MODULES IN PYTHON

Introduction

•A module is a file containing Python definitions and statements. A module can define functions, classes and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

•A module allows you to logically organize your Python code. Simply, it is a file consisting of Python code. A module can define functions, classes and variables.

•Modular programming refers to the process of breaking a large, unwieldy programming task into separate, smaller, more manageable subtasks or modules.

Creating A Module:

Example:

```
#Save this code in a file named mymodule.py
def greeting(name):
  print("Hello, " + name)
```

Using a Module:

Now we can use the module we just created, by using the import statement

Example:

```
#Import the module named mymodule, and call the greeting function:
import mymodule
mymodule.greeting("Jonathan")
```

Output:

```
Hello, Jonathan
```

The *import* statement

We can use any Python source file as a module by executing an import statement in some other Python source file.

When interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches for importing a module.

 # A simple module, calc.py

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

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

For example, to import the module calc.py, we need to put the following command at the top of the script :

```
# importing  module calc.py
import calc

print add(10, 2)
```

Output:

12

The *from import* Statement

Python's *from* statement lets you import specific attributes from a module. The *from* .. *import* .. has the following syntax :

```
# importing sqrt() and factorial from the module math

from math import sqrt, factorial

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

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

Output:

```
4.0
720
```

Naming a Module
You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module
You can create an alias when you import a module, by using the as keyword:

Example
Create an alias for mymodule called mx:

```
import mymodule as mx
a = mx.person1["age"]
print(a)
```

Output:
36

Import all names

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

```
# import all names from the standard module math

from math import *
print("The value of pi is", pi)
```

Output:

```
The value of pi is 3.141592653589793
```

Locating Modules

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

•The current directory.
•If the module is not 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/python3/.

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.

The PYTHONPATH Variable

The PYTHONPATH is an environment variable, consisting of a list of directories. The syntax of PYTHONPATH is the same as that of the shell variable PATH.

•Here is a typical PYTHONPATH from a Windows system −
set PYTHONPATH = c:\python34\lib;

•And here is a typical PYTHONPATH from a UNIX system −
set PYTHONPATH = /usr/local/lib/python

Namespaces and Scoping

Variables are names (identifiers) that map to objects. A *namespace* is a dictionary of variable names (keys) and their corresponding objects (values).

•A Python statement can access variables in a *local namespace* and in the *global namespace*. If a local and a global variable have the same name, the local variable shadows the global variable.

•Each function has its own local namespace. Class methods follow the same scoping rule as ordinary functions.

•Python makes educated guesses on whether variables are local or global. It assumes that any variable assigned a value in a function is local.

•Therefore, in order to assign a value to a global variable within a function, you must first use the global statement.

•The statement *global VarName* tells Python that VarName is a global variable. Python stops searching the local namespace for the variable.

Example:
We define a variable *Money* in the global namespace. Within the function Money, we assign Money a value, therefore Python assumes Money as a local variable.

However, we accessed the value of the local variable *Money* before setting it, so an UnboundLocalError is the result.

```
Money = 2000
def AddMoney():
    global Money
    Money = Money + 1

print (Money)
AddMoney()
print (Money)
```

Output:

2000
2001

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.

Example:

```
 # Import built-in module math

import math
content = dir(math)
print (content)
```

Output:

['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log', 'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']

Note:
Here, the special string variable __name__ is the module's name, and __file__ is the filename from which the module was loaded.

The reload() Function

When a module is imported into a script, the code in the top-level portion of a module is executed only once.

Therefore, if you want to reexecute the top-level code in a module, you can use the *reload()* function. The reload() function imports a previously imported module again. The syntax of the reload() function is this −

```
reload(module_name)
```

Here, module_name is the name of the module you want to reload and not the string containing the module name. For example, to reload hello module, do the following −

```
reload(hello)
```

THANK YOU!!