WORKING WITH PYTHON MODULES (MODULARITY)
Module is anyone which ends with .py Extension.
OR
A module is a file consisting of Python code. A module can contains functions, classes and variables.

Advantages of Python Modules.
1 Each and every module is independent unless linked explicitly. So, any changes made in one module will not affect another module.
2 Modules provide the reusability of code. So, a single module can be reused in another application (if required) or stored for later purposes.

Creating a Module:
Example:MyMod.py
```
def MyFun1():
  print("Welcome to Python & Advance")
  print("Modular Programming")
  return()
def MyFun2():
  print("It is Perfect Feature")
  print("Thank U ")
  return()
```

How To Use Modules in Python?
The import Keyword
You can use any Python source file as a module by executing an import statement in some other Python source file.

Syntax:
```
import module1[, module2[,... moduleN]
```

Example:moduletwo.py
```
import MyMod
MyMod.MyFun1()
MyMod.MyFun2()
print("END")
```

Import with renaming OR module aliasing
We can import a module by renaming it.

Syntax:
```
import ModName as NewModName
```

Example:modulethree.py
```
def add(a,b):
  return a+b
def sub(a,b):
  return a-b
def prod(a,b):
  return a*b
```

Example:modulefour.py
```
import modulethree as mt
Sum=mt.add(10,20)
print("Sum of the numbers is: " ,Sum)
```

```
dif=mt.sub(20,10)
print("Difference of the numbers is: " ,dif)
prd=mt.prod(10,10)
print("Difference of the numbers is: " ,prd)
```

The from...import Statement
Python's from statement lets you import specific attributes from a module into the current namespace.

Syntax:

```
from modname import name1[, name2[, ... nameN]]
```

Example:mfive.py

```
a="BigData"
b="Hadoop"
c="Spark"
d="Data Science"
print(a,b,c,d)
print(a,d)
```

```
# Grab the whole module:
>>>import mfive (Everything Imported to RAM)

 # Access its attributes
>>>mfive.a, mfive.c
('Big Data','Spark')

# Copy multiple names out
from mfive import a, b, c
print(b, c)
```

Using wildcard imports (from … import *)
It is also possible to import all properties from a module into the current namespace.

Syntax:

```
from modname import *
```

Example:

```
def Add1(a,b):
    return(a+b)
def Sub1(a,b):
    return(a-b)
def Pro1(a,b):
    return(a*b)
def Div1(a,b):
    return(a/b)
def FDiv1(a,b):
    return(a//b)
```

Example:

```
from MyProg1 import *
print(Add1(1,2))
print(Sub1(1,2))
print(Pro1(20,2))
```

Python Builtin Modules:

Python has a large number of built-in modules. These modules can be used in Python programs by simply importing them using their name followed by the keyword 'import'

Module	Description
math	Math module is used to perform mathematical operations.
random	This module is used to generate random numbers.
Operator	This module provides a set of pre-defined functions
decimal	This module is used to print the complete decimal value
string	string module provides a set of functions & operations on characters.

Math Module in PYTHON:
Python Number abs() Function:
It returns absolute value of x - the (positive) distance between x and zero.

Syntax
```
abs( x )
```

Example:
```
print(abs(-45))
print(abs(100.12))
```

ceil() Method:
It returns ceiling value of x, always returns forward round value.

Syntax
```
math.ceil( x )
```

EXAMPLE:
```
import math
print(math.ceil(-45.17))
print (math.ceil(100.12))
print (math.ceil(100.72))
```

Python Number floor() Method
It returns floor of x ,always returns backward round value.

Syntax
```
math.floor( x )
```

Example:
```
import math
print(math.floor(-45.17))
print(math.floor(100.12))
print(math.floor(100.72))
```

math.fabs(x)
Return the absolute value of x.

Syntax:
```
math.fabs(x)
```

Example:

```
import math
print(math.fabs(-10.99))
print(math.fabs(10.99))
print(math.fabs(-100.01))
```

math.factorial(x)
Return x factorial. Raises ValueError if x is not integral or is negative.

Syntax:

```
math.factorial(x)
```

Example:

```
import math
print(math.factorial(5))
```

math.fsum(iterable)
Return an accurate floating point sum of values in the iterable.

