In [5]:
```
#oops= object oriented programming

#1 = class
#2 = object
#3 = encapsulation
#4 = data hiding
#5 = inheritance-single and multiple
#6 = polymorphism

#Python class -
# in class we use attributes and methods

```

Class and Objects

In [2]:
```
#Python class
class Employee:
    ename="Ankita"  #attribute
    dept="IT"       #attribute

    def emp_info(self):  #to write self is compulsary is in class
        print("I am",self.ename)
        print("My department is",self.dept)
e1=Employee() #object creation (e1). using object is imp
e1.emp_info()

```

```
I am Ankita
My department is IT
```

In [3]:
```
class calc:
    def add(self,a,b):
        return a+b
print("Enter two number:")
first=int(input())
second=int(input())

c1=calc()
res=c1.add(first,second)
print(res)


#que = addition of two numbers
```

```
Enter two number:
15
20
35
```

In [4]:
```
class calc:
    def add(self,a,b):
        return a+b
print("Enter two number:")
first=int(input())
second=int(input())

c1=calc()
res=c1.add(first,second)
print("\n"+str(first)+"+"+str(second)+"="+str(res))
```

```
Enter two number:
15
20

15+20=35
```

In [7]:
```
class calc:
    def multi(self,a,b):
        return a*b

    def sub(self,a,b):
        return a-b
print("Enter two number:")
first=int(input())
second=int(input())


c1=calc()
res=c1.multi(first,second)
print("Multiplication = ",res)

c1=calc()
res=c1.sub(first,second)
print("Substraction = ",res)


```

```
Enter two number:
20
15
Multiplication =  300
Substraction =  5
```

In [2]:
```
#constuctor = __init__

class Student:
    def __init__(self): #Constructor creation__init__
        self.sname=input("Enter Name of Student:")
        self.rollno=int(input("Enter Roll no:"))
    def display(self):
        print("Name: ",self.sname,"Roll No: ",self.rollno)
s1=Student()
s2=Student()
s3=Student()

s1.display()
s2.display()
s3.display()
```

```
Enter Name of Student:Aditya
Enter Roll no:101
Enter Name of Student:Ankita
Enter Roll no:102
Enter Name of Student:Suraj
Enter Roll no:103
Name:  Aditya Roll No:  101
Name:  Ankita Roll No:  102
Name:  Suraj Roll No:  103
```

In [3]:
```
class Student:
    def __init__(self): #Constructor creation__init__
        self.sname=input("Enter Name of Student:")
        self.rollno=int(input("Enter Roll no:"))
        self.display()

    def display(self):
        print("Name: ",self.sname,"Roll No: ",self.rollno)
s1=Student()
s2=Student()
s3=Student()
```

```
Enter Name of Student:Rutuja
Enter Roll no:111
Name:  Rutuja Roll No:  111
Enter Name of Student:Aditi
Enter Roll no:222
Name:  Aditi Roll No:  222
Enter Name of Student:Shiva
Enter Roll no:333
Name:  Shiva Roll No:  333
```

Inheritance

Single inheritance

In [2]:
```
#Single inheritance

class parents:
    def info(self):
        print("This Is Parent Class")
class Child(parents):
    def child_info(self):
        print("This is Child class")
c1=Child()
c1.info()
c1.child_info()


```

```
This Is Parent Class
This is Child class
```

In [23]:
```
class Parents:
    def cal(self,l,b):
        self.l=int(input("enter number"))
        self.b=int(input("enter number"))
        print("length and breadth")
class Child(Parents):
    def multi(self):
        area_rect=l*b
        return area_rect

a1=Child()

a1.cal()
a1.multi
print("Area of Rectangle:")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_6264/2905073650.py in <module>
     10
     11 a1=Child()
---> 12 a1.cal()
     13 a1.multi
     14 print("Area of Rectangle:")

TypeError: cal() missing 2 required positional arguments: 'l' and 'b'
```

In [9]:
```
class rectangle:
    def __init__(self,le,br):
        self.le=le
        self.br=br
class area(rectangle):
    def area(self):
        rect_area=le*br
        return rect_area
le=int(input("Enter length of rectangle:"))
br=int(input("Enter breadth of rectangle:"))
a1=area(le,br)
print("Area of Reactangle: ",a1.area())
```

```
Enter length of rectangle:6
Enter breadth of rectangle:5
Area of Reactangle:  30
```

Multiple Inheritance

In [2]:
```
class father:
    eyes="black"
    hair="black"

    def info(self):
        return "I am father class"
class mother:
    eyes="brown"
    hair="black-brown"
    nose="straight"

    def info(self):
        return "I am mother class"
class child(father,mother):

    def info(self):
        return "I am child class"
c1=child()
print(c1.eyes)
print(c1.info())
```

```
black
I am child class
```

