# Chapter Seven: Python and Polymorphism

## 7.1 Chapter objectives

By the end of this chapter, learners should be exhibit thorough understanding on how python implements the concept of inheritance

## 7.2 Introduction

Polymorphism comes the Greek words Poly (many) and morphism (forms). It means that the same function name can be used for different types.

Polymorphism refers to *having multiple forms*. Polymorphism is a programming term that refers to the use of the same method name, but with different signatures, for multiple types. It also allows morphism us to define methods in the child class with the same name as defined in their parent class.

## 7.3 Ways of Implementing polymorphism in python

There are four ways of implementing Polymorphism in Python i.e., *duck typing, method overloading, operator overriding, method overloading and operator overloading*

### 7.3.1 Method Overloading

By default, Python does not support method overloading, but we can achieve it by modifying out methods.

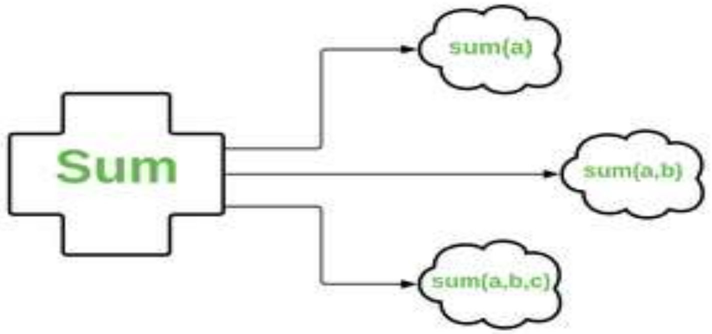

Fig 7.0-concept of method overloading

```
# A simple Python function to demonstrate method
#overloading in python

def add(a, b, c = 0):

   return a + b + c

 # Driver code

print(add(7,9))

print (add (7, 5, 1))
```

The above program will calculate the sum based on the number of arguments passed at runtime. The first print statement will print sum of two numbers while the second one will print sum of three numbers.

This process of calling the same method in different ways is called method overloading.

### 7.3.2 Method overriding

Method Overriding in Python is an OOPs concept closely related to inheritance. When a child class method provides its own implementation different from the parent class method of the same name, parameters and return type, we call this method overriding.

The child class's method is called the overriding method and the parent class's method is called the overridden method. Method overriding is completely different from the concept of method overloading. Method overloading occurs when there are two functions with the same name but different parameters.

**Advantages of Method Overriding in Python**

i Method Overriding allows us to change the implementation of a function in the child class which is defined in the parent class.

ii Method Overriding avoids duplication of code iii Method Overriding also enhances the code adding some additional properties.

Prerequisites for method overriding

i Method overriding cannot be done within a class. So, we need to derive a child class from a parent class. Hence Inheritance is mandatory.

ii The method must have the same name as in the parent class

iii The method must have the same number of parameters as in the parent class.

An example of a program that demonstrates method overriding is shown below. The method perimeter which has been defined in base class shapes and redefined (overridden) in square and rectangle classes

```
class shapes:
    def __init__(self):
        self.length=float(input("Enter the length\n"))
        self.width = float(input("Enter the width\n"))
    def perimeter(self):
        pass
class square(shapes):
    def __init__(self):
        pass
    def perimeter(self):
        self.side=float(input("Enter the square
side\n"))
        print("The perimeter of the square
is:\t",4*(self.side))
class rectangle(shapes):
    def __init__(self):
        shapes.__init__(self)
    def perimeter(self):
        print("The perimeter of the rectangle
is:\t",2*(self.length+self.width))
s=square()
s.perimeter()
p=rectangle()
p.perimeter()
```

### 7.3.3 Operator Overloading

In OOPs, operator overloading means we provide extended meaning beyond their predefined operational meaning.

For example, the operator add “+" is known for performing addition operations as well as adding two concatenating two strings or merging two lists.

However, we can overload the operator to give it additional behavior

### 7.3.4 Duck typing

Duck typing comes from the saying, “***If it walks like duck, swims like duck, looks like a duck, then it probably should be a duck***."

Duck typing is a concept related to dynamic typing, where the type or the class of an object is less important than the methods it defines

```python
# Python program to demonstrate
# duck typing

class Bird:
    def fly(self):
        print("fly with wings\n")

class Airplane:
    def fly(self):
        print("fly with Gasoline\n")

class Fish:
    def swim(self):
        print("fish swim in sea")

# Attributes having same name are
# considered as duck typing
for x in Bird(), Airplane(), Fish():
    x.fly()
```

## 7.4 Chapter summary

## 7.5 Further reading suggestions