Module 4:

Polymorphism

By Ninad Gaikwad

Reference - “Core Python Programming”
Dr. R. Nageshwara Rao
Dreamtech Press

14.4 Polymorphism

- If a variable, object or method exhibits different behavior in different context it is called as polymorphism
- Examples of polymorphism is python are:
    - Duck typing philosophy in python
    - Operator overloading
    - Method overloading
    - Method overriding

14.4.1 Duck typing philosophy in python

- If it walks like a duck and talks like a duck, then it must be a duck.
- Eg.

```
# Duck class contains talk() method
class Duck:
        def talk(self):
                print('Quack, Quack')
# Human class contains talk() method
class Human:
        def talk(self):
                print('Hello Hi')
# this function accepts an object and calls its talk method
def call_talk(obj):
        obj.talk()
# call call_talk() and pass an object
# depending on the type of the object the talk() method is executed
x = Duck()
call_talk(x)
x = Human()
call_talk(x)

"""Output
Quack, Quack
Hello Hi  """
```

14.4.1 Duck typing philosophy in python

- Call to talk will give Attribute error if talk() method is not present in class
- To check if the method is presen in the class “hasattr(object,attribute)” function is used
- Eg: # Duck class contains talk() method

```
class Duck:
        def talk(self):
                print(‘Quack, Quack’)
# Human class contains talk() method
class Human:
        def speak(self):
                print(‘Hello Hi’)
# this function accepts an object and calls its talk method
def call_talk(obj):
        if hasattr(obj,’talk’):
                obj.talk()
        elif hasattr(obj,’speak’):
                obj.speak()
# call call_talk() and pass an object
# depending on the type of the object the talk() method is executed
x = Duck()
call_talk(x)
x = Human()
call_talk(x)
“””Output
Quack, Quack
Hello Hi  “””
```

14.4.2 Operator Overloading

- When an operator can perform different operations it is said to exhibit polymorphism
- Eg

```
# + operator operating on integers
print(10+5)

# operating on Strings
s1 = “Red”
s2 = “Fort”
print(s1+s2)

# Operating on lists
l1 = [1,2,3]
l2 = [4,5,6]
print(l1+l2)

“”” Output
15
RedFort
[1,2,3,4,5,6]
```

14.4.2 Operator Overloading

- a+b operator is internally written as a.__add__(b)
- We can make + operator act on objects by method overriding
- Eg.

```
class BookX:
        def __init__(self, pages):
            self.pages = pages
        def __add__(self, other):
            return self.pages+other.pages
class BookY:
        def __init__(self, pages):
            self.pages = pages
b1 =BookX(100)
b2 =BookY(150)
print("Total pages ", b1+b2)
""" Output
Total pages 250 """
```

14.4.2 Operator Overloading

- Internal methods that can be overridden to act on objects are called as magic methods
- Following table lists operators and their magic methods

Operator	Magic Method	Operator	Magic Method
+	object.__add__(self,other)	/=	object.__idiv__(self,other)
-	object.__sub__(self,other)	//=	object.__ifloordiv__(self,other)
*	object.__mul__(self,other)	%=	object.__imod__(self,other)
/	object.__div__(self,other)	**=	object.__ipow__(self,other)
//	object.__floordiv__(self,other)	<	object.__lt__(self,other)
%	object.__mod__(self,other)	<=	object.__le__(self,other)
**	object.__pow__(self,other)	>	object.__gt__(self,other)
+=	object.__iadd__(self,other)	>=	object.__ge__(self,other)
-=	object.__isub__(self,other)	==	object.__eq__(self,other)
*=	object.__imul__(self,other)	!=	object.__ne__(self,other)

14.4.3 Method Overloading

- Method overloading is writing more than one method with the same name
- Eg

```
sum(int a, int b) {}
sum(int a, int b, int c) {}
```

- Method overloading is not available in python
- To implement method overloading in python
- Eg

```
Myclass:
        def sum(self, a=None, b=None, c=None):
                if a!=None and b!=None and c!=None:
                        print(“sum is ”, a+b+c)
                elif a!=None and b!=None:
                        print(“sum is ”, a+b)
# Call sum using objects
m=Myclass()
m.sum(10, 20, 30)
m.sum(25.3,23)
```

14.4.4 Method Overriding

- Already discussed under Inheritance section

Experiment 8:

Problem Statement:

Write a program to demonstrate single and multiple inheritance in python with method overriding