Chap.- 9

Polymorphism

- ## using same to perform different methods

In [14]:

```
class A:
    def display(self, x):
        print("Display without parameter")

    def display(self, x, y):
        print("Display x :",x,"y :",y)

ob = A()
# ob.display()
#
ob.display(1,2) # Display x : 1 y : 2
ob.display(5, 1) # Display x : 5 y : 1
```

```
Display x : 1 y : 2
Display x : 5 y : 1
```

In [11]:

```
class A:
    def display(self, x):
        print("Display without parameter")

    def display(self, x, y):
        print("Display x :",x,"y :",y)

ob = A()
# ob.display()
#
ob.display(1,2) # Display x : 1 y : 2
ob.display(5) # TypeError: display() missing 1 required positional argu
```

```
Display x : 1 y : 2

---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-11-8bbd0f12908e> in <module>
     10 #
     11 ob.display(1,2) # Display x : 1 y : 2
---> 12 ob.display(5) # Display x : 5 y : None

TypeError: display() missing 1 required positional argument: 'y'
```

In [15]:

```python
class A:
    def display(self):
        print("Display without parameter")

    def display(self, x, y=None):
        print("Display x :",x,"y :",y)

ob = A()
ob.display(1,2) # Display x : 1 y : 2
ob.display(5) # Display x : 5 y : None
```

```
Display x : 1 y : 2
Display x : 5 y : None
```

In [24]:

```python
class A:
    def display(self, x, y=None):
        if(y == None):
            print(x)
        else:
            print(x, y)
ob = A()
# ob.display()
#
ob.display(1,2) # 1 2
ob.display(5) # 5
```

```
1 2
5
```

- **Can't support overloading...**
- **In python we create illusion of using overloading.**

In [19]:

```python
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y

p1 = A(2,3)
p2 = A(10,20)
print(p1 + p2)
# TypeError: unsupported operand type(s) for +: 'A' and 'A'
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-19-e58a3b6646e9> in <module>
      6 p1 = A(2,3)
      7 p2 = A(10,20)
----> 8 print(p1 + p2)
      9 # TypeError: unsupported operand type(s) for +: 'A' and 'A'

TypeError: unsupported operand type(s) for +: 'A' and 'A'
```

• Operator Overloading

In [22]:

```python
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return(x,y)

p1 = A(2,3)
p2 = A(10,20)
print(p1 + p2) # (12, 23)
```

```
(12, 23)
```

In [26]:

```python
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return(x,y)

p1 = A(2,3)
p2 = A(10,20)
print(p1 * p2) # TypeError: unsupported operand type(s) for *: 'A' and

# fun name je hoy te sign use karvi
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
<ipython-input-26-6c8a3007c8d0> in <module>
     11 p1 = A(2,3)
     12 p2 = A(10,20)
---> 13 print(p1 * p2) # TypeError: unsupported operand type(s) for *: 'A'
and 'A'

TypeError: unsupported operand type(s) for *: 'A' and 'A'
```

In [28]:

```python
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x * other.x
        y = self.y * other.y
        return(x,y)

p1 = A(2,3)
p2 = A(10,20)
print(p1 + p2) # (20, 60)
```

```
(20, 60)
```

- **add**
- **sub**
- **mul**
- **truediv (/)**
- **floordiv (//)**
- **mod (%)**
- **pow (**)**
- **lt (<)**
- **gt (>)**
- **le (<=)**
- **ge (>=)**
- **ne (!=)**
- **eq (==)**

