In [1]:
```python
#By using one obj reference if we perform any change to the instance variables,the change wont be ef
fected in the remainning objects...
class Test:
    def __init__(self):
        self.x=10
t1=Test()
t2=Test()
print(t1.x,t1.x)
t1.x=888
print(t1.x,t2.x)
```
```
10 10
888 10
```

In [4]:
```python
class Test:
    def __init__(self):
        self.a=1000
        self.b=2000
    def m1(self):
        self.c=3000
        self.d=4000
t1=Test()
t1.m1()
print(t1.__dict__)
print(t1.a,t1.b,t1.c,t1.d)
```
```
{'a': 1000, 'b': 2000, 'c': 3000, 'd': 4000}
1000 2000 3000 4000
```

In [5]:
```python
#Declaring instance variable outside aclass using instance variable
class SE:
    def __init__(self,name,age,tech):
        self.name=name
        self.age=age
        self.tech=tech
s1=SE('Durga',50,'Java')
s2=SE('Mayank',22,'python')
s1.gf='sunny'
s1.gf='Mallika'
s2.brand='RC'
s2.brand='kF'
print(s1.__dict__)
print(s2.__dict__)
```
```
{'name': 'Durga', 'age': 50, 'tech': 'Java', 'gf': 'Mallika'}
{'name': 'Mayank', 'age': 22, 'tech': 'python', 'brand': 'kF'}
```

In [8]:
```python
#How to remove instance variable from an object
class Test:
    def __init__(self):
        self.a=10
        self.b=20
        self.c=30
        self.d=40
t1=Test()
t2=Test()
del t1.c
del t2.d
print('t1:',t1.__dict__)
print('t2:',t2.__dict__)
```
```
t1: {'a': 10, 'b': 20, 'd': 40}
t2: {'a': 10, 'b': 20, 'c': 30}
```

In [9]:
```python
#Static variable
class Test:
    a=10 #static variable
    def __init__(self):
        self.b=20
t=Test()
print(t.a,t.b) #accessing by using object name
print(Test.a,t.b) #accessing by using class name
```
```
10 20
10 20
```

In [10]:
```python
class Test:
    a=10
    def __init__(self):
        self.b=20
t1=Test()
t2=Test()
print("t1:",t1.a,t1.b)
print("t2:",t2.a,t2.b)
Test.a=888
t1.b=999
print("t1:",t1.a,t1.b)
print("t2:",t2.a,t2.b)
```
```
t1: 10 20
t2: 10 20
t1: 888 999
t2: 888 20
```

In [12]:
```python
#Various places to declare satic variables
class Test:
    a=10
    def __init__(self):
        self.b=20
        Test.c=30
    def m1(self):
        self.d=40
        Test.e=50
    @classmethod
    def m2(cls):
        cls.f=60
        Test.g=70
    @staticmethod
    def m3():
        Test.h=80
t=Test()
t.m1()
Test.m2()
Test.m3()
Test.i=90
print(Test.__dict__)
print(t.a,t.c,t.e,t.f,t.g,t.h,t.i)
```
```
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x00000296A9446C80>,
'm1': <function Test.m1 at 0x00000296A9446BF8>, 'm2': <classmethod object at 0x00000296A945B908
>, 'm3': <staticmethod object at 0x00000296A945BAC8>, '__dict__': <attribute '__dict__' of 'Tes
t' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__': None, 'c': 3
0, 'e': 50, 'f': 60, 'g': 70, 'h': 80, 'i': 90}
10 30 50 60 70 80 90
```

In [14]:
```python
class Test:
    x=10
    def __init__(self):
        self.y=20
t1=Test()
t2=Test()
t1.x=t1.x+1
t2.y=t2.y+1
print('t1:',t1.x,t1.y)
print('t2:',t2.x,t2.y)
print('Test:',Test.x)
```
```
t1: 11 20
t2: 10 21
Test: 10
```

In [16]:
```python
class Test:
    a=10
    def __init__(self):
        self.b=20
    @classmethod
    def m1(cls):
        cls.a=888
        cls.b=999
t1=Test()
t2=Test()
t1.m1()
print(t1.a,t1.b)
print(t2.a,t2.b)
print(Test.a,Test.b)
```
```
888 20
888 20
888 999
```