Syntax:

```
math.fsum()
```

Example:

```
import math
print(math.fsum([5,7,2,4]))
print(math.fsum({5,7,2,4}))
print(math.fsum((5,7,2,4)))
```

pow() Method: This method returns value of x to the power of y.

Syntax:

```
math.pow(x,y)
```

Example:

```
import math
print(math.pow(100, 2))
print(math.pow(2, 4))
print(math.pow(3, 0))
```

Python Number round() Function
It returns x rounded to n digits from the decimal point.

Syntax

```
round( x [, n]  )
```

Example:

```
print(round(10.234,2))#10.23
print(round(10.236,2))#10.24
print(round(10.236,3))#10.236
print(round(10.990,1))#11.0
print(round(80.23456, 2))
print(round(100.5623, 3))
```

sqrt() Method
It returns the square root of x for x > 0.

Syntax
math.sqrt(x)

Example:
```
import math
print(math.sqrt(100))
print(math.sqrt(7))
```

Random Module in PYTHON
Working with Random Numbers in Python
The random.random() method returns a random float in the interval [0.0, 1.0). This means the returned random number will always be smaller than the right-hand endpoint (1.0). This is also known as a semi-open range.

Syntax:
```
random.random()
```

Example:
```
import random
print(random.random())
print(random.random())
```

Generating Random Ints Between x and y
This is how you can generate a random integer between two endpoints in Python with the random.randint() function. This spans the full [x, y] interval and may include both endpoints:

Syntax:
```
random.randint(x,y)
```

Example:
```
import random
print(random.randint(1,10))
print(random.randint(1,10))
```

randrange():
This method you can exclude the right-hand side of the interval, meaning the generated number always lies within [x, y) and it will always be smaller than the right endpoint:

Syntax:
```
random.randrange(x,y)
```

Example:
```
import random
print(random.randrange(1,10))
print(random.randrange(1,10))
```

Generating Random Floats Between x and y
If you need to generate random float numbers that lie within a specifc [x, y] interval you can use the random.uniform function:

Syntax:
```
random.uniform(x,y)
```

Example:

```
import random
print(random.uniform(1,20))
print(random.uniform(1,20))
```

Picking a Random Element From a List
To pick a random element from a non-empty sequence (like a list or a tuple) you can use Python's random.choice method.

Syntax:
```
random.choice(seq)
```

Example:
```
import random
x=[1,2,3,4,5]
print(random.choice(x))#1
print(random.choice(x))#3
print(random.choice(x))#5
```

Example:
```
import random
items=[1,2,3,4,5,6,7]
print(random.choice(items))
print(random.choice(items))
print(random.choice(items))
```

Randomizing a List of Elements
You can randomize a sequence in place using the random.shuffle function. This will modify the sequence object and randomize the order of elements:

Syntax:
```
random.shuffle(x)
```

Example:
```
import random
items=[1,2,3,4,5,6,7]
print(items)
random.shuffle(items)
print(items)
```

Picking n Random Samples From a List of Elements
To pick a random sample of n unique elements from a sequence, use the random.sample function. It performs random sampling without replacement:

Syntax:
```
random.sample(population,k)
```

Example:
```
import random
items=[1,2,3,4,5,6,7]
print(items)
print(random.sample(items,3))
print(random.sample(items,3))
print(random.sample(items,3))
```

Operator Module in PYTHON

This module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, operator.add(x, y) is equivalent to the expression x+y.

```
Operation       Syntax          Function
Addition                a + b           add(a, b)
Concatenation   seq1 + seq2     concat(seq1, seq2)
Containment Test        obj in seq      contains(seq, obj)
Division                a / b           truediv(a, b)
Division                a // b          floordiv(a, b)
Bitwise And     a & b           and_(a, b)
Bitwise Exclusive Or a ^ b              xor(a, b)
Bitwise Inversion       ~ a             invert(a)
Bitwise Or      a | b           or_(a, b)
Exponentiation  a ** b          pow(a, b)
Identity                a is b          is_(a, b)
Identity                a is not b              is_not(a, b)
```