Multilevel Inheritance

In [1]:
```
class Emp:
    name="Ankita"
    sal=60000
    city="Koparkhairane"

    def info(self):
        print(f"{self.name} is having {self.sal} and living in {self.city}")
class Edu_tax_info(Emp):
    deg="BSC"
    grade="A+"
    tax=6.5
class sal_calc(Edu_tax_info):
    def annual_inc(self):
        tot_sal=self.sal*self.tax
        return tot_sal
s1=sal_calc()
s1.info()
print("Income: ",s1.annual_inc())
```

```
Ankita is having 60000 and living in Koparkhairane
Income:  390000.0
```

In [17]:
```
class Circle:
    r=int(input("enter number r:"))
class Rect:
    l=int(input("enter number l:"))
    b=int(input("enter number b:"))
class Area(Rect,Circle):
    def area_circle(self):
        ac=3.14*self.r**2
        return print("Area of circle :",ac)
    def area_rect(self):
        ar=self.l*self.b
        return print("Area of Rectangle :",ar)
a1=Area()
a1.area_circle()
a1.area_rect()


```

```
enter number r:5
enter number l:10
enter number b:15
Area of circle : 78.5
Area of Rectangle : 150
```

Hierarchical Inheritance

In [25]:
```
class A:
    num1=10
    num2=20
class B(A):
    def mult(self):
        multi=self.num1*self.num2
        return multi
class C(A):
    def add(self):
        add=self.num1+self.num2
        return add
x1=C()
y1=B()
print(f"The Addition is {x1.add()}")
print(f"The Multiplication is {y1.mult()}")


```

```
The Addition is 30
The Multiplication is 200
```

In [22]:
```
#Super() function

class parent:
    def __init__(self):
        self.p_attri="I am parent"
    def p_method(self):
        print("back in my days...")
class child(parent):
    def __init__(self):
        super().__init__()
        self.c_attri="Im child"
c1=child()
print(c1.p_attri)
print(c1.c_attri)
c1.p_method()
```

```
I am parent
Im child
back in my days...
```

Polymorphism

13-04-2023

In [ ]:
```
1
```

In [1]:
```
class Product1:
    def __init__(self,name,price):
        self.name=name
        self.price=price
    def info(self):
        print(f"Product name is {self.name} and price is {self.price}")
    def status_info(self):
        print("In Stock")
class Product2:
    def __init__(self,name,price):
        self.name=name
        self.price=price
    def info(self):
        print(f"Product name is {self.name} and price is {self.price}")
    def status_info(self):
        print("Out of Stock")

p1=Product1("Mobile",40000)
p2=Product2("Laptop",80000)

for prods in (p1,p2):
    prods.status_info()
    prods.info()
    prods.status_info()

```

```
In Stock
Product name is Mobile and price is 40000
In Stock
Out of Stock
Product name is Laptop and price is 80000
Out of Stock
```

In [2]:
```
from math import *

class shape:
    def __init__(self,name):
        self.name=name
    def area(self):
        pass
    def fact(self):
        return "I am a two-dimantional Shape"
    def __str__(self):
        return self.name
class square(shape):
    def __init__(self,length):
        super().__init__("square")
        self.length=length
    def area(self):
        return self.length**2
    def fact(Self):
        return "Square has 4 equal sides"
class circle(shape):
    def __init__(self,radius):
        super().__init__("Circle")
        self.radius=radius
    def area(self):
        return pi*self.radius**2

a=square(4)
c=circle(7)
print(c)
print(c.fact())
print(a.fact())
print(c.area())

```

```
Circle
I am a two-dimantional Shape
Square has 4 equal sides
153.93804002589985
```

17-04-2023

Operator Overloading

In [4]:
```
class A:
    def __init__(self,x):
        self.x=x
    def __add__(self,other):    #magic method __add__
        return self.x+other.x

a1=A(1)
a2=A(2)
print(a1+a2)
```

```
3
```

In [6]:
```
class A:
    def __init__(self,x):
        self.x=x
    def __add__(self,other):    #magic method __add__
        return self.x+other.x

a1=A(1)
a2=A(2)

a3="Data"
a4=" Science"
print(a1+a2)
print(a3+a4)
```

```
3
Data Science
```

In [7]:
```
class Numbers:
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def __add__(self,o):
        return self.a+o.a,self.b+o.b
n1=Numbers(1,2)
n2=Numbers(2,3)

res=n1+n2
print(res)

```

```
(3, 5)
```

In [9]:
```
class check:
    def __init__(self,x):
        self.x=x
    def __gt__(self,o):
        if(self.x>o.x):
            return True
        else:
            return False
c1=check(10)
c2=check(20)
if(c1>c2):
    print("C1 is greater")
else:
    print("C2 is greater")
```

```
C2 is greater
```

In [ ]:
```

```