Python Module

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 organised 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 `SayHello()` function. It is saved as **hello.py**.

Example: hello.py

```
def SayHello(name):
    print("Hello {}! How are you?".format(name))
    return
```

Importing a Module

We can now import this module and execute the `SayHello()` function from the Python prompt.

```
>>> import hello
>>> hello.SayHello("Jonathan")
Hello Jonathan! How are you?
```

In the same way, to use the above `hello` 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.

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

Renaming the Imported Module

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

```
>>> 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 `functions.py` has three functions in it.

Example: functions.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: test.py
```
from functions import sum, average
print("sum: ", sum(10, 20)) # Output: sum: 30
print("average: ", average(10, 20)) # Output: average: 15
# calling power(2, 4) will throw an error
```

The following example imports only one function - sum.

Example: test.py
```
from functions import sum
print("sum:", sum(10, 20))
# calling average() or power() will throw an error
```

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

Example: test.py
```
from functions import *
print("sum: ", sum(10, 20)) # Output: sum: 30
print("average: ", average(10, 20)) # Output: average: 15
print("power: ", power(2, 2)) # Output: power: 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 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 put all the above locations in a list returned by the sys.path attribute.

```
>>> 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

We have a **hello.py** module with the following function defined in it.

Example: hello.py

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

We can call the `SayHello()` function after importing the hello module, as shown below.

```
>>> import hello
>>> hello.SayHello("Amitabh")
Hi Amitabh! How are you?
```

Now suppose we need to modify the `SayHello()` function before executing it again. The updated function is :

Example: hello.py

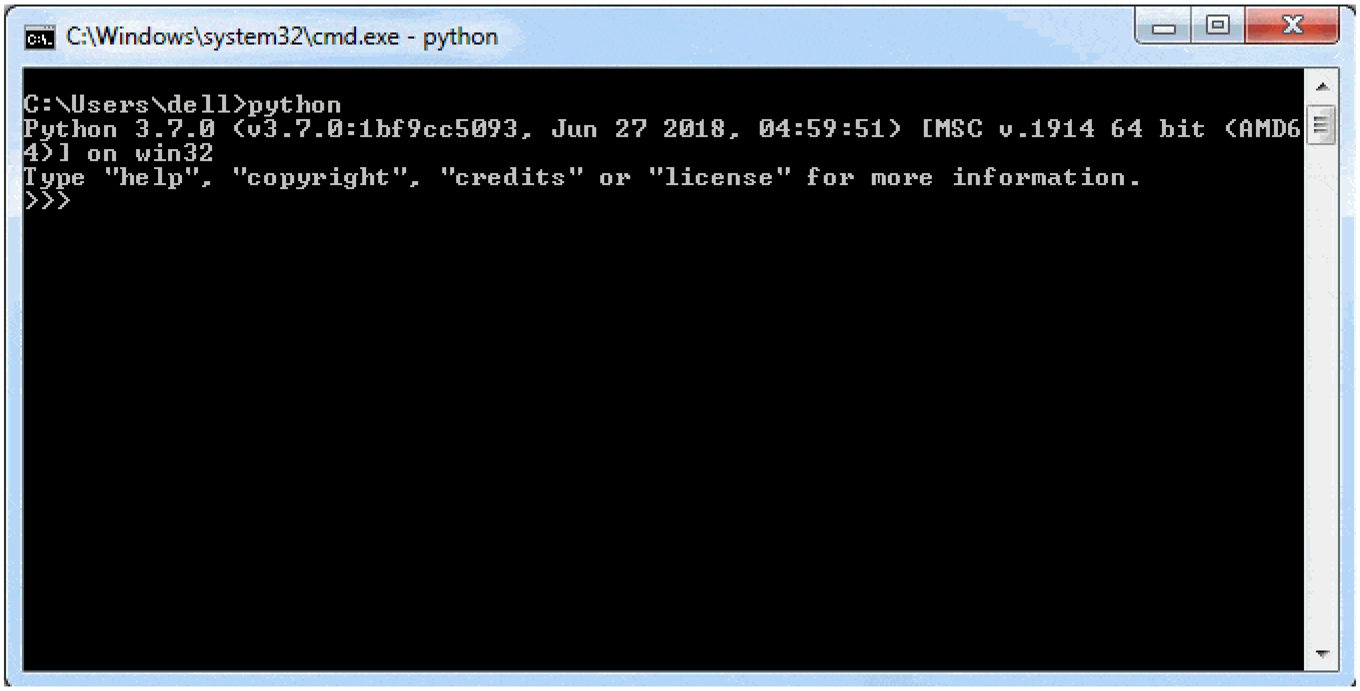
```
def SayHello(name, subject):
    print ("Hi {}! How are you?".format(name))
    print ("Welcome to {} tutorial on TutorialsTeacher".format(subject))
    return
```

This change will be executed only if we close the current interpreter session and relaunch it. If such changes are to be done frequently, to close and relaunch Python is cumbersome. In order to call the `SayHello()` function without ending the current interpreter session, we can reload it by using the `reload()` function from the `imp` module, as shown below.

