Polymorphism

Monday, July 21, 2025 7:55 PM

Poly → many

Morphism → forms

Eg:
2+3 → addition (+)
5

same operator
different operations

'hai'+'hello' → concatenation (+)
'haihello'

- Polymorphism is the phenomenon of making the operator or method to work on 2 or more different functionalities.

The 2 concepts are:

i. Method Overloading
ii. Operator Overloading

i. Method overloading: It is a phenomenon of making a method to work on 2 or more operations.

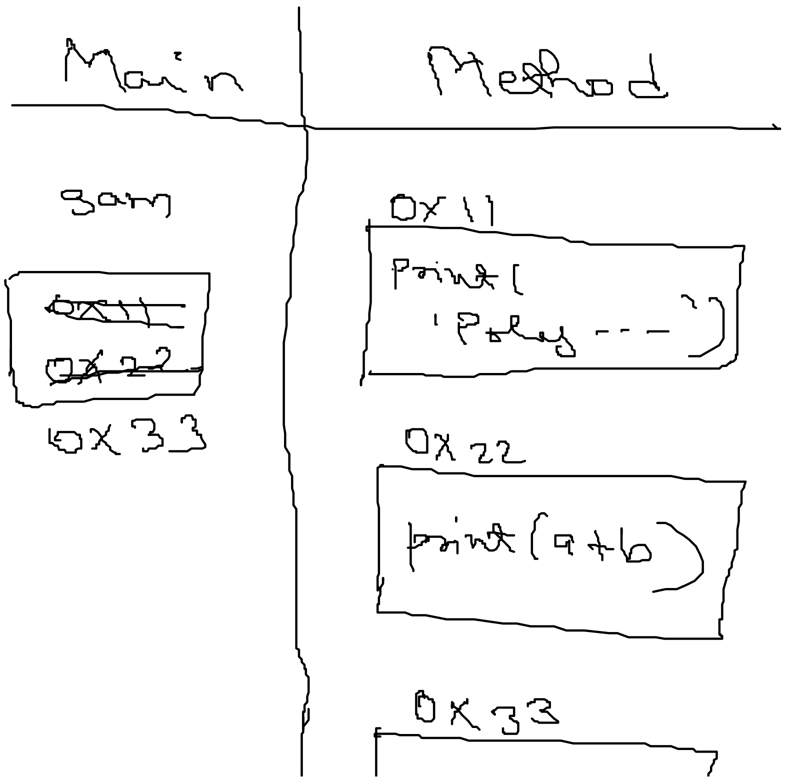
Eg:

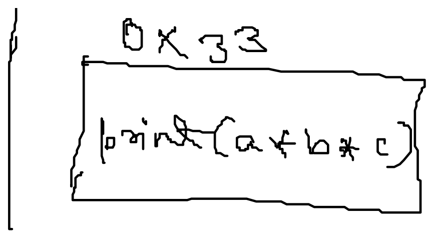
```
def sam():
   print('Polymorphism')

def sam(a,b):
   print(a+b)

def sam(a,b,c):
   print(a*b*c)

sam(1,2,3)
```





In python, Method overloading is not possible.
If tried, Method overriding happens.

Method overriding is the phenomenon of overriding the address of the previous method with the address of new method.

Note:

- In other languages method overloading happens based on the number of arguments passed.
- In python, overloading is not possible, here it takes the address of lastly created functions only and performs overriding.

If Still want to use the previous method, then is used through Monkey Patching.

Monkey Patching: Passing the address of a function to a variable, so that the variable can act as a function.

Eg:

```
def sam():
   print('Polymorphism')
mid1=sam

def sam(a,b):
   print(a+b)
mid=sam

def sam(a,b,c):
   print(a*b*c)

sam(1,2,3)
mid(2,5)
mid1()
```

If sam() is called, 3rd function will get executed.
If mid() is called, 2nd function will get executed.
If mid1() is called, 1st function is executed.

ii. Operator Overloading:

It is the phenomenon of making an operator to work on 2 or more operations.

Eg:

```
a=10
b=20
print(a+b)
print(type(a))
```

} → variables are the objects of Inbuilt-class.

Op:

```
30
<class 'int'>
```

```
class A:
   def __init__(self,a):
      self.a=a
```

[illegible] class

```
class A:
    def __init__(self,a):
        self.a=a
ob1=A(10)
ob2=A(20)     } → objects of user-defined class
print(ob1+ob2)
```

Op:
TypeError

Most operators supports objects of inbuilt type but none of the operators support user-defined type.

By using dir(A) we will get all the operations that can be performed on the class

Here in Operator overloading, we have to make the operators to work on user-defined classes using magic methods.

Eg:
```
class A:
    def __init__(self,a):
        self.a=a
    def __add__(self,other,new,odd):
        return self.a+other.a+new.a+odd.a
    def __sub__(self,other):
        return self.a-other.a
ob1=A(10)
ob2=A(20)
ob3=A(30)
ob4=A(40)
print(ob1.__add__(ob2,ob3,ob4))
print(ob1+ob2)  #ob1.__add__(ob2)
print(ob1-ob2)  #ob1.__sub__(ab2)
```

Operations	**Operator**	**Internal Operations**
Addition	Ob1+ob2	Ob1.__add__(ob2)
Substraction	Ob1-ob2	Ob1.__sub__(ob2)
Multiplication	Ob1*ob2	Ob1.__mul__(ob2)
True Division	Ob1/ob2	Ob1.__truediv__(ob2)
Floor Division	Ob1//ob2	Ob1.__floordiv__(ob2)
Modulus	Ob1%ob2	Ob1.__mod__(ob2)
power	Ob1**ob2	Ob1.__pw__(op2)
Bitwise And	Ob1&ob2	Ob1.__and__(ob2)
Bitwise OR	Ob1|ob2	Ob1.__or__(ob2)
Bitwise Not	~ob1	Ob1.__invert__()
Bitwise X-or	Ob1^ob2	Ob1.__xor__(ob2)
Bitwise Leftshift	Ob1<<ob2	Ob1.__lshift__(ob2)
Bitwise Rightshift	Ob1>>ob2	Ob1.__rshift__(ob2)
Equal to	Ob1==ob2	Ob1.__eq__(ob2)
Not Equal	Ob1!=ob2	Ob1.__ne__(ob2)
Greater Than	Ob1>ob2	Ob1.__gt__(ob2)
Less Than	Ob1<ob2	Ob1.__lt__(ob2)
Greater than or equal to	Ob1>=ob2	Ob1.__ge__(ob2)
Less than or equal to	Ob1<=ob2	Ob1.__le__(ob2)

Greater than or equal to	Ob1>=ob2	Ob1.__ge__(ob2)
Less than or equal to	Ob1<=ob2	Ob1.__le__(ob2)
Assignment Addition	Ob1+=ob2	Ob1.__iadd__(ob2)
Assignment Subtraction	Ob1-=ob2	Ob1.__isub__(ob2)

(for all the other assignment operators use the same forms.

Eg for collection data:

```
class Col:
    def __init__(self,a):
        self.a=a
    def __len__(self):  #find the length
        return len(self.a)
    def __getitem__(self,index): #to get the data based on values
        return self.a[index]
    def __setitem__(self,index,new): #to update the data
        self.a[index]=new
ob=Col([10,20,30,40,50])
print(len(ob))
print(ob[3])
ob[3]=60
ob[2]=100
print(ob[3])
print(ob[2])
```

Assignment:
Use operator overloading for the Bitwise and relational operators.