MODULES

- **What is a module?**

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.

- **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.
For example,

```
import math
math.sqrt(25)      #5.0
math.pow(6)        #36
```

- **Module Aliases/renaming a module**

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:

For example,

```
import math as m
m.sqrt(25)      #5.0
m.pow(6)        #36
```

- **From ... Import**

Python's *from* statement lets you import specific attributes from a module. The *from .. import ...*
For example.,

```
from math import sqrt, factorial
print sqrt(16)
print factorial(6)

output:
```

```
4.0
720
```

- **Reloading a module**

reload() reloads a previously imported module. This is useful if you have edited the module source file using an external editor and want to try out the new version without leaving the Python interpreter. The return value is the module object.
Note: The argument should be a module which has been successfully imported.

Usage:
For Python2.x

```
reload(module)
```

For above 2.x and <=Python3.3

```
import imp

imp.reload(module)
```

For >=Python3.4

```
import importlib

importlib.reload(module)
```

- Built in properties of a module
- **Dir() function**

dir() is a powerful inbuilt function in Python3, which returns list of the attributes and methods of any object (say functions , modules, strings, lists, dictionaries etc.)
Syntax :

```
dir({object})
```

Parameters :
object *[optional]* : Takes object name
Returns :
dir() tries to return a valid list of attributes of the object it is called upon. Also, dir() function behaves rather differently with different type of objects, as it aims to produce the most relevant one, rather than the complete information.

- *For Class Objects, it returns a list of names of all the valid attributes and base attributes as well.*
- *For Modules/Library objects, it tries to return a list of names of all the attributes, contained in that module.*
- *If no parameters are passed it returns a list of names in the current local scope.*

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'module1']
```

```
>>> a=10
>>> b=["dzf","awrg"]
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'a', 'b', 'module1']
```

Note: If called without an argument, return the names in the current scope.

```
>>> import math
>>> x=dir(math)
>>> print(x)
```

Output:

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

- Creating user defined modules
- Module search path
- **Command line arguments**

Accessing Command Line Arguments

The Python sys module provides access to any of the command-line arguments via sys.argv. It solves two purposes:

- sys.argv is the list of command line arguments
- len(sys.argv) is the number of command line arguments that you have in your command line
- sys.argv[0] is the program, i.e. script name

Example:commands.py

```
import sys
print("Number of arguments:", len (sys.argv), "arguments.")
print("Argument List:", str(sys.argv))
```

Executing Python

You can execute Python in this way: in command prompt

filepath/>python Commands.py inp1, inp2 inp3

Output:

Number of arguments: 4 arguments.

Argument List: ['commands.py', 'inp1,', 'inp2,', 'inp3']

- Working with pre defined Standard modules (Math, Random, Datetime, Os, Sys, String,)

math:

NAME

math

DESCRIPTION

This module provides access to the mathematical functions

defined by the C standard.

Built-in functions in math module

['acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

Example:

```
Import math
math.pi()      #3.141592653589793
```

Python Random module

The Python random module functions depend on a pseudo-random number generator function random(), which generates the float number between 0.0 and 1.0.

There are different types of functions used in a random module which is as follows:

random.random()

This function generates a random float number between 0.0 and 1.0.

Example:

```
import random
a=random.random()
print(a)
```

output:

```
0.6357651732992502
```

random.randint()

This function returns a random integer between the specified integers.

```
Syntax: randint(start, end)
```

Parameters :

```
(start, end) : Both of them must be integer type values.
```

Example:

Import random

a=random.randint(10,100)

print(a)

output:

35

Note: every time take the random number with in range of 10 to 100.

random.randrange()

This function returns a randomly selected element from the range created by the start, stop, and step arguments. Value of start is 0 by default.

Syntax:

```
Syntax :
random.randrange(start(opt),stop,step(opt))
```

Example:

Print(random.randrange(100))

Output:

55

random.choice()

This function returns a randomly selected element from a non-empty sequence.or

choice() is an inbuilt function in Python programming language that returns a random

item from a list, tuple, or string.

Example:

```
import random
numberList = [111,222,333,444,555]
print("random item from list is: ", random.choice(numberList))
```

output:

```
random item from list is: 333
```

example2)

```
import random
list = [20, 30, 40, 50 ,60, 70, 80]
sampling = random.choices(list, k=4)
print("Randomly selected multiple choices using random.choices() ", sampling)
```

output:

