Modules in Python

Introduction to Modules and Packages in Python.

- A library is a collection of related functions.
- In Python, library is a set of packages and every package may have one or more modules.
- Each Module can have many functions.
- The reason for this hierarchical design is to extend the functionality of the python language.
- A module in like a library, like a library, a module greatly extends the functionalities of Python

Creation of a Module

- Modules in python are the collection of functions as single or multiple files.
- The Modules are created as a usual python file with the ".py" extension.
- Example: Illustrating the creation of module to find the factorial and permutation.

The factorial of a number n is represented mathematically as follows.

$$n! = n.(n - 1).(n - 2)\ldots 2.1, \forall n \geq 1$$

When the 0! = 1 (Where n = 0) and the value of n can not be negative.

The permutation of two numbers n and r defines the number of arrangements of n objects in r orders. Mathematically, it is expressed as follows.

$$nP_r = \frac{n!}{(n-r)!}$$

Consider creating a module for finding the factorial for a number and permutation. The Python script for the example is shown below. The following Python script is saved as factorial.py.

```
1   '''Factorial Module is for finding the factorial of a number and finding the permutation of a
    given number!'''
2   #Factorial Module
3   #Fucntion for finding Factorial of a Number
```

```
4   def factorialForNumber(n):
5       if n < 0 :
6           return "Not a Valid Input for Factorial."
7       elif n == 1 or n == 0:
8           return 1
9       else:
10          return n * factorialForNumber(n-1)
11
12
13  #Function for finding Permutation n and r
14  def permutate(n, r):
15      n = int(n)
        r = int(r)
16      if r > n:
17          return "Invalid Input. (r > n)"
18      else:
19          return (factorialForNumber(n)/factorialForNumber(n-r))
20
```

Importing Modules

- The Process of importing can be performed for the following activities.

1. Using importing operation, everything inside the module can be imported.
2. Certain named variables and certain functions can be imported from a module.

The Syntax for importing the entire module is shown below.

- The import statement performs the following two activities.

1. Find the given module in the directory and follow by loading the module to the environment and initializing the module. The process of initialization is not mandatory for all packages and modules.
2. Registering the imported successfully, it will be made available in the local namespace in one of the three ways.

If no other name is specifies, and the module being imported is a top-level module, the module's name is bound in the local namespace as a reference to the imported module.

Importing Submodule of a Module

- The scope of a module is larger than a simple function and a single class. For practical purposes, a module could consist of more than one function. In such cases, Python allows us to import specific submodules using the following syntax.

Types of Imports

- There are two types of imports in Python, namely, absolute import and relative import.
- Absolute Import involves importing the entire module from its root folder and also demands the programmer to utilize the properties of the modules.
- Relative import specifies an object or module imported from its current working directory, that is the location where the import statement resides.
- There are two types of relative imports.

1. Implicit relative imports
2. Explicit relative imports

Uses of Standard Modules in Python

- When a Python interpreter shell is opened, some built-in functions are loaded by default.
- One could notice that, the functions, such as print() and input(), and number conversion functions, such as int(), float(), and complex() are available by default and no need for any import operations.
- Among the available standard modules in Python, a few modules are considered important modules for a python programmer

Math and Sympy Packages

1. *math.pi*, a constant that is available in the package, is a good approximation of the mathematical but will lead to some errors when *math.pi* is used with other functions that are available in the package. The following examples discuss the case.

   For example, the value of is *0* but the equivalent Python statement using *math* package *math.sin(math.pi)*, which results in the closest approximated value not exact zero , but it is universal truth that sin π = 0.

```
>>> math.sin(math.pi)
1.2246467991473532e-16
```

```
>>> import sympy
>>> sympy.sin(sympy.pi)
0
>>> math.pi
3.141592653589793
```

```
>>> sympy.pi
pi
>>> type(sympy.pi)
<class 'sympy.core.numbers.Pi'>
```

os Module

- The os module provides a handful way of utilizing the operating system based functionalities, such as file descriptors handling, process handling, path handling, and so on.

