Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

MODULES

Learning outcomes:

What is a module?
Creating a module
The import Statement
The ‘from’ import Statement
Renaming a module
Using the dir() Function
The ‘from’ import * Statement
Locating Modules:

What is a Module?

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.
Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Creating a module

To create a module just save the code you want in a file with the file extension **.py**

For example:

A simple module, **Calculator.py**

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

def subtract(x, y):
    return (x-y)

def multiply(x, y):
    return (x*y)
```

import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file.
Now we can use the module we just created, by using the import statement:
When using a function from a module, use the syntax: ***module_name.function_name.***

import Calculator

print (Calculator.add(100, 12))
print (Calculator.multiply(100, 12))

The *from...import* Statement

Python's ***from*** statement lets you import specific attributes from a module.

For Example:

```
from math import sqrt, factorial

# if we simply do "import math", then
# math.sqrt(16) and math.factorial() are needed

print (sqrt(16))
print (factorial(6))
```

Re-naming a Module

You can create an alias when you import a module, by using the **as** keyword:

For Example:

```
import Calculator as clc
print (clc.subtract(100, 12))
print (clc.multiply(10, 12))
```

Using the dir() Function

The dir() built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables and functions that are defined in a module.

This built-in function is used to list all the function names (or variable names) in a module.

For example:

```
#  Import built-in module  random
import  random
print  (dir(random))
```

The *from...import* * Statement

It is also possible to import all names from a module into the current namespace by using the following import statement:

from modulename import *

This provides an easy way to import all the items from a module into the current namespace.

For Example:

```
from math import *
print(pow(5,3))
print(radians(30))
print(sqrt(36))
print(degrees(0.5239))
```

Locating Modules

When you import a module, the Python interpreter searches for the module in the following sequences:

- The current directory.
- If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
- If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.

Locating Modules

The module search path is stored in the system module sys as the **sys.path** variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default.

By the way, we can find the current working directory of python as follows:

```
import os
print(os.getcwd())
```