In [ ]:
```
## Todays Agenda!
#oops
# Class
#object
#constructor
#Inheritance
```

In [ ]:
```
# class ia a blueprint of an object
# class is a collection of variables(attributes) and methods!
# oops provide better security,modularity,extensibulity features of real wor
```

In [ ]:
```
#Syntax of class
class sample_class:
    list of variables
    list of methods

# calling a class
#classname.variablename
#classname.methodname
```

In [4]:
```
#Example
class simple:
    print('welcome object oriented programming')

    def sim():
        print('this is my function')
simple.sim()
print("this is my class name",simple.__name__)
```

```
welcome object oriented programming
this is my function
this is my class name simple
```

In [5]:
```
class simple:
    'welcome object oriented programming'

    def sim():
        print('this is my function')
simple.sim()
print("this is my class name",simple.__name__)
print("this is my Doc string",simple.__doc__)
```

```
this is my function
this is my class name simple
this is my Doc string welcome object oriented programming
```

In [ ]:
```
#Object
- object is an instance of a class
```

In [12]:
```
class vehicle:
    company="tayota"
    brand="Inova"

    def vehicle_brand(self):
        print('company')
    def vehicle_company(self):
        print('brand')
obj=vehicle()
obj.company
obj.brand
obj.vehicle_brand()
obj.vehicle_company()
```

```
company
brand
```

In [29]:
```
class operations:
    "Arthematic operations"
    a=10
    b=20
    def add(self):
        print("Addition of %d+%d=%d"%(self.a,self.b,self.a+self.b))
    def sub(self):
        print("Substraction of %d-%d=%d"%(self.a,self.b,self.a-self.b))
obj=operations()
obj.add()
obj.sub()
```

```
Addition of 10+20=30
Substraction of 10-20=-10
```

In [ ]:
```
#constructor
#at the time of object creation
 -syntax
 def __int__(self):
        print("statements")
```

In [28]:
```
class Arthametic_operations():
    "Symbols +,-,/,%,//,**"
    a=int(input('Enter A value:'))
    b=int(input('Enter B value:'))
    def __int__(self,a,b):
        self.a=a
        self.b=b
    def addition(self):
         print("Addition of %d+%d=%d"%(self.a,self.b,self.a+self.b))
    def substraction(self):
         print("substraction of %d-%d=%d"%(self.a,self.b,self.a-self.b))
obj=Arthametic_operations()
obj.addition()
obj.substraction()
```

```
Enter A value:30
Enter B value:20
Addition of 30+20=50
substraction of 30-20=10
```

In [39]:
```
#global values in constructor
class calc:
    x=int(input('Enter x value:'))
    y=int(input('Enter y value:'))
    def __int__(self,x,y):
        self.x=x
        self.y=y

    def ad(self,z):
        self.z=z
        print("Addition of %d + %d + %d = %d "%(self.x, self.y, self.z, self
z=int(input("Enter z value:'"))
obj=calc()
obj.ad(z)
```

```
Enter x value:40
Enter y value:40
Enter z value:'30
Addition of 40 + 40 + 30 = 110
```

In [ ]:
```
# inheritance
 - derived class access the properties of base class
#types
--->single level
--->multi level
--->multiple
--->hybrid inheritance
```

In [41]:
```python
#single level inheritance
#parent to child
class A:
    p=10
    q=10

    def addi(self):
        print("addition of two nuum:",self.p+self.q)
class B(A):
    p=10
    q=10

    def sub(self):
        print("substraction of two nuum:",self.p-self.q)

addition=B()
addition.addi()
addition.sub()

```

```
addition of two nuum: 20
substraction of two nuum: 0
```

In [ ]:
```python
# multilevel inheritance
# class A--->class B--->class C
```

In [43]:

```
#single level inheritance
#parent to child
class A:
    p=10
    q=10

    def addi(self):
        print("addition of two nuum:",self.p+self.q)
class B(A):
    r=10
    s=10

    def sub(self):
        print("substraction of two nuum:",self.r-self.s)

class C(B):
    t=50
    u=100
    def mul(self):
        print("substraction of two nuum:",self.t*self.u)

ob=C()
ob.addi()
ob.sub()
ob.mul()
ob.r
ob.p
ob.u
```

```
addition of two nuum: 20
substraction of two nuum: 0
substraction of two nuum: 5000
```

Out[43]:

```
100
```

In [58]:
```
#hierarical inheritance
# one parent class-->two different child classes
class A:
    print("This is class A")

    def aclass(self):
        print("A")
class B(A):
    print("This is class B")

    def bclass(self):
        print("B")
class C(A):
    print("This is class C")

    def cclass(self):
        print("C")

obj=C()
obj.aclass()
obj.cclass()
o=B()
o.bclass()
o.aclass()
```

```
This is class A
This is class B
This is class C
A
C
B
A
```

In [60]:
```
# multiple inheritance
# two  base classes-->one derived class
class A:
    def a_class(self):
        print("this is A class")
class B:
    def b_class(self):
        print("this is B class")
class C(A,B):
    def c_class(self):
        print("this is C class")
obj=C()
obj.a_class()
obj.b_class()
obj.c_class()
```

```
this is A class
this is B class
this is C class
```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```