name, doc, file, dict

Python module has its attributes that describes it. Attributes perform some tasks or contain some information about the module. Some of the important attributes are explained below:

### `__name__` Attribute

The `__name__` attribute returns the name of the module. By default, the name of the file (excluding the extension .py) is the value of `__name__` attribute.

Example: `__name__` Attribute

```
>>> import math
>>> math.__name__
'math'
```

In the same way, it gives the name of your custom module.

Example: `__name__` Attribute

```
>>> hello.__name__
'hello'
```

However, this can be modified by assigning different strings to this attribute. Change `hello.py` as shown below.

Example: Set `__name__`

```
def SayHello(name):
    print ("Hi {}! How are you?".format(name))
__name__="SayHello"
```

And check the `__name__` attribute now.

```
>>> import hello
>>> hello.__name__
'SayHello'
```

The value of the `__name__` attribute is `__main__` on the [Python interactive shell](#).

```
>>> __name__
'__main__'
```

When we run any Python script (i.e. a module), its `__name__` attribute is also set to `__main__`. For example, create the following `welcome.py` in [IDLE](#).

Example: `welcome.py`

```
print("__name__ = ", __name__)
```

Run the above `welcome.py` in IDLE by pressing F5. You will see the following result.

Output in IDLE:

```
>>> __name__ = __main__
```

However, when this module is imported, its `__name__` is set to its filename. Now, import the `welcome` module in the new file `test.py` with the following content.

Example: `test.py`

```
import welcome
print("__name__ = ", __name__)
```

Example: `test.py`

```
import welcome
print("__name__ = ", __name__)
```

Now run the `test.py` in IDLE by pressing F5. The `__name__` attribute is now "welcome".

Example: test.py

```
__name__ = welcome
```

This attribute allows a Python script to be used as an executable or as a module.

## \_\_doc\_\_ Attribute

The `__doc__` attribute denotes the documentation string (docstring) line written in a module code.

Example:

```
>>> import math
>>> math.__doc__
'This module is always available. It provides access to the
```

Consider the the following script is saved as `test.py` module.

test.py

```
"""This is docstring of test module"""
def SayHello(name):
    print ("Hi {}! How are you?".format(name))
    return
```

The `__doc__` attribute will return a string defined at the beginning of the module code.

Example:

```
>>> import test
>>> test.__doc__
'This is docstring of test module'
```

## `__file__` Attribute

`__file__` is an optional attribute which holds the name and path of the module file from which it is loaded.

Example: `__file__` Attribute

```
>>> import io
>>> io.__file__
'C:\\python37\\lib\\io.py'
```

## `__dict__` Attribute

The `__dict__` attribute will return a dictionary object of module attributes, functions and other definitions and their respective values.

Example: `__dict__` Attribute

```
>>> import math
>>> math.__dict__
{'__name__': 'math', '__doc__': 'This module is always available. It provides access to the mathematical functions defined by the C standard.', '__package__': '', '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': ModuleSpec(name='math', loader=<class '_frozen_importlib.BuiltinImporter'>, origin='built-in'), 'acos': <built-in function acos>, 'acosh': <built-in function acosh>, 'asin': <built-in function asin>, 'asinh': <built-in function asinh>, 'atan': <built-in function atan>, 'atan2': <built-in function atan2>, 'atanh': <built-in function atanh>, 'ceil': <built-in function ceil>, 'copysign': <built-in function copysign>, 'cos': <built-in function cos>, 'cosh': <built-in function cosh>, 'degrees': <built-in function degrees>, 'erf': <built-in function erf>, 'erfc': <built-in function erfc>, 'exp': <built-in function exp>, 'expm1': <built-in function expm1>, 'fabs': <built-in function fabs>, 'factorial': <built-in function factorial>, 'floor': <built-in function floor>, 'fmod': <built-in function fmod>, 'frexp': <built-in function frexp>, 'gamma': <built-in function gamma>, 'hypot': <built-in function hypot>, 'isfinite': <built-in function isfinite>, 'isinf': <built-in function isinf>, 'isnan': <built-in function isnan>, 'ldexp': <built-in function ldexp>, 'lgamma': <built-in function lgamma>, 'log': <built-in function log>, 'log10': <built-in function log10>, 'log1p': <built-in function log1p>, 'log2': <built-in function log2>, 'loggamma': <built-in function loggamma>, 'logspace': <built-in function logspace>, 'modf': <built-in function modf>, 'nan': <float nan>, 'nextafter': <built-in function nextafter>, 'pi': <float pi>, 'radians': <built-in function radians>, 'sin': <built-in function sin>, 'sinh': <built-in function sinh>, 'sqrt': <built-in function sqrt>, 'tan': <built-in function tan>, 'tanh': <built-in function tanh>, 'tau': <float tau>, 'trunc': <built-in function trunc>, 'zeta': <built-in function zeta>}
```