Oop:

```
"""
The __init__ method
"""

class Computer:
    #attributes-> variables
    #behaviour-> methods
    def __init__(self,cpu,ram):
        #need to assign values to objects
        self.cpu=cpu
        self.ram=ram
    def config(self):
        print(self.cpu,self.ram)

com1=Computer("i5",16)
com2=Computer("Ryzen-3",8)
"""
Computer.config(com1)
Computer.config(com2)"""
com1.config()
com2.config()
```

Constructors and comparing objects

```
"""
Constructores and Comparing Objects
"""

class computer:
    def __init__(self):
        self.name="Balaji"
        self.age=23

    def compare(self,other):
        if self.age==other.age:return True
        else:return False

#address of memeory 1903586265936
c1=computer()
c1.age=32
c2=computer()
if c1.compare(c2):
    print("equal")
else:
    print("Not equal")

print(c1.name)
print(c2.name)
```

Types of Variables

```
"""
types of variables
variables are three types
1) class->common to all objcts
2) instance -> different for each object
```

```
"""
class Car:
    #Class variables
    wheels=4#common to all objects
    def __init__(self):
        #instance Variables
        self.mil=10
        self.com="BMW"

c1=Car()
c2=Car()

#c1.mil=21

print(c1.mil,c1.com,c1.wheels)
print(c2.mil,c2.com,c2.wheels)
print(Car.wheels)
#print(Car.mil) #enrror
```

Types of Methods

```
"""
Types of methods
methods are three types
1) class
2) instance
3) static -> which has no concern about instance or class variables



Accessors- get methods
mutators- set methods

Decorartors-calssmethod-to indicate its an classs method
"""
class Student:
    school="CTS"

    def __init__(self,m1,m2,m3):
        self.m1=m1
        self.m2=m2
        self.m3=m3

    def avg(self):#instance method
        return (self.m1+self.m2+self.m3)/3

    def get_m1(self):#Accessors
        return self.m1

    def set_m1(self,val):#mutators
        self.m1=val

    @classmethod#Decorartors
    def getSchool(cls):
        return cls.school

    @staticmethod
    def stat():
        print("This is a static method")
        return  0

s1=Student(12,12,33)
s2=Student(21,34,56)

print(s1.avg())
```

```
print(s2.avg())

print(s1.get_m1())
s1.set_m1(21)
print(s1.get_m1())

print(Student.getSchool())

print(Student.stat())#prints text and reurn 0 which is again printed
Student.stat()
```

Inner Class

```
"""
    inner class in python
    object of inner class should be inside of outer classi.e., line 10
"""

class Student:

    def __init__(self,name,rollno):
        self.name=name
        self.rollno=rollno
        self.lap=self.Laptop()##creation of object of inner class inside the outer
class(1)

    def show(self):#mtd of student
        print(self.name,self.rollno)
        self.lap.show()

    class Laptop:
        def __init__(self):
            self.ram="16gb"
            self.cpu="i3"
            self.disk="1TB"

        def show(self):#mtd of laptop
            print (self.ram,self.cpu,self.disk)

s1=Student("Balaji",23)
s2=Student("MSC",25)
s1.show()

# s1.lap.show()# anothe rway of accesing the attributes
# print(s2.lap.disk)

"""
    lap1=Student.Laptop()   To create object diresctly oustide (2)
    lap1.disk
    lap1.show()
"""

"""
    print(s1.name)
    print(s2.name) This wont lok good instead s1.show( need to print all details
about student

    s1.show()
    print(s1.lap.ram) #16gb one way of accessing inner class attributes
"""
"""
    To create two another different objects

    lap1=s1.lap
    lap2=s2.lap
```

```
    print(lap1.disk)#1TB
    lap2.show()#16gb i3 1TB
"""
```

Practice

```
"""
Practice
"""
class Dog:

    def __init__(self,name,level):
        self.name=name
        self.level=level
        self.dogPet=self.Pet()

    def eat(self):
        print(self.name,self.level)
        self.dogPet.info()

    class Pet:
        def __init__(self):
            self.type="P-kid"
            self.age="P-younger"
        def info(self):
            print(self.type,self.age)

"""
    #1
    o1=Dog.Pet()
    print(o1.age,o1.type,sep=",")
    o1.info()
"""
"""
    #2
    o2=Dog("MIckey","Mother")
    o2.dogPet.info()
"""

o3=Dog("jimmy","Kid")
o3.eat()
o3.dogPet.info()
```

Inheritance

```
"""
Inheritance

1)single
2)multilevel
3)multiple
"""
class A:
    def feature1(self):
        print("Feature 1 working")
    def feature2(self):
        print("feature 2 working")

class B: #Single inheritance
    def feature3(self):
        print("Feature 3 working")
    def feature4(self):
        print("feature 4 working")

class C:
    def feature5(self):
        print("Feature C working")
```

```
'''
class C(B):
    def feature5(self):
        print("Feature C working")'''

class D(A,B,C):#multiple
    pass

a1=A()
a1.feature1()
a1.feature2()
'''
b1=B()
b1.feature4()
b1.feature3()
b1.feature1()

c1=C()
c1.feature5()
'''
d1=D()
d1.feature1()
d1.feature3()
d1.feature5()
```