In [17]:
```python
#local variable
class Test:
    def m1():
        i=0 #local variable
        while i<10:
            print("Hello")
            i=i+1
Test.m1()
```
```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

In [19]:
```python
#Instance Methods
class Student:
    def __init__(self,name,marks):
        self.name=name
        self.marks=marks
    def display(self):
        print("Hi",self.name)
        print("Your marks are:",self.marks)
    def grade(self):
        if self.marks>=60:
            print("You got first grade")
        elif self.marks>=50:
            print("You got second grade")
        elif self.marks>=35:
            print("You got third grade")
        else:
            print("You are failed")
n=int(input("Enter no of students"))
for i in range(n):
    name=input("Enter student name:")
    marks=int(input("Enter student marks:"))
s=Student(name,marks)
s.display()
s.grade()
print()
```
```
Enter no of students1
Enter student name:mayank
Enter student marks:99
Hi mayank
Your marks are: 99
You got first grade
```

In [21]:
```python
#Getter and Setter Method
class Student:
    def setName(self,name):
        self.name=name
    def getName(self):
        return self.name

    def setMarks(self,marks):
        self.marks=marks
    def getMarks(self):
        return self.marks
n=int(input("Enter no of student:"))
for i in range(n):
    s=Student()
    name=input("Enter name of student:")
    s.setName(name)
    marks=int(input("Enter marks of student:"))
    s.setMarks(marks)
print('Hi',s.getName())
print('Your marks are:',s.getMarks())
print()
```
```
Enter no of student:1
Enter name of student:mayank
Enter marks of student:88
Hi mayank
Your marks are: 88
```

In [24]:
```python
#class method
#def m1(cls):
#  cls.x
class Animal:
    legs=4
    @classmethod
    def walks(cls,name):
        print("{} walks with {} legs".format(name,cls.legs))
Animal.walks('Dog')
```
```
Dog walks with 4 legs
```

In [27]:
```python
#program to track the no of objects created for a class
class Test:
    count=0
    def __init__(self):
        Test.count +=1