```
>>> import imp
>>> imp.reload(hello)
<module 'hello' from 'C:/python36\\hello.py'>
>>> hello.SayHello("Deepak", "Python")
Hi Deepak! How are you?
Welcome to Python tutorial on TutorialsTeacher
```

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)`.

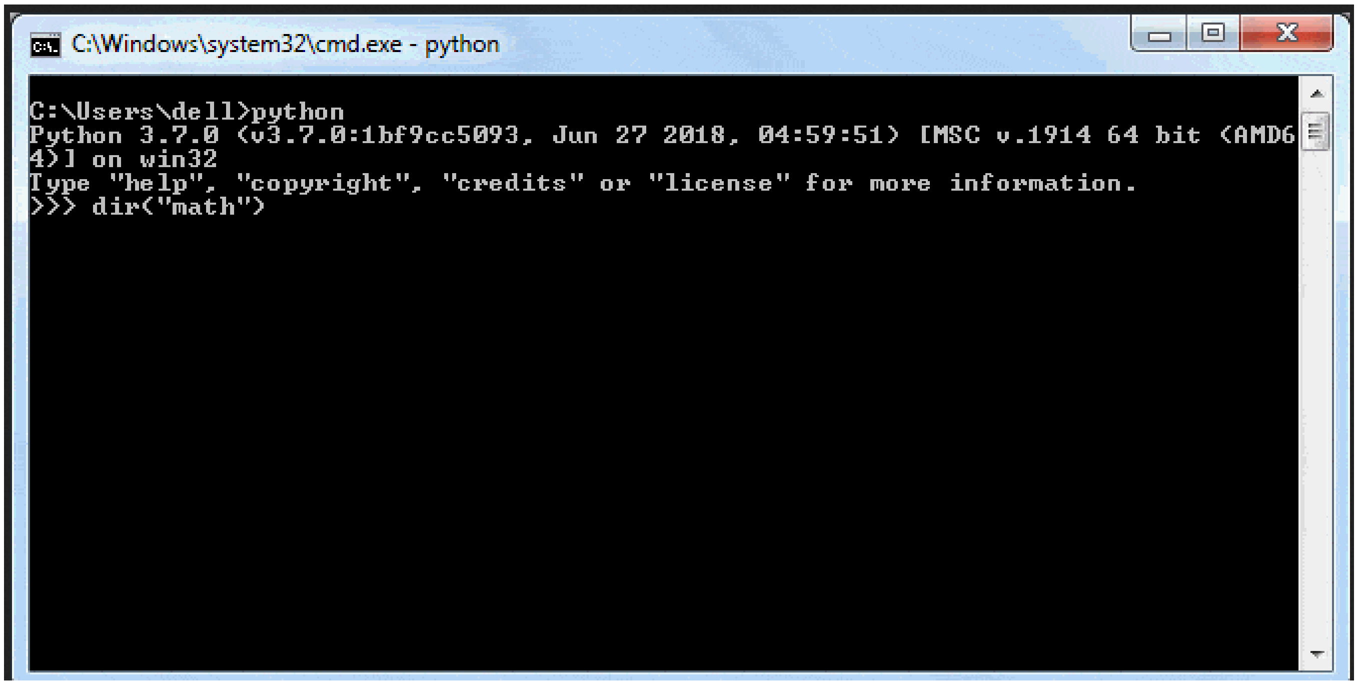


```
C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Getting Help on Module

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:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir("math")
```