Example:

```
import operator
a=int(input("Enter Any Valid  Number: "))
b=int(input("Enter Any Valid  Number: "))
print(operator.lt(a, b))
print(operator.le(a, b))
print(operator.eq(a, b))
print(operator.ne(a, b))
print(operator.ge(a, b))
print(operator.gt(a, b))
```

Example

```
a=int(input("Enter Any Valid  Number: "))
b=int(input("Enter Any Valid  Number: "))
print(operator.__lt__(a, b))
print(operator.__le__(a, b))
print(operator.__eq__(a, b))
print(operator.__ne__(a, b))
print(operator.__ge__(a, b))
print(operator.__gt__(a, b))
```

Decimal Module in PYTHON:
This decimal module provides support for fast correctly-rounded decimal floating point arithmetic. It offers several advantages over the float datatype.

Example:

```
from decimal import Decimal
x = 1 / 3
print(type(x))
print(x)
y = Decimal(1) / Decimal(3)
print(type(y))
print(y)
```

Python compare floating point values:
Caution should be exercised when comparing floaing point values. While in many real world problems a small error is negligible, financial and monetary calculations must be exact.

Example:

```
from decimal import Decimal
x = 0.1 + 0.1 + 0.1
print(x == 0.3)
print(x)

print("----------------------")
x = Decimal('0.1') + Decimal('0.1') + Decimal('0.1')
print(x == Decimal('0.3'))
print(float(x) == 0.3)
print(x)
```

Sting module in PYTHON:
Python String module contains some constants, utility function, and classes for string manipulation

Example:

```
import string
print(string.ascii_letters)
print(string.ascii_lowercase)
print(string.ascii_uppercase)
print(string.digits)
print(string.hexdigits)
print(string.punctuation)
```

Working with Packages in Python
It is collection of modules.
OR
Any folder or directory which contains __init__.py file , then we call that folder or directory as a Python Package

PIC: PackageModuleStructure:

Importing module from a package:
We can import modules from packages using the dot (.) operator. For example, if want to import the start module from above Structure:

Syntax:

```
import Package.ModuleName.FunName
```

Creating a pakage(As per Live Project)
1 Goto PYTHON Installed location.

```
>>> import os
>>> os.getcwd()
C:\\Users\\admin\\AppData\\Local\\Programs\\Python\\Python38-32
```

2 Create a Directory/Folder and Provide any name (MyPackage)
3 Open Notepad save with, __init__ .py file in the directory (PYTHON recognize a Package)
4. Create a python file in that Package (MyDemo.py)

Example:

```
def MyFun():
   print("Welcome to Packages")
   print("Packages Are Dirs/Folders")
   return
MyFun()
```

Save with MyDemo.py Extension..!!

Executing a Module from a Package:

```
>>> import MyPackage.MyDemo
>>> MyPackage.MyDemo.MyFun()
>>> import MyPackage.MyDemo as nit
>>> nit.MyFun()
>>> from MyPackage.MyDemo import MyFun
>>> MyFun()
```

Creating SubPackage:
1. Create a Folder/Directory inside existing package and Provide A name (SubPackage)
2. Create __init__ .py file in the directory (PYTHON recognize as Package)
3. Create a python file in that (MyDemo.py)

Executing a Module from a Sub-Package:

```
>>> import MyPackage.SubPackage.MyDemo
>>> MyPackage.SubPackage.MyDemo.MyFun()
>>> import MyPackage.SubPackage.MyDemo as nit
>>> nit.MyFun()
>>> from MyPackage.SubPackage.MyDemo import MyFun
>> MyFun()
```

Popular Python Packages:
1. Requests
2. wxPython
3. BeautifulSoup
4. Twisted
5. boto3 for AWS
6. Python Imaging Library (PIL)
7. MoviePy
8. Pywin32
9. Pytest
10. Pendulum for Date&Time

eval()
eval Function take a String and evaluate the Result.

Syntax:

```
eval(expression)
```

Parameters
expression - this string as parsed and evaluated as a Python expression

Example:

```
x=eval("18-90+87")
print(x)
```

Example:

```
x=eval(input("Enter Expression: "))
print(x)
```

Example:

```
PyData=eval(input("Enter Data: "))
```

```
print(type(PyData))
print(PyData)
```