Polymorphism

- Poly means many
- Morphs is nothing but forms

- Polymorphism plays an important role in allowing multiple object to have difference internal structure by sharing same external interface
- In Simple words ability to perform more than one form for an example consider an operator "+" in python is polymorphic behavior i.e if we pass two integers as an operands then it will perform an addition and produce sum [10+20 = 30]
- similarly if we pass two strings as an operands it will perform concatenation and produce concatenated output ["sai"+"baba" => "saibaba"]

- In other languages Polymorphism classified into 2 types

- **Compile time Polymorphism**
- **Run-time Polymorphism**

- In Python Only Polymorphism. As per the python if anything exhibit more than one behavior then they are fall into polymorphism

- In Python polymorphism can do through
  - Method overloading
  - Constructor overloading
  - Method overriding
  - Constructor overriding
  - Operator overloading

➢ Method Overloding :

- In other languages method overloading is the process of defining more than one method with the same for difference purpose
- In Python if we define more than one method with same name priority always given to last defined method
- In Python "Methodoverloading" is nothing but making a method to exhibit more than one behavior.

Example:

```
class Sample:
   def method1(self):
      print("Mtd-1 with out args ")

   def method1(self,x):
      print("Mtd-1 with 1-arg : ",x)

   def method1(self,x,y):
      print("Mtd-1 with  2-args : ",x,y)

#calling
s=Sample()
s.method1(10,20)
#s.method1(90)
#s.method1()
```

Example 2:

```
class Sample:
   def sum(self,x=None,y=None,z=None):
      if x!=None and y!=None and z!=None:
         return x+y+z
```

```
        elif x!=None and y!=None:
           return x+y
        elif x!=None:
           return x
        else:
           return 0

    #calling
    s=Sample()
    r=s.sum(10,20,30)
    print("sum is : ",r)
    r=s.sum(10,20)
    print("sum is : ",r)
    r=s.sum(10)
    print("sum is : ",r)
    r=s.sum()
    print("sum is : ",r)
```

Example 3:

```
import time
class Sample:
   def method1(self,*n):  # *n is varargs parameter
      time.sleep(1)
      print("Data is : ",n)   # here n is tuple | iterable
      s=sum(n)   #  sum(iterable)  predefined  function  from  built-ins
module
      print("Sum is : ",s)
      print("- "*30)

#Calling
s=Sample()
s.method1()
```

```
s.method1(10,20)
s.method1(10,20,30)
s.method1(10,20,30,40,50,60,70,80,90)
```

Constructor Overloading :

- In Java constructor overloading is the process of defining more than one constructor for difference purpose. But if python if we define more than one constructor in the class then priority is given to the last defined constructor only
- Here constructor overloading is nothing but making the constructor to exhibit more than one behavior

Example:

```
import  time

class Sample:
   def __init__(self):
      print("def const of Sample ")

   def __init__(self,x):
      print("1-para const of Sample : ",x)

#calling
s1=Sample(120)
s2=Sample()
```

Example 2:

```
import  time

class Sample:
   def __init__(self,x=None,y=None,z=None):
      if x!=None and y!=None and z!=None:
         print("3 parameter constructor : ",x,y,z,sep=' ')
      elif x!=None and y!=None:
         print("2 parameter constructor : ",x,y,sep=' ')
      elif x!=None:
         print("1 para const : ",x)
      else:
         print("0 para const ")

#calling
s1=Sample(10,20,30)
s2=Sample(10,20)
s3=Sample(10)
s4=Sample()
```

Example : 3:

```
import time

class Student:
   def __init__(self,**kwargs):
      time.sleep(1)
      print("- "*30)
      for k,d in kwargs.items():
         print(k,"<<>>",d)
```

```
#calling
s1=Student(sno=101)
s2=Student(sno=102,sname='Ramesh')
s3=Student(sno=103,sname="sudha",scity="Hyd")
```

Method Overriding :

- It is the process of defining the sub class method same as super class method.
- Whenever you want use the super class method into the sub class with difference implementation then we need to go for method overriding.
- If we define sub class method same as super class method then priority is given the sub class method. If you want call the super members [field | methods | constructor] then we have to make use of "super()"
    - **super().methodname([list of args])**

