What is constructor?
Constructor is a special method or special instance method. This method is used to initialize object (OR) this method is used to define properties/instance variables/attributes/object level variables of object.
Constructor is a instance method.
The method is executed automatically whenever object of class is created.
Constructor is a magic method.

Syntax:
def **__init__(**self,arg1,arg2,..):
    statement-1
    statement-2

Constructor can be defined,
1. With arguments
2. Without arguments

Example:
```
class A:
   def __init__(self):
      print("object is created....")



def main():
   obj1=A()
   obj2=A()

main()
```

Output:
```
========= RESTART: F:/python6pmaug/ooptest5.py ========
object is created....
object is created....
```

Example of constructor without argument
```
class Product:
   def __init__(self): # constructor method
```

```
        self.pname=None
        self.price=None

def main():
    prod1=Product()
    print(prod1.pname,prod1.price)
    comp1=complex()
    print(comp1.real,comp1.imag)

main()
```

Output:

```
None None
1.0    0.0
```

Example of constructor with arguments:

```
class Product:
   def __init__(self,pn,p):
        self.pname=pn
        self.price=p


def main():
    prod1=Product("Monitor",5000)
    print(prod1.pname,prod1.price)
    comp1=complex(1.2,1.5)
    print(comp1.real,comp1.imag)

main()
```

Output:

```
Monitor 5000
1.2 1.5
```

Example:

```
# write a program to read the details of n products
# and display

class Product:
    def __init__(self,pname,price):
        self.pname=pname
```

```
        self.price=price

def main():
    n=int(input("Enter how many products?"))
    productList=[]
    for i in range(n):
        pname=input("Enter Product Name")
        price=float(input("Enter Price"))
        p=Product(pname,price)
        productList.append(p)

    for p in productList:
        print(f'{p.pname}--->{p.price}')

main()
```

Output:

```
======== RESTART: F:/python6pmaug/ooptest8.py ========
Enter how many products?2
Enter Product NameMonitor
Enter Price5000
Enter Product NameMouse
Enter Price100
Monitor--->5000.0
Mouse--->100.0
```

Example:

```
# constructor with default arguments
class Date:
    def __init__(self,d=0,m=0,y=0):
        self.dd=d
        self.mm=m
        self.yy=y

def main():
    d1=Date()
    print(d1.dd,d1.mm,d1.yy)
    d2=Date(12,12,2022)
    print(d2.dd,d2.mm,d2.yy)
```

```
main()
```

Output:

```
========= RESTART: F:/python6pmaug/ooptest9.py ========
0 0 0
12 12 2022
```

Advantage of encapsulation is data hiding. Data hiding is preventing data access from unrelated operations or functions. This allows developing secured applications.

C++ → private, protected, public
Java → private, protected, public
Python → __ (private), _(procted), No underscore (public)

private, public and protected are called access specifiers/modifiers. These access modifiers defines the accessibility of members of the class.

Any member which is prefix with __ is called private member. Private members are accessible within class or members of same class but cannot accessible by outside members.

Example:

```
class A:
    def __m2(self):
        print("private method")
    def m1(self):
        print("public method")
        self.__m2()

def main():
    obj1=A()
    obj1.m1()

main()
```

Output:

```
======== RESTART: F:/python6pmaug/ooptest11.py ========
public method
private method
```

Example:

```
class A:
    def __init__(self):
        self.__x=100 # private variable
        self.y=200 # public variable


def main():
    obj1=A()
    #print(obj1.__x)
    print(obj1.y)

main()
```

Output:

```
======== RESTART: F:/python6pmaug/ooptest12.py ========
200
```

Abstraction

Abstraction is the process of hiding the internal details of an application from the outer world.

Real life example of Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM.
Note: Data abstraction can be used to provide security for the data from the unauthorized methods.

Hidden data is operated is used public methods.

Example:

```
class Employee:
    def __init__(self):
        self.__empno=None # piv
        self.__ename=None # piv
```

```
        self.__salary=None # piv
    def print_employee(self): # public instance method
        print(f'EmployeeNo {self.__empno}')
        print(f'EmployeeName {self.__ename}')
        print(f'Salary {self.__salary}')


def main():
    emp1=Employee()
    emp1.print_employee()

main()
```

Output:

```
======== RESTART: F:/python6pmaug/ooptest13.py ========
EmployeeNo None
EmployeeName None
Salary None
```

The methods defined inside class perform two operations.

1. Setter operation
2. Getter operation

Setter operation is an operation which modifies values of the object.
Getter operation is an operation which read values of the object.

Example:

```
class Complex:
    def __init__(self):
        self.__real=0.0 # private instance variable
        self.__imag=0.0 # private instance variable
    def set_real(self,r):
        self.__real=r # instance variable
    def set_imag(self,i):
        self.__imag=i
    def get_real(self):
        return self.__real
    def get_imag(self):
        return self.__imag
```

```
def main():
    comp1=Complex()
    comp1.set_real(1.5)
    comp1.set_imag(2.5)
    print(comp1.get_real(),comp1.get_imag())
    comp1.set_real(1.2)
    print(comp1.get_real(),comp1.get_imag())
main()
```

Output:

```
======== RESTART: F:/python6pmaug/ooptest14.py ========
1.5 2.5
1.2 2.5
```