    @classmethod
    def noofobjects(cls):
        print('The no of objects created:',cls.count)
t1=Test()
t2=Test()
Test.noofobjects()
```
```
The no of objects created: 2
```

In [28]:
```python
#static method
class Durgamath:
    @staticmethod
    def add(x,y):
        print("The sum is:",x+y)
    @staticmethod
    def product(x,y):
        print("The product is:",x*y)
    @staticmethod
    def average(x,y):
        print("The average is:",(x+y))
Durgamath.add(10,20)
Durgamath.product(10,20)
Durgamath.average(10,20)
```
```
The sum is: 30
The product is: 200
The average is: 30
```

In [31]:
```python
#passing members of one class to another class
class Employee:
    def __init__(self,eno,ename,esal):
        self.eno=eno
        self.ename=ename
        self.esal=esal
    def display(self):
        print("Employee Number:",self.eno)
        print("Employee Name:",self.ename)
        print("Employee Salary",self.esal)
class Test:
    def modify(emp):
        emp.esal=emp.esal+1000
        emp.display()
e=Employee(100,'Durga',10000)
Test.modify(e)
```
```
Employee Number: 100
Employee Name: Durga
Employee Salary 11000
```

In [32]:
```python
#Inner classes/Nested class
class Outer:
    def m1(self):
        print("outer class method")
    class Inner:
        def m2(self):
            print("Inner class method")
o=Outer()
o.m1()
i=o.Inner()
i.m2()
```
```
outer class method
Inner class method
```

In [36]:
```python
class Person:
    def __init__(self):
        self.name='Durga'
        self.db=self.Dob()
    def display(self):
        print('Name:',self.name)
    class Dob:
        def __init__(self):
            print("Dob class constructor")
            self.dd=28
            self.mm=5
            self.yy=1947
        def display(self):
            print('Dob={}/{}/{}'.format(self.dd,self.mm,self.yy))
p=Person()
p.display()
p.db.display()
```
```
Dob class constructor
Name: Durga
Dob=28/5/1947
```

In [38]:
```python
class Human:
    def __init__(self):
        self.name='sunny'
        self.head=self.Head()
        self.brain=self.Brain()
    def display(self):
        print('Hello',self.name)
    class Head:
        def talk(self):
            print('Talking...')
    class Brain:
        def think(self):
            print("Thinking...")
h=Human()
h.display()
h.head.talk()
h.brain.think()
```
```
Hello sunny
Talking...
Thinking...
```

In [39]:
```python
#Garbage collector
#how to enable or disable garbage collector?
#gc module
#gc.isenabled()    True/False
#gc.disable()
#gc.enable()
import gc
gc.disable()
print(gc.isenabled())
```
```
False
```

In [40]:
```python
#Destructor
#__del__()
import time
class Test:
    def __init__(self):
        print('Object Initialization')
    def __del__(self):
        print("Fulfilling last wish and performing cleanup activities")
t1=Test()
t1=None #variable will be there we can use for the next level
time.sleep(5)
print("End of apps")
```
```
Object Initialization
Fulfilling last wish and performing cleanup activities
End of apps
```

In [44]:
```python
import time
class Test:
    def __init__(self):
        print("Constructor Execution")
    def __del__(self):
        print("Destructor Execution")
l1=[Test(),Test(),Test()]
time.sleep(8)
del l1
time.sleep(7)
print("End of Application")
```
```
Constructor Execution
Constructor Execution
Constructor Execution
Destructor Execution
Destructor Execution
Destructor Execution
End of Application
```

In [ ]:
```python
#Inheritance
#How we can use members of one class inside another class
#1. By using composition (Has A RElationship)
#2. By using inheritance
```

In [46]:
```python
#By using composition
class car:
    def __init__(self,name,model,color):
        self.name=name
        self.model=model
        self.color=color
    def getinfo(self):
        print("car name : {}, model :{} and color:{}".format(self.name,self.model,self.color))
class Employee:
    def __init__(self,ename,eno,car):
        self.ename=ename
        self.eno=eno
        self.car=car #using car reference we can access functionality
    def empinfo(self):
        print("Employee name:",self.ename)
        print("Employee no",self.eno)
        print("Employee car info:")
        self.car.getinfo()
c=car('Innova','2.5z','Grey')
e=Employee('Durga',100,c)
e.empinfo()
```
```
Employee name: Durga
Employee no 100
Employee car info:
car name : Innova, model :2.5z and color:Grey
```

In [49]:
```python
#By using Inheritance
# If you define any variable to parent it is already available to child class
class X:
    a=10
    def m1(self):
        print("parent class inheritance method")
    @classmethod
    def m2(cls):
        print("parent class class method")
    @staticmethod
    def m3():
        print("parent classs static method")
    def __init__(self):
        self.b=8888
        print("Parent constructor")

class Y(X):
    pass
y=Y()
print(y.a)
y.m1()
y.m2()
y.m3()
print(y.b)
```
```
Parent constructor
10
parent class inheritance method
parent class class method
parent classs static method
8888
```

In [55]:
```python
#super() method
class X:
    a=10
    def m1(self):
        print("parent class instance variable")
    @classmethod
    def m2(cls):
        print("parent class static method")
    @staticmethod
    def m3():
        print("parent class static method")
    def __init__(self):
        print("parent constructor")
class Y(X):
    def __init__(self):
        a=7777
        print("child constructor...")
        super().m1()
        super().m2()
        super().m3()
        super().__init__()
        print(y.a)
y=Y()
```
```
child constructor...
parent class instance variable
parent class static method
parent class static method
parent constructor
10
```

In [6]:
```python
class person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def eatndrink(self):
        print("Briyani eating and beer drinking")
class SE(person):
    def __init__(self,name,age,eno,esal):
        super().__init__(name,age)
        self.eno=eno
        self.esal=esal
    def work(self):
        print("python coding is something like drinking chilled beer")
s=SE('Durga',48,100,1000)
print(s.name,s.age,s.eno,s.esal)
s.eatndrink()
s.work()
```
```
Durga 48 100 1000
Briyani eating and beer drinking
python coding is something like drinking chilled beer
```

In [1]:
```python
#we cannot access instance variable of parent class through super()
class P:
    def m1(self):
        self.a=10
class C(P):
    def __init__(self):
        super().m1()
        print(self.a)
C=C()
```
```
10
```

In [3]:
```python
class P:
    a=7777
    def __init__(self):
        self.b=10

