INHERITENCE

Single level inheritence

```
class  Above:
    i=5
    def fun1(self):
        print("parent class")
class Below(Above):
    i=10
    def fun2(self):
        print("Child class")
temp1=Below()
temp2=Above()
temp1.fun1()
temp1.fun2()
print(temp1.i)
print(temp2.i)
```

Output:

parent class
Child class
10
5

Hierarchical Inheritence

```
class Shape:
    def __init__(self,colour):
        self.colour=colour
        print(f"colour is {self.colour}")
class Rectangle(Shape):
    def __init__(self, length, width,colour):
        super().__init__(colour)
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
class Circle(Shape):
    def __init__(self,radius,colour):
        super().__init__(colour)
        self.radius=radius
    def area(self):
            return self.radius*3.14*self.radius
rect = Rectangle(10,5,"red")
cir=Circle(5,"yellow")
```

```
print("Area of Rectangle:", rect.area())
print("Area of Circle :",cir.area())
```

Output:

```
colour is red
colour is yellow
Area of Rectangle: 50
Area of Circle : 78.5
```

```

class Shape:
    def __init__(self,ht,wt):
        self.ht=ht
        self.wt=wt
    def area(self,ht,wt):
        return self.ht*self.wt
class Rectangle(Shape):
    pass
class Triangle(Shape):
    def area(self):
        print("Area of triangle is ",.5*self.ht*self.wt)
r=Rectangle(12,14)
print("Area of rectangle is",r.area(12,14))
t=Triangle(12,14)
t.area()
```

Output:

```
Area of rectangle is 168
Area of triangle is 84.0
```

Multiple Inheritence

```
class A:
    demo1=10
    def fun1(self):
        print(self.demo1)
class B:
    demo2=15
    def fun2(self):
        print(self.demo2)
class C(A,B):
    demo3=20
    def fun3(self):
        print(self.demo3)
c=C()
c.fun1()#methods can be accessed
c.fun2()
c.fun3()
print(c.demo1)#attributes can also be accessed.
```

Output:

```
10
15
20
10
```

Create a base class Employee with attributes like name and salary. Create a derived class Manager that adds a department and overrides a method to display full details.

```
#Create a base class Employee with attributes like name and salary. Create a derived class Manager that adds a department and overrides a method to display full details.
class Employee():
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary
    def display(self):
        print(self.name,"has a salary of ",self.salary)
class Manager(Employee):
    def __init__(self,name,salary,dept):
        super().__init__(name,salary)
        self.dept=dept
    def display(self):
        print(self.name,"is in ",self.dept,"department")
class Finance(Employee):
        pass
m=Manager("Amritha",2000,"HR")
m.display()
f=Finance("Kevin",3000)
f.display()
```

Output:

```
Amritha is in HR department
Kevin has a salary of 3000
```

Method overloading

```python
class Demo:
    def m1(self):
        print("no arguments")
    def m1(self,a):
        print("1 argument")
    def m1(self,a,b):
        print("2 arguments")
d=Demo()
d.m1()
d.m1(10)
d.m1(10,10)
```

Here we will not get output. New method will overwrite the old method. So this raise an error.
Method overloading can be done in a different way

Method Overloading

```python
class Student:
    def average(self,a=None,b=None,c=None):
        if a!=None and b!=None and c!=None:
            s=a+b+c
            return s
        elif a!=None and b!=None:
            s=a+b
            return s
        else:
            return a
s=Student()
print(s.average(10,10,10))
print(s.average(10,10))
print(s.average(10))
```

```python
class A:
    def show(self):
        print("A in show")

class B(A):
   # pass
   def show(self):
       print("B show")
b1=B()
```

```python
b1.show()
# If  B doesnt have a method show()and since class B inherits from A and  if there is method Show()iwn A (base class) then B inherits from A.But when B defines a method called show,then its object will call its own method.
```

Operator Overloading

```python
class Complex:
    def __init__(self,r,i):
        self.real=r
        self.imag=i
    def __str__(self):
        return f" {self.real} + {self.imag}i"
    def __add__(self,other):
        return Complex( self.real+other.real,self.imag+other.imag)
    def __mul__(self,other):
        return Complex(self.real*other.real,self.imag*other.imag)
c1=Complex(3,4)
c2=Complex(4,7)
print("First Number",c1)
print("Second Number",c2)
result=c1+c2
result1=c1*c2
print("Result of Addition is ",result)
print("Result of Multiplication is ",result1)

#print(c1+c2)
```

Output:

```
First Number 3 + 4i
Second Number 4 + 7i
Result of Addition is 7 + 11i
Result of Multiplication is 12 + 28i
```

Using comparison operator overloading, find who will get promotion given the employee name,performance score,yr of experience

```python
class Person:
    def __init__(self,name,score,yrofexp):
        self.name=name
        self.score=score
        self.exp=yrofexp
    def __gt__(self,other):
        if self.score>other.score:
            return self.name
        elif self.score==other.score:
            if self.exp>other.exp:
                return self.name
        else:
            return other.name


p1=Person("Ann",124,11)
p2=Person("Ema",130,10)
result=p1>p2
if result:
    print(p1.name ,"is eligible for promotion")
else:
    print(p2.name, "is eligible for promotion")
```

Output:
Ann is eligible for promotion

```python
# Base class
class Student:
    def __init__(self, studentName):
        self.studentName = studentName

    def displayDetails(self):
        print(f"Student Name: {self.studentName}")

# Derived class (Level 1)
class Graduate(Student):
    def __init__(self, studentName, graduationYear):
        super().__init__(studentName)
        self.graduationYear = graduationYear

    def displayDetails(self):
        print(f"Student Name: {self.studentName}")
        print(f"Graduation Year: {self.graduationYear}")

# Derived class (Level 2)
class Postgraduate(Graduate):
    def __init__(self, studentName, graduationYear, thesisTopic):
        super().__init__(studentName, graduationYear)
        self.thesisTopic = thesisTopic

    def displayDetails(self):
        print(f"Student Name: {self.studentName}")
        print(f"Graduation Year: {self.graduationYear}")
        print(f"Thesis Topic: {self.thesisTopic}")
student = Student("Alice")
student.displayDetails()

print("\nGraduate:")
graduate = Graduate("Bob", 2025)
graduate.displayDetails()

print("\nPostgraduate:")
postgrad = Postgraduate("Charlie", 2024, "Machine Learning in Healthcare")
postgrad.displayDetails()
```

Output:

```
Student Name: Alice
Graduate:
Student Name: Bob
Graduation Year: 2025
Postgraduate:
Student Name: Charlie
Graduation Year: 2024
Thesis Topic: Machine Learning in Healthcare
```