Constructor in Inheritance

```
'''
Constructor in Inheritance
Method resolution order(MRO)
Super constr


'''
"""
class A:

    def __init__(self):
        print("in a INIT")

    def feature1(self):
        print("Feature 1 working")
    def feature2(self):
        print("feature 2 working")

class B(A): #Single inheritance
    def feature3(self):
        print("Feature 3 working")
    def feature4(self):
        print("feature 4 working")

#a=A()
b=B()#since it is inherited rom A constrc of A is called when obj for B is created
output:in a INIT
"""
"""
class A:

    def __init__(self):
        print("in a INIT")

    def feature1(self):
        print("Feature 1 working")
    def feature2(self):
```

```
        print("feature 2 working")

class B(A): #Single inheritance
    def __init__(self):
        print("in B init")

    def feature3(self):
        print("Feature 3 working")
    def feature4(self):
        print("feature 4 working")

#a=A()
b=B()#since B has an init it wont call A init Instead it calls B init
output:
in B init

"""
'''
class A:

    def __init__(self):
        print("in a INIT")

    def feature1(self):
        print("Feature 1 working")
    def feature2(self):
        print("feature 2 working")

class B(A): #Single inheritance
    def __init__(self):
        super().__init__()
        print("in B init")

    def feature3(self):
        print("Feature 3 working")
    def feature4(self):
        print("feature 4 working")



#a=A()
b=B()#It calls first B init and calls A init as we used super()
output:
in a INIT
in B init

'''

class A:

    def __init__(self):
        print("in A INIT")

    def feature1(self):
        print("Feature 1 working")

class B: #Single inheritance
    def __init__(self):
        #super().__init__()
        print("in B init")

    def feature3(self):
        print("Feature 3 working")

class C(B,A):
    def __init__(self):
        #super(C, self).__init__()
        super().__init__()
```

```
        print("in C init")


#a=A()
b=C()#It calls first B and then C because it follows L->R(MRO)
output:
in B init
in C init
```

Constructors for methods

```
"""
MRO for methods
Super constr
super method

"""
'''
class A:

    def __init__(self):
        print("in A INIT")

    def feature1(self):
        print("Feature 1-A working")

class B: #Single inheritance
    def __init__(self):
        #super().__init__()
        print("in B init")

    def feature1(self):
        print("Feature 1-B working")

class C(B,A):
    def __init__(self):
        #super(C, self).__init__()
        super().__init__()
        print("in C init")


#a=A()
b=C()
b.feature1()
op:
in B init
in C init
Feature 1-B working
'''


class A:

    def __init__(self):
        print("in A INIT")

    def feature1(self):
        print("Feature 1-A working")
```

```
class B: #Single inheritance
    def __init__(self):
        #super().__init__()
        print("in B init")

    def feature1(self):
        print("Feature 1-B working")

class C(A,B):
    def __init__(self):
        #super(C, self).__init__()
        super().__init__()#super constructor
        print("in C init")

    def feat(self):
        super().feature1()#super method

c=C()
c.feature1()
c.feat()

# op:
# in A INIT
# in C init
# Feature 1-A working
```

Polymorphism

Duck Typing

```
"""
Duck Typing

Its not nede which class object is provided if it contains execute method
then we can pass that object(class) as parameter
it is not concerned that whic clas object it is but only we need execute
method in it

"""

class Pycharm:
    def execute(self):
        print("execution")
        print("Spell check")
        print("Color enhancement")

class Myeditor:
    def execute(self):
        print("execution")
        print("Spell check")
        print("Color enhancement")
        print("Also error detection")
        print("Error correction")

class Laptop:
    def code(self,ide):
```

```
        ide.execute()



ide=Myeditor()

lap=Laptop()
lap.code(ide)
```

Operator Overloading

```
"""
Operator Overloading

a=2
b=1
print(int.__add__(a,b))
#overloading + operator
#overloading > operator

"""
class Student:
    def __init__(self,m1,m2):
        self.m1=m1
        self.m2=m2

    def __add__(self, other):
        t1=self.m1+other.m1
        t2=self.m2+other.m2
        s3=Student(t1,t2)
        return s3

    def __gt__(self, other):
        if (self.m1+self.m2)>(other.m1+other.m2):
            return True
        else:
            return  False

    def __str__(self):
        return '{} {}'.format(self.m1,self.m2)
        #return '%d %d'%(self.m1,self.m2)

mark=Student(21,19)
mark2=Student(13,12)

tot=mark+mark2  #overloading + operator
print(tot.m2)

if mark>mark2:  #overloading > operator
    print("student 1 wins")
else:
    print("student 2 wins")

#print(mark) #printing address
print(mark2)#overloading __str__() mtd which is print method
```

Method Overloading

```
"""
Method OverLoading->two methods insde a sam eclassbut different arguments
Method OverRiding->
"""

class Student:
    def __init__(self,m1,m2):
        self.m1=m1
        self.m2=m2

    def sum(self,n1=None,n2=None,n3=None):#method overloading block
        if n1!=None and n2!=None and n3!=None:
            return n1+n2+n3
        elif n1!=None and n2!=None:
            return n1+n2
        else:
            return n1


o1=Student(1,2)
print(o1.sum(2))
```

Method OverRiding

```
"""
Method OverRiding
"""
class A:

    def show(self):
        print("in A Show")

class B(A):
    ######overridfing take splace like if i dont have show in B then A's
show will be called else if B has the B's show will be primted
    def show(self):
        print("inside B Show which was overriden A show method")

b=B()
b.show()
```