# Python Modules

Modules refer to a file containing Python statements and definitions. A file containing Python code, for e.g.: example.py, is called a module and its module name would beexample. We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code. We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Let us create a module. Type the following and save it as example.py.

# Python Module example

def add(a, b):

"""This program adds two

numbers and return the result"""

result = a + b

return result

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

Importing modules

We can import the definitions inside a module to another module or the interactive interpreter in Python. We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

>>> import example

This does not enter the names of the functions defined in example directly in the current symbol table. It only enters the module name example there. Using the module name we can access the function using dot (.) operation. For example:

>>> example.add(4,5.5)

9.5

Python has a ton of standard modules available. You can check out the full list of [Python standard modules](http://docs.python.org/3/py-modindex.html) and what they are for. These files are in the Lib directory inside the location where you installed Python. Standard modules can be imported the same way as we import our user-defined modules.

There are various ways to import modules. They are listed as follows.

The import statement

We can import a module using import statement and access the definitions inside it using the dot operator as described above. Here is an example.

# import statement example

# to import standard module math

import math

print("The value of pi is", math.pi)

Output

The value of pi is 3.141592653589793

Import with renaming

We can import a module by renaming it as follows.

# import module by renaming it

import math as m

print("The value of pi is", m.pi)

The output of this is same as above. We have renamed the math module as m. This can save us typing time in some cases. Note that the name math is not recognized in our scope. Hence, math.pi is invalid, m.pi is the correct implementation.

The from...import statement

We can import specific names form a module without importing the module as a whole. Here is an example.

# import only pi from math module

from math import pi

print("The value of pi is", pi)

The output of this is same as above. We imported only the attribute pi form the module. In such case we don't use the dot operator. We could have imported multiple attributes as follows.

>>> from math import pi, e

>>> pi

3.141592653589793

>>> e

2.718281828459045

Import all names

We can import all names(definitions) form a module using the following construct.

# import all names form

# the standard module math

from math import \*

print("The value of pi is", pi)

The output of this is same as above. We imported all the definitions from the math module. This makes all names except those beginnig with an underscore, visible in our scope.

Importing everything with the asterisk (\*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

Python Module Search Path

While importing a module, Python looks at several places. Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path. The search is in this order.

* The current directory.
* PYTHONPATH (an environment variable with a list of directory).
* The installation-dependent default directory.

>>> import sys

>>> sys.path

['',

'C:\\Python33\\Lib\\idlelib',

'C:\\Windows\\system32\\python33.zip',

'C:\\Python33\\DLLs',

'C:\\Python33\\lib',

'C:\\Python33',

'C:\\Python33\\lib\\site-packages']

We can add modify this list to add our own path.

Reloading a module

The Python interpreter imports a module only once during a session. This makes things more efficient. Here is an example to show how this works.

Suppose we have the following code in a module named my\_module.

# This module shows the effect of

# multiple imports and reload

print("This code got executed")

Now we see the effect of multiple imports.

>>> import my\_module

This code got executed

>>> import my\_module

>>> import my\_module

We can see that our code got executed only once. This goes to say that our module was imported only once.

Now if our module changed during the course of the program, we would have to reload it. One way to do this is to restart the interpreter. But this does not help much. Python provides a neat way of doing this. We can use the reload() function inside theimp module to reload a module. This is how its done.

>>> import imp

>>> import my\_module

This code got executed

>>> import my\_module

>>> imp.reload(my\_module)

This code got executed

<module 'my\_module' from '.\\my\_module.py'>

The dir() built-in function

We can use the dir() function to find out names that are defined inside a module. For example, we have defined a function add() in the module example that we had in the beginning.

>>> dir(example)

['\_\_builtins\_\_',

'\_\_cached\_\_',

'\_\_doc\_\_',

'\_\_file\_\_',

'\_\_initializing\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'add']

Here, we can see a sorted list of names (along with add). All other names that begin with an underscore are default Python attributes associated with the module (we did not define them ourself). For example, the \_\_name\_\_ attribute contains the name of the module.

>>> import example

>>> example.\_\_name\_\_

'example'

All the names defined in our current namespace can be found out using the dir()function without any arguments.

>>> a = 1

>>> b = "hello"

>>> import math

>>> dir()

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'a', 'b', 'math', 'pyscripter']