In [35]:

```python
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return(x,y)
    def __sub__(self, other):
        x = self.x - other.x
        y = self.y - other.y
        return(x,y)
    def __mul__(self, other):
        x = self.x * other.x
        y = self.y * other.y
        return(x,y)
    def __truediv__(self, other):
        x = self.x / other.x
        y = self.y / other.y
        return(x,y)
    def __floordiv__(self, other):
        x = self.x // other.x
        y = self.y // other.y
        return(x,y)
    def __mod__(self, other):
        x = self.x % other.x
        y = self.y % other.y
        return(x,y)
    def __pow__(self, other):
        x = self.x ** other.x
        y = self.y ** other.y
        return(x,y)
    def __lt__(self, other):
        x = self.x < other.x
        y = self.y < other.y
        return(x,y)
    def __gt__(self, other):
        x = self.x > other.x
        y = self.y > other.y
        return(x,y)
    def __le__(self, other):
        x = self.x <= other.x
        y = self.y <= other.y
        return(x,y)
    def __ge__(self, other):
        x = self.x >= other.x
        y = self.y >= other.y
        return(x,y)
    def __ne__(self, other):
        x = self.x != other.x
        y = self.y != other.y
        return(x,y)
    def __eq__(self, other):
        x = self.x == other.x
        y = self.y == other.y
        return(x,y)


p1 = A(2,3)
p2 = A(10,20)
```

```python
print(p1 + p2) # (12, 23)
print(p1 - p2) # (-8, -17)
print(p1 * p2) # (20, 60)
print(p1 / p2) # (0.2, 0.15)
print(p1 // p2) # (0, 0)
print(p1 % p2) # (2, 3)
print(p1 ** p2) # (1024, 3486784401)
print(p1 < p2) # (True, True)
print(p1 > p2) # (False, False)
print(p1 <= p2) # (True, True)
print(p1 >= p2) # (False, False)
print(p1 != p2) # (True, True)
print(p1 == p2) # (False, False)
```

```
(12, 23)
(-8, -17)
(20, 60)
(0.2, 0.15)
(0, 0)
(2, 3)
(1024, 3486784401)
(True, True)
(False, False)
(True, True)
(False, False)
(True, True)
(False, False)
```

P.b. = 680

In [45]:

```python
class St:
    def __init__(self, name, rn, age, marks):
        self.n = name
        self.r = rn
        self.a = age
        self.m = marks
    def display(self):
        print("Name :",self.n)
        print("Roll No. :",self.r)
        print("Age :",self.a)
        print("Marks :",self.m)

    def __eq__(self, other):
        if(self.m == other.m):
            print("Both marks are Same :)")
            return True
        else:
            print("Both marks are not Same")
            return False

s1 = St('Romil', 84, 18, 24)
s2 = St('Yash' , 94, 18, 25)
s3 = St('Rudra' , 90, 18, 24)

print(s1 == s2) # Both marks are not Same
print(s1 == s3) # Both marks are Same :)
```

```
Both marks are not Same
False
Both marks are Same :)
True
```

- ## Inheritance :

In [51]:

```python
class A:
    def demo(self):
        print("Class A")
class B:
    def dis(self):
        print("Class B")

ob = B()
ob.demo()
ob.dis() # AttributeError: 'B' object has no attribute 'demo'
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-51-2092b482f89d> in <module>
      7
      8 ob = B()
----> 9 ob.demo()
     10 ob.dis() # AttributeError: 'B' object has no attribute 'demo'

AttributeError: 'B' object has no attribute 'demo'
```

- ## Single (Simple) Inheritance

In [54]:

```python
class A:
    def demo(self):
        print("Class A")
class B(A):
    def dis(self):
        print("Class B")

ob = B()
ob.demo() # Class A
ob.dis() # Class B
```

```
Class A
Class B
```

In [56]:

```python
class A:
    def demo(self):
        print("Class A")
class B(A):
    def dis(self):
        print("Class B")

ob = A()
ob.demo() # Class A
ob.dis() # Class B
```

```
Class A

---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-56-0d7de25aaab0> in <module>
      8 ob = A()
      9 ob.demo() # Class A
---> 10 ob.dis() # Class B

AttributeError: 'A' object has no attribute 'dis'
```

P.b. 671

In [63]:

```python
class Book():
    def __init__(self):
        self.name = input("Enter Name of Book : ")
        self.no = input("Enter No. of Book : ")
        self.a = input("Enter Name of Author : ")
        self.pub = input("Enter Name of Publiser : ")
        self.isbn = input("Enter ISBN : ")
        self.y = input("Enter Year : ")

    def display(self):
        print("Name :", self.name)
        print("No. :", self.no)
        print("Name of Author :", self.a)
        print("Publiser :", self.pub)
        print("ISBN :", self.isbn)
        print("Year :", self.y)

class TextBook(Book):
    def __init__(self):
        super().__init__()
        self.co = input("Enter Course : ")

    def display(self):
        super().display()
        print("Course : ", self.co)

# --------------------------------------------------------
B1 = TextBook()
B1.display()
```

```
Enter Name of Book : sdsd
Enter No. of Book : 234
Enter Name of Author : dfh
Enter Name of Publiser : jhg
Enter ISBN : 5214
Enter Year : 2045
Enter Course : adfgad
Name : sdsd
No. : 234
Name of Author : dfh
Publiser : jhg
ISBN : 5214
Year : 2045
Course :  adfgad
```

