

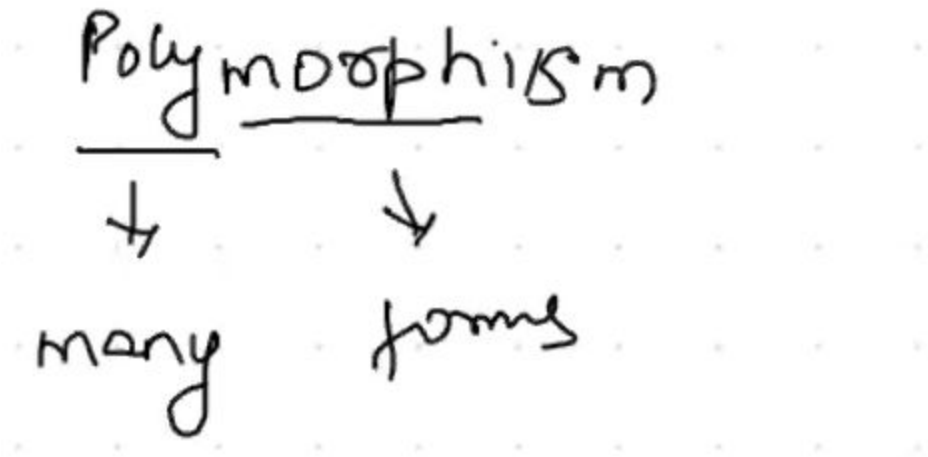
Date : 9th - 09 - 2020

Morning Session : 9am – 11.00 PM

By ~ Rohan Kumar

Topics: Python OOPs Day-3

What is Polymorphism : The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.



Types of polymorphism:

- 1) Method overloading
- 2) Method overriding

Duck typing: if bird walk like a duck , quack like a duck , swim like a duck, we can say that bird is duck

```
class Employee:
    def details(self):
        print("this is employee details")

class PrintDetails:
    def print(self,obj):
        obj.details()

e1 = Employee()
p1 = PrintDetails()
p1.print(e1)
```

this is employee details

```
class Employee:
    def details(self):
        print("this is employee details")

class GetDetails:
    def print(self,e1):
        e1.details()    #e1.details()

class Employer:
    def details(self):
        print("this is employer details")

e1 = Employee()    #created obj for emp
g1 = GetDetails()  #created obj for get details
emp = Employer()  #obj for employer class
g1.print(e1)
```

this is employee details

Method Overloading:

Method Overloading is the class having methods that are the same name with different arguments.

Arguments differently will be based on a number of arguments and types of arguments.

It is used in a single class.

It is also used to write the code clarity as well as reduce complexity.

```
class Calculator:    #method overloading
    def sum(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 a

cal = Calculator()
cal.sum(4)
```

Method Overriding:

same number of arguments, in different class,

It is implemented with inheritance also.

```
: class A:  #method overriding|
    def demo(self):
        print("i am in A demo")

class B(A):
    def demo(self):
        print(" i am in B demo")

b = B()
b.demo()
```

i am in B demo

Abstraction: Abstraction means hiding the complexity and only showing the essential features of the object. So in a way, Abstraction means hiding the real implementation and we, as a user, know only how to use it.

Real world example would be a vehicle which we drive without caring or knowing what all is going underneath.

A TV set where we enjoy programs without knowing the inner details of how TV works.

abstract class : is a class that contains one or more abstract methods. An Abstract method is a method that generally doesn't have any implementation, it is left to the sub classes to provide implementation for the abstract methods. Here note that in Python, abstract methods can have implementation in the abstract class too but the subclass inheriting that abstract class is still forced to override the abstract methods.

abstract class is created by deriving from the *meta class ABC* which belongs to the abc (Abstract Base Class) module.

```
from abc import ABC
Class MyClass(ABC):
```

Abstract Method: For defining abstract methods in an abstract class method has to be decorated with @abstractmethod decorator. From abc module @abstractmethod decorator has to be imported to use that annotation.

```
from abc import ABC, abstractmethod
Class MyClass(ABC):
    @abstractmethod
    def mymethod(self):
        #empty body
        pass
```

Abstract method: methods which does not have definition and it has only have declaration

Abstract Class : class with abstract methods, abstract class cant be instantiated(we can't create objects) and requires sub class to provide implementation for the abstract methods.

Requirements:

- 1) If we know the requirement and we don't know the design part then only we recreate abstract class.
- 2) We have the requirement what we should do we don't know how to implement it

```
: class X(ABC):
    @abstractmethod
    def m1(self):
        pass

    @abstractmethod
    def m2(self):
        pass

    def test(self):
        print("testing")

class Y(X):
    def m1(self):
        print("this is test")

    def m2(self):
        print("m2")

y = Y()
y.test()
```

Output:Testing

MCQ 1:

Polymorphism means

Attempted - 33 (122.22%)EASY^

☒ Same name methods78.79%

☐ same name class24.24%

MCQ 2:

Method Overloading is

Attempted - 36 (133.33%)EASY^

☐ Same method name , different class47.22%

☒ Same method name,same class52.78%

MCQ 3:

Method overriding is


Attempted - 37 (137.04%)EASY^

☐ Same name method, same class21.62%

☒ Same name method,different class81.08%


MCQ 4:

Abstraction means

 00:18

Attempted - 35 (129.63%)

EASY



<input type="checkbox"/>	Hiding data for protection	22.86%
<input checked="" type="checkbox"/>	Hiding unimportant data from user	77.14%