General os functions

os.name(): os.name() returns the name of the currently working operating system.

Process parameters:

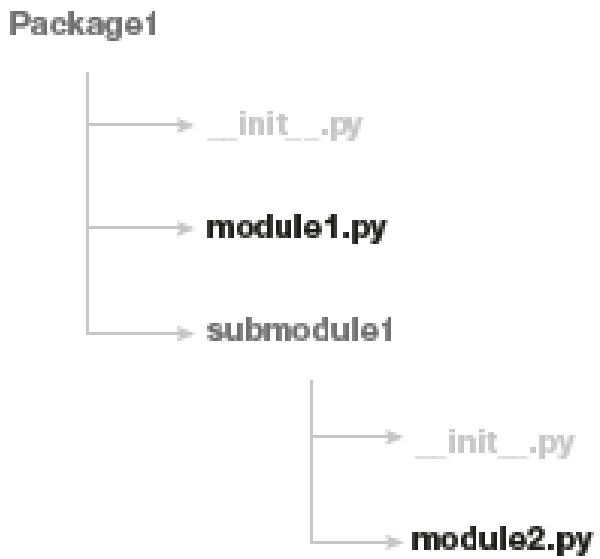
os.environ: os.environ is a class object that consists of all the locations and is stored in the system environment variables that hold the variable name and also its values.

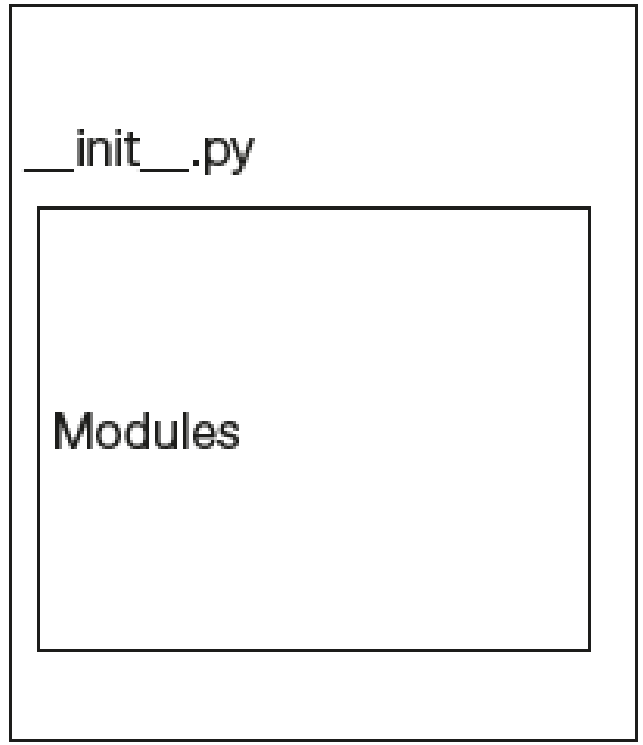
sys Module

- The sys module provides system specific parameters and functions that allow the programmer too access some variables that are maintained by the interpreter and functions that interact with the interpreter strongly.
- sys.argv: sys.argv is a variable of list data type that stores the command line arguments that are passed to a module of python while executing it.
- The general syntax for giving a command line argument is as follows.

Packages

- A package is a folder that has one or more modules and is differentiated from other folder by the special method called “__init__.py”





Differences between Packages and Modules

Packages

- Package holds __init__.py for every user oriented code.
- Package is a directory and various subdirectory that contains a collection of modules and __init__.py in the directory and every sub-directories.
- A directory contains Python modules and also has a __init__.py file by which the interpreter interprets it as a package

Modules

- Modules need not have __init__.py for any user specific codes.
- The module is a simple Python file that contains the collection of functions and global variables.
- Simple.py script with functions, classes, and objects constitutes a module.