```
Randomly selected multiple choices using random.choices() [30,
70, 20, 30]
```

random.shuffle()

This function randomly reorders the elements in the list.

Example1:

```
import random
number_list = [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
print ("Original list : ",  number_list)
random.shuffle(number_list) #shuffle method
print ("List after first shuffle  : ",  number_list)
random.shuffle(number_list)
print ("List after second shuffle : ",  number_list)
```

output:

```
Original list : [7, 14, 21, 28, 35, 42, 49, 56, 63, 70] List after first shuffle : [7, 21, 63, 28, 14, 42, 49, 56, 35, 70] List after second shuffle : [28, 42, 70, 35, 21, 7, 14, 63, 56, 49]
```

Example2:

```
import random
string_one = "pynative"
print ("Original String: ", string_one)
char_list = list(string_one) # convert string inti list
random.shuffle(char_list) #shuffle the list
string_one = ''.join(char_list)
print ("shuffled String is: ", string_one)
```

output:

Original String: pynative

shuffled String is: eiapnvyt

Datetime module

Python has a module named **datetime** to work with dates and times.

Example

```
import datetime

datetime_object = datetime.datetime.now()
print(datetime_object)
```

output:

```
2020-02-24 13:04:23.607842
```

The strftime() Method

The `datetime` object has a method for formatting date objects into readable strings.

The method is called `strftime()`, and takes one parameter, `format`, to specify the format of the returned string.

Example:

import datetime

x = datetime.datetime.now()

print(x.strftime("%B")) # %B indicates Month name, full version

output:

```
February
```

Example:

from datetime import date

date object of today's date

today = date.today()

print("Current year:", today.year)

print("Current month:", today.month)

print("Current day:", today.day)

output:

```
Current year: 2020

Current month: 2

Current day: 24
```

```
Example of timedelta:
from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)
t3 = t1 - t2
print("t3 =", t3)
print(type(t3))
output:
```

```
t3 = 14 days, 13:55:39
<class 'datetime.timedelta'>
```

Os module

The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

- **os.name:** This function gives the name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'os2', 'ce', 'java' and 'riscos'

  example:

  ```
  import os
  print(os.name)

  output:
  nt
  ```

- **os.getcwd():** Function os.getcwd(), returns the Current Working Directory(CWD) of the file used to execute the code, can vary from system to system.

  Example:

  ```
  Import os
  print(os.getcwd())

  output:
  D:\naveen\programs
  ```

- **os.error:** All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system. os.error is an alias for built-in OSError exception.

  Example:

  ```
  import os
  try:
      filename = 'GFG.txt'
      f = open(filename, 'w')
      text = f.read()
      print(text)
      f.close()
  ```

```
except IOError:
    print('Problem reading: ' + filename)

output:
Problem reading: GFG.txt
```

- **os.popen():** This method opens a pipe to or from command. The return value can be read or written depending on whether mode is ‘r’ or ‘w’.

  Example:

```
import os
fd = "GFG.txt"
# popen() is similar to open()
file = open(fd, 'w')
file.write("Hello")
file.close()
file = open(fd, 'r')
text = file.read()
print(text)
file = os.popen(fd, 'w')
file.write("Hello")

output:
Hello
5
```

Note: Output for popen() will not be shown, there would be direct changes into the file.

- **os.close():** Close file descriptor fd. A file opened using open(), can be closed by close()only. But file opened through os.popen(), can be closed with close() or os.close(). If we try closing a file opened with open(), using os.close(), Python would throw

  TypeError.

```
Example:
import os
fd = "GFG.txt"
file = open(fd, 'r')
text = file.read()
print(text)
os.close(file)

output:
Hello
```

```
TypeError: an integer is required (got type _io.TextIOWrapper)
```

Note: The same error may not be thrown, due to non-existent of file or permission privilege

- **os.rename():** A file old.txt can be renamed to new.txt, using the function os.rename(). The name of the file changes only if, the file exists and user has sufficient privilege permission to change the file.

Example:

```
import os
fd = "D:/naveen/programs/demofile.txt"
os.rename(fd,'demo_file.txt')
os.rename(fd,'New.txt')
```

output:

```
FileNotFoundError: [WinError 2] The system cannot find the file specified:
'D:/naveen/programs/demofile.txt' -> 'demo_file.txt'
```