class C(P):
    def __init__(self):
        self.b=20
        print('Before:',self.b)
        super().__init__()
        print('After:',self.b)
c=C()
```
```
Before: 20
After: 10
```

In [4]:
```python
#Inheritance
#1 Single
#The concept of inheriting members of one class to another class
class P:
    def m1(self):
        print("Parent method")
class C(P):
    def m2(self):
        print ("Child method")
c=C()
c.m1()
c.m2()
```
```
Parent method
Child method
```

In [5]:
```python
#Multi level
class GF:
    def m1(self):
        print('Land')
class F(GF):
    def m2(self):
        print('Cash')
class U(F):
    def m3(self):
        print('Enjoy')
c=U()
c.m1()
c.m2()
c.m3()
```
```
Land
Cash
Enjoy
```

In [6]:
```python
#Hierarchical Inheritance
class P:
    def m1(self):
        print("Parent")
class C1(P):
    def m2(self):
        print("child1")
class C2(P):
    def m3(self):
        print("child2")
c1=C1()
c1.m1()
c1.m2()

c2=C2()
c2.m1()
c2.m3()
```
```
Parent
child1
Parent
child2
```

In [7]:
```python
#Multiple inheritance
class Father:
    def height(self):
        print("Height is 6 feet")
class Mother:
    def color(self):
        print("brown color")
class Child(Father,Mother):
    pass
c=Child()
print("child inherited properties from parents")
c.height()
c.color()
```
```
child inherited properties from parents
Height is 6 feet
brown color
```

In [8]:
```python
class P1:
    def m1(self):
        print("p1 method")
class P2:
    def m1(self):
        print("p2 method")
class C(P2,P1):
    pass
c=C()
c.m1()
```
```
p2 method
```

In [9]:
```python
#Polymorphism
#poly- Many
#Morphs-Forms
#1 Duck type philosophy of python
class Duck:
    def talk(self):
        print("Quack Quack Quack")
class Dog:
    def talk(self):
        print("Bow Bow Bow")
class Cat:
    def talk(self):
        print("Meow Meow...")
class Goat:
    def talk(self):
        print("Myaah Myaah..")
l=[Duck(),Dog(),Goat(),Cat()]
for obj in l:
    obj.talk()
```
```
Quack Quack Quack
Bow Bow Bow
Myaah Myaah..
Meow Meow...
```

In [11]:
```python
#hasattr(obj,'attributename')  returns True or False
class Duck:
    def talk(self):
        print("Quack Quack...")
class Dog:
    def bark(self):
        print("Bow Bow Bow...")
class Cat:
    def talk(self):
        print("Meow Meow...")
class Goat:
    def talk(slef):
        print("Myaah Myaah...")
l=[Duck(),Dog(),Goat(),Cat()]
for obj in l:
    if hasattr(obj,'talk'):
        obj.talk()
    elif hasattr(obj,'bark'):
        obj.bark()
```
```
Quack Quack...
Bow Bow Bow...
Myaah Myaah...
Meow Meow...
```

In [12]:
```python
#operator overloading
class Book:
    def __init__(self,pages):
        self.pages=pages
    def __add__(self,other):
        return self.pages + other.pages
b1=Book(100)
b2=Book(200)
print("The total number of pages:",b1+b2)
```
```
The total number of pages: 300
```

In [14]:
```python
#over riding concept
class P:
    def Property(self):
        print("cash + land + gold + power")
    def marry(self):
        print("Appalamma")
class C(P):
    def marry(self):
        super().marry()
        print("Katrina")
c=C()
c.Property()
c.marry()
```
```
cash + land + gold + power
Appalamma
Katrina
```

In [16]:
```python
#Abstract Method:
from abc import *
class fruit:
    @abstractmethod
    def taste(self):
        pass
```

In [26]:
```python
from abc import *
class vehicle(ABC):
    @abstractmethod
    def getNoofWheels(self):
        pass
class Bus(vehicle):
    def getNoofwheels(self):
        return 7
class Auto(vehicle):
    def getNoofwheels(self):
        return 3
b=Bus()
print(b.getNoofwheels())
a=Auto()
print(a.getNoofwheels())
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-fb9d918c9c6d> in <module>
     10     def getNoofwheels(self):
     11         return 3
---> 12 b=Bus()
     13 print(b.getNoOfwheels())
     14 a=Auto()

TypeError: Can't instantiate abstract class Bus with abstract methods getNoofWheels
```

In [ ]: