Polymorphism:

Polymorphism is a combination of two Greek words . Poly means many and morphos means forms so polymorphism means multiple forms.

Changing behaviour of elements (operator, method, object etc) according to given situations is refered as polymorphism in python.

Types of polymorphism :

- Method overloading
- Method overriding
- Operator overloading
- Duck typing

Duck Typing:

The name Duck Typing comes from the phrase:

“If it looks like a duck and quacks like a duck, it’s a duck”

Using Duck Typing, python does not check types at all while calling methods or using attributes. Instead, it check for the presence of a given method or attribute for that object. Here the type or the class of an object is less important than the method it defines.

```
class Duck:
    def run(self):
        print("Duck is running")
class Cat:
    def run(self):
        print("Cat is running")
class Dog:
    def walk(self):
        print("Dog is walking")
def runner(obj):
    obj.run()
d=Duck()
runner(d)
```

Strong Typing:

Presence or absence of a method or variable for an object can be checked using hasattr(object, “attribute”) method

It returns true or false

```
class Duck:
    def run(self):
        print("Duck is running")
class Cat:
    def run(self):
        print("Cat is running")
class Dog:
    def walk(self):
        print("Dog is walking")
for d in Duck(),Cat(),Dog():
    if hasattr(d,'run'):
        d.run()
    elif hasattr(d,’walk’):
        d.walk()
```

Method Overloading:

The concept of having different methods with same name but different argument-number and argument-types is known as method overloading.

Unlike other programming languages Python doesn't support this concept.

We may define many methods of the same name and different arguments, but we can only use the latest defined method. Calling the other method will produce an error.

```
class OvrldTest:
    def add(self,a,b):
        return a+b
    def add(self,a,b,c):
        return a+b+c
obj=OvrldTest()
print(obj.add(3,4,5))
print(obj.add(3,4)) #error !!!
```

Achieving Method Overloading indirectly:

We can use different ways to achieve the method overloading in python.

We can use the same methods to act differently with different types of arguments.
(Remember there are no multiple methods concept)

```
class OvrldTest:
    def add(self,a=None,b=None,c=None):
        if (a!=None and b!=None and c!=None):
            return a+b+c
        elif (a!=None and b!=None):
            return a+b
        else:
            return("Some values missing")
obj=OvrldTest()
print(obj.add(3,4,5))
print(obj.add(3,4))
print(obj.add(3))
```

Method Overriding:

The child class provides the specific implementation of the method that is already provided by the parent class.

It occurs between methods from the parent class (superclass) and the child class (child class).

It is used to change the behavior of existing methods

In simple words when a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to **override** the method in the super-class.

```
#over ridding (same method in both parent and child class)
class Test:
    def show(self):
        print("Parent class show")

class Demo(Test):
    def show(self):
        print("Child class show")
d = Demo()
d.show() #calls Demo class show method
```

Operator Overloading:

Changing the meaning of an operator in Python depending upon the operands used is known as operator overloading.

For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

We can use this concept to add two objects.

```
class OpOverload:
    def __init__(self,x):
        self.x=x
    def __add__(self,other):
        return self.x+other.x
class Other:
    def __init__(self,x):
        self.x=x
obj1=OpOverload(3)
obj2=Other(4)
print(obj1+obj2)
```