- ## Types of Inheritance

1.) Single P-C

2.) Multiple 2P-C

3.) Multilevel

4.) Heirachical

5.) Hybrid

- # Multiple Inheritance

In [69]:

```
class Person():
    def __init__(self):
        self.name = input("Enter Name : ")
        self.age = input("Enter Age : ")

class Car():
    def __init__(self):
        self.model = input("Enter Model : ")
        self.Color = input("Enter Color : ")

class Parking(Person, Car):
    def __init__(self):
        Person.__init__(self)
        Car.__init__(self)
        self.pn = input("Enter Parking No. : ")
    def display(self):
        print("------------Person Details----------------")
        print("Person Name :", self.name)
        print("Person Age :", self.age)
        print("------------Car Details----------------")
        print("Car Model :", self.model)
        print("Car Color :", self.Color)
        print("------------Parking Details----------------")
        print("Parking No. :", self.pn)

ob = Parking()
ob.display()

# Enter Name : Romil
# Enter Age : 18
# Enter Model : Mustang 1969
# Enter Color : Black
# Enter Parking No. : 8
# ------------Person Details----------------
# Person Name : Romil
# Person Age : 18
# ------------Car Details----------------
# Car Model : Mustang 1969
# Car Color : Black
# ------------Parking Details----------------
# Parking No. : 8
```

```
Enter Name : Romil
Enter Age : 18
Enter Model : Mustang 1969
Enter Color : Black
Enter Parking No. : 8
------------Person Details----------------
Person Name : Romil
Person Age : 18
------------Car Details----------------
Car Model : Mustang 1969
Car Color : Black
------------Parking Details----------------
Parking No. : 8
```

- ## Method Resolution Order (Left to Right)

- ### DFS Search (Depth First Search)

Mcq. 636

In [5]:

```
class A:
    def rk(self):
        print(" In class A")
class B:
    def rk(self):
         print(" In class B")
class C(A, B):
    def rk(self):
         pass
r = C()
print(C.__mro__)
# (<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <c

print(B.__mro__) # (<class '__main__.B'>, <class 'object'>)

```

```
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class
'object'>)
(<class '__main__.B'>, <class 'object'>)
```

Mcq. 648

In [8]:

```
class P:
    pass
class Q:
    pass
class R(P,Q):
    pass
class S(Q):
    pass
class T(S,R):
    pass
a=T()
T.__mro__
# (__main__.T, __main__.S, __main__.R, __main__.P, __main__.Q, object)
```

Out[8]:

```
(__main__.T, __main__.S, __main__.R, __main__.P, __main__.Q, object)
```

Mcq. 640

In [15]:

```python
class A:
    pass
class B:
    pass
class C:
    pass
class X(A,B):
    pass
class Y(C,A,B):
    pass
class Z(A):
    pass
class P(Z,Y,X):
    pass

P.__mro__

# (__main__.P,
#  __main__.Z,
#  __main__.Y,
#  __main__.C,
#  __main__.X,
#  __main__.A,
#  __main__.B,
#  object)
```

Out[15]:

```
(__main__.P,
 __main__.Z,
 __main__.Y,
 __main__.C,
 __main__.X,
 __main__.A,
 __main__.B,
 object)
```

- **Also Check left to right in Sub-Parent (for all tree)**

In [20]:

```
class A:
    pass
class B:
    pass
class C:
    pass
class X(A,B):
    pass
class Y(A,B,C):
    pass
class Z(A):
    pass
class P(Z,Y,X):
    pass

P.__mro__

# (__main__.P,
#  __main__.Z,
#  __main__.Y,
#  __main__.X,
#  __main__.A,
#  __main__.B,
#  __main__.C,
#  object)
```

Out[20]:

```
(__main__.P,
 __main__.Z,
 __main__.Y,
 __main__.X,
 __main__.A,
 __main__.B,
 __main__.C,
 object)
```

In [21]:

```
class A:
    pass
class B:
    pass
class C:
    pass
class X(B,A):
    pass
class Y(A,B,C):
    pass
class Z(A):
    pass
class P(Z,Y,X):
    pass

P.__mro__


# TypeError: Cannot create a consistent method resolution
# order (MRO) for bases A, B
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-21-40d2f551462d> in <module>
     11 class Z(A):
     12     pass
---> 13 class P(Z,Y,X):
     14     pass
     15

TypeError: Cannot create a consistent method resolution
order (MRO) for bases A, B
```

Mcq. 695

In [24]:

```python
class A: pass
class B: pass
class C: pass
class D:pass
class E:pass
class K1(C,A,B): pass
class K3(A,D): pass
class K2(B,D,E): pass
class Z( K1,K3,K2): pass

Z.__mro__

# (__main__.Z,
#  __main__.K1,
#  __main__.C,
#  __main__.K3,
#  __main__.A,
#  __main__.K2,
#  __main__.B,
#  __main__.D,
#  __main__.E,
#  object)
```

Out[24]:

```
(__main__.Z,
 __main__.K1,
 __main__.C,
 __main__.K3,
 __main__.A,
 __main__.K2,
 __main__.B,
 __main__.D,
 __main__.E,
 object)
```

P.b. 691

In [25]:

```python
# Write a python program to create a Bus child class that inherits from
# In Vehicle class vehicle name, mileage and seatingcapacity as its dat
# seating capacity * 100. If Vehicle is Bus instance, we need to add an
# total fare for bus instance will become the final amount = total fare
# Sample Output:
# The bus seating capacity is 50. so, the final fare amount should be 5
# The car seating capacity is 5. so, the final fare amount should be 50
```

In [30]:

```python
class Vehicle():
    def __init__(self):
        self.vn = input("Enter Vehicle Name : ")
        self.m = int(input("Enter Mileage : "))
        self.sc = int(input("Enter Seating Capacity : "))

    def fare(self):
        return self.sc*100

class Bus(Vehicle):
    def __init__(self):
        super().__init__()
    def display(self):
        print(self.name, self.mileage, self.sc)
    def fare(self):
        return super().fare()+super().fare()*0.1
class car(Vehicle):
    def __init__(self):
        super().__init__()
    def fare(self):
        return super().fare()
b=Bus()
print(b.fare())
c=car()
print(c.fare())
```

```
Enter Vehicle Name : abc
Enter Mileage : 50
Enter Seating Capacity : 50
5500.0
Enter Vehicle Name : sdf
Enter Mileage : 50
Enter Seating Capacity : 100
10000
```

P.b 694

In [5]:

```
class Matrix():
    def __init__(self):
        self.r = 3
        self.c = 3

    def getr(self):
        print(self.r)

    def getc(self):
        print(self.c)

    def set_m(self):
        self.mat = []
        for i in range(self.r):
            l=[]
            for j in range(self.c):
                n = int(input("Enter element : "))
                l.append(n)
            self.mat.append(l)
        print(self.mat)

m = Matrix()
m.getr()
m.getc()
m.set_m()
```

```
3
3
Enter element : 1
Enter element : 2
Enter element : 3
Enter element : 4
Enter element : 5
Enter element : 6
Enter element : 7
Enter element : 8
Enter element : 9
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Abstraction

- ### when inheriting class then class with restriction

In [ ]:

```
# from abc import ABC             \
#                                  | } abstract class
# abc - abstraction base class    /
```

In [6]:

```python
from abc import ABC
class Abdemo(ABC):
    def demo(self):
        pass

class Base (Abdemo):
    def display(self):
        print("Base")

ob = Base()
ob.display()
```

```
Base
```

- ## use Decorator to create abstract method

In [12]:

```python
from abc import ABC,abstractmethod
class AbDemo(ABC):
    @abstractmethod
    def demo(self):
        pass

class Base (AbDemo):
    def display(self):
        print("Base")

ob = Base()
ob.display()

# TypeError: Can't instantiate abstract class Base with abstract method
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-12-f15ec5789ca0> in <module>
      9         print("Base")
     10 
---> 11 ob = Base()
     12 ob.display()

TypeError: Can't instantiate abstract class Base with abstract methods dem
o
```

In [9]:

```python
from abc import ABC, abstractmethod
class AbDemo(ABC):
    @abstractmethod
    def demo(self):
        pass

class Base(Abdemo):
    def demo(self):
        print("Base")

ob = Base()
ob.demo()

# Base
```

```
Base
```

P.b.- 692 IMP

In [17]:

```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def cal_area(self):
        pass

class Rect(Shape):
    def cal_area(self, l, b):
        self.Rarea = l * b
        return self.Rarea

    def __gt__(self,other):
        return self.Rarea > other.Rarea

class Circle(Shape):
    def cal_area(self, r):
        self.Carea = 3.14 * r**2
        return self.Carea

r1 = Rect()
print(r1.cal_area(10,20))
r2 = Rect()
print(r2.cal_area(20,30))

c1 = Circle()
print(c1.cal_area(10))

print(r1 > r2)
print(r2 > r1)

print(Circle.__mro__)

# 200
# 600
# 314.0
# False
# True
# (<class '__main__.Circle'>, <class '__main__.Shape'>, <class 'abc.ABC
```

```
200
600
314.0
False
True
(<class '__main__.Circle'>, <class '__main__.Shape'>, <class 'abc.ABC'>, <
class 'object'>)
```

P.b.- 690 IMP

In [28]:

```python
from abc import ABC, abstractmethod
class Employee(ABC):
    @abstractmethod
    def receive_call(self):
        pass

    @abstractmethod
    def end_call(self):
        pass
    @abstractmethod
    def is_free(self):
        pass

    @abstractmethod
    def get_rank(self):
        pass

class Respondent(Employee):
    def __init__(self):
        self.id = 101
        self.name = 'abc'
        self.rank = 3
        self.free = True

    def receive_call(self):
        print("call received by", self.name)
        self.free = False

    def end_call(self):
        print("call ended")
        self.free = True

    def is_free(self):
        return self.free

    def get_rank(self):
        return self.rank

class Manager(Employee):
    def __init__(self):
        self.id = 103
        self.name = 'lmn'
        self.rank = 2
        self.free = True

    def receive_call(self):
        print("call received by", self.name)
        self.free = False

    def end_call(self):
        print("call ended")
        self.free = True

    def is_free(self):
        return self.free

    def get_rank(self):
        return self.rank

class Director(Employee):
    def __init__(self):
        self.id = 104
        self.name = 'xyz'
        self.rank = 1
        self.free = True

    def receive_call(self):
        print("call received by", self.name)
        self.free = False

    def end_call(self):
        print("call ended")
        self.free = True

    def is_free(self):
        return self.free

    def get_rank(self):
        return self.rank

class Call():
    def __init__(self):
        self.id = 102
        self.name = "indipendent"
        self.assigned = False

class CallHandler():
    respondents = []
    managers = []
    directors = []

    def add_employee(self, ob):
        if ob.rank == 3:
            CallHandler.respondents.append(ob)
        elif ob.rank == 2:
            CallHandler.managers.append(ob)
        elif ob.rank == 1:
            CallHandler.directors.append(ob)

    def dispatch_call(self, call):
        for employee in CallHandler.respondents:
            if employee.is_free():
                employee.receive_call()
                call.assigned = True
                print(f"Call assigned to {employee.name} (Respondent)")
                return

        for employee in CallHandler.managers:
            if employee.is_free():
                employee.receive_call()
                call.assigned = True
                print(f"Call assigned to {employee.name} (Manager)")
                return

        for employee in CallHandler.directors:
            if employee.is_free():
                employee.receive_call()
                call.assigned = True
                print(f"Call assigned to {employee.name} (Director)")
                return

        print("Sorry! All employees are currently busy.")
```

```python


r1 = Respondent()
r2 = Respondent()
r3 = Respondent()
m1 = Manager()
d1 = Director()

ch = CallHandler()
ch.add_employee(r1)
ch.add_employee(r2)
ch.add_employee(r3)

ch.add_employee(m1)
ch.add_employee(d1)

call = Call()

ch.dispatch_call(call)
ch.dispatch_call(call)
ch.dispatch_call(call)
ch.dispatch_call(call)


# Output:

# call received by abc
# Call assigned to abc (Respondent)
# call received by abc
# Call assigned to abc (Respondent)
# call received by abc
# Call assigned to abc (Respondent)
# call received by lmn
# Call assigned to lmn (Manager)

# call received by abc
# Call assigned to abc (Respondent)

# If Respondent is Busy then :
# call received by lmn
# Call assigned to lmn (Manager)
```

```
call received by abc
Call assigned to abc (Respondent)
call received by abc
Call assigned to abc (Respondent)
call received by abc
Call assigned to abc (Respondent)
call received by lmn
Call assigned to lmn (Manager)
```

P.b.- 696

In [*]:

```
from abc import ABC, abstractmethod

class Employee(ABC):
    @abstractmethod
    def data(self):
        self.id = 101
        self.name = 'abc'
        self.salary = 25000

    def display(self):
        print(f"ID : {self.id} Name : {self.name} Salary : {self.salary

    def get_sal(self):
        return self.salary

    @abstractmethod
    def emp_id(self):
        pass

class Perks(Employee):
    def cal_perk(self):
        self.da = self.salary * 0.35
        self.hra = self.salary * 0.17
        self.pf = self.salary * 0.12
        self.total = self.da + self.hra - self.pf
        print("DA : ",self.da)
        print("HRA : ",self.hra)
        print("PF : ",self.pf)
        print("Total : ", self.total)

class NetSalary(Perks):
    def cal_nets(self):
        self.final = self.salary + self.total
        print("Net Salary : ", self.final)

n = NetSalary()

n.cal_perk()
n.cal_nets()
```

In [ ]:

```

```