Example:

```
class Super:
   def method1(self):
      print("Mtd-1 of Super ")

class Sub(Super):
   def method1(self):
      print("Mtd-1 of Sub")

#calling
s=Sub()
s.method1()
```

Example 2:

```
class Super:
   def method1(self):
      print("Mtd-1 of Super ")

class Sub(Super):
   def method1(self):
      super().method1()
      print("Mtd-1 of Sub")

#calling
s=Sub()
s.method1()
```

Constructor Overriding:

- If the sub class is not defined with any constructor then PVM will invoke the super constructor implicitly while creating an Object of the “subclass”.
- If the sub class is defined constructor then priority is given to the sub class constructor only
- If you want call the super constructor from the sub class constructor then we have to use “super()”
  - super().__init__([arguments])

Example:

```
class Super:
   def __init__(self):
      print("def const of super ")
```

```
class Sub(Super):
   def __init__(self):
      super().__init__()        #calling    super    class    "default"
constructor
      print("def const of sub")

#calling
s=Sub()
```

Example 2:

```
class Person:
   def __init__(self,sno,sname):
      self.sno=sno
      self.sname=sname

   def getData(self):
      print("Sno is : ",self.sno)
      print("Sname is : ",self.sname)

class Student(Person):
   def __init__(self,sno,sname,scourse):
      super().__init__(sno,sname)
      self.scourse=scourse

   def getData(self):
      super().getData()
      print("scourse is : ",self.scourse)

#calling
no=input("Enter student number ")
name=input("Enter sname ")
course=input("Enter course ")
```

```
s=Student(no,name,course)
s.getData()
```

Operator Overloading :

Operator overloading is the process of providing an additional functionalities to the operator without making it preexisted nature of the operator

In order to achieve operator overloading we override the predefined methods provided by python in the "Object" class

For every operator there is a predefined methods is provided class object i.e "MAGIC methods"

Magic Methods

```
+  :  __add__(self,other)
-  : __sub__(self,other)
*  : __mul__(self,other)
/  : __div__(self,other)
% : __mod__(self,other)
// : __floordiv__(self,other)


+= :  __iadd__(self,other)
-=  : __isub__(self,other)
*=  : __imul__(self,other)
/=  : __idiv__(self,other)
```

%= : __imod__(self,other)
//= : __ifloordiv__(self,other)

> : __gt__(self,other)
>= : __ge__(self,other)
< : __lt__(self,other)
<= : __le__(self,other)
== : __eq__(self,other)
!= : __ne__(self,other)

Example:

```
class Book_Java:
   def __init__(self):
      self.pages=300

   def __add__(self,other):
      return self.pages+other.pages

class Book_Python:
   def __init__(self):
      self.pages=400

#Calling
bj=Book_Java()
bp=Book_Python()
np=bj+bp      # bj.__add__(bp)
print("No.of.Pages are : ",np)

'''
```

```
x=10  #<class 'int'>
y=20  #<class 'int'>
s=x+y  # int + int
print("Sum is : ",s)

m="Sai" #str
n="Baba" #str
o=m+n   #str + str -> str
print("Result is : ",o)  # SaiBaba  '''
```

Example :

```
class Employee:
   def __init__(self):
      self.name=input("enter name : ")
      self.salpd=int(input("enter sal per day  :"))

   def __mul__(self,other):
      return self.salpd*other.dw

class Attend:
   def __init__(self):
      self.name=input("enter name : ")
      self.dw=int(input("Enter days worked "))

   def __mul__(self,other):
      return self.dw*other.salpd

#calling
e=Employee()
a=Attend()
ns=a*e   # a.__mul__(e)
print("Net salary is : ",ns)
```

Example 3:

```
class Student:
   def __init__(self):
      self.sname=input("Enter sname :")
      self.tot=int(input("Enter total marks : "))

   def getStudent(self):
      print("Sname is : ",self.sname)
      print("Total marks is : ",self.tot)

   def __gt__(self,other):
      if self.tot > other.tot:
         return True
      else:
         return False


#calling
s1=Student()
s2=Student()

if s1>s2:   # __gt__(self,other)    s1.__gt__(s2)
   s1.getStudent()
else:
   s2.getStudent()
```