Quadrant II – Transcript and Related Materials

Programme: B.Sc.
Subject: Computer Science
Semester: III
Paper Code: CSS103
Paper Title: Programming in Python

Unit IV: Object Oriented Concepts
Module Name: Polymorphism, Encapsulation, Modifier
Module No: 14

Name of the Presenter: Mrs. Sonali Sarvesh Karapurkar

POLYMORPHISM

Polymorphism is the ability of any data to be processed in more than one form. The word itself indicates the meaning as poly means many and morphism means types. Polymorphism is one of the most important concept of object oriented programming language. The most common use of polymorphism in object-oriented programming occurs when a parent class reference is used to refer to a child class object. Here we will see how to represent any function in many types and many forms. Real life example of polymorphism, a person at the same time can have different roles to play in life. Like a woman at the same time is a mother, a wife, an employee and a daughter. So the same person has to have many features but has to implement each as per the situation and the condition. Polymorphism is considered as one of the important features of Object Oriented Programming.

Example of inbuilt polymorphic functions

```
# Python program to demonstrate in-built poly-morphic functions

# len() being used for a string
print(len("geeks"))

# len() being used for a list
print(len([10, 20, 30]))
```

Output:

```
5
3
```

Example of used defined polymorphic functions

```
# A simple Python function to demonstrate
# Polymorphism
def add(x, y, z = 0):
    return x + y + z
```

```
# Driver code
print(add(2, 3))
print(add(2, 3, 4))
```
Output:
5
9

Polymorphism with class methods

- Python can use two different class types, in the same way.
- We create a for loop that iterates through a tuple of objects.
- Then call the methods without being concerned about which class type each object is.
- We assume that these methods actually exist in each class.

Example:
```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")
    def language(self):
        print("Hindi is the most widely spoken language of India.")
    def type(self):
        print("India is a developing country.")
class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")
    def language(self):
        print("English is the primary language of USA.")
    def type(self):
        print("USA is a developed country.")
obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
    country.type()
```

Output:
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.

ENCAPSULATION

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. Those types of variables are known as **private variable**. A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc

Private methods

- We create a class Car which has two methods: drive() and updateSoftware().
- When a car object is created, it will call the private methods __updateSoftware().

Car
+drive() -_updateSoftware()

Example:

```
class Car:

def __init__(self):
self.__updateSoftware()

def drive(self):
print('driving')

def __updateSoftware(self):
print('updating software')

redcar = Car()
redcar.drive()
```

This program will output:

```
updating software
driving
```

PYTHON ACCESS MODIFIERS

Python makes the use of **underscores** to specify the access modifier for a specific data member and member function in a class. Access modifiers play an important role to protect the data from unauthorized access as well as protecting it from getting manipulated.

TYPES OF ACCESS MODIFIERS

Access Modifier: Public

The members declared as Public are accessible from outside the Class through an object of the class.

Access Modifier: Protected

The members declared as Protected are accessible from outside the class but only in a class derived from it that is in the child or subclass.

Access Modifier: Private

These members are only accessible from within the class. No outside Access is allowed.

Example:

```
# define parent class Company
class Company:
    # constructor
    def __init__(self, name, proj):
        self.name = name      # name(name of company) is public
        self._proj = proj     # proj(current project) is protected

    # public function to show the details
    def show(self):
        print("The code of the company is = ",self.ccode)

# define child class Emp
class Emp(Company):
    # constructor
    def __init__(self, eName, sal, cName, proj):
        # calling parent class constructor
        Company.__init__(self, cName, proj)
        self.name = eName   # public member variable
        self.__sal = sal    # private member variable

    # public function to show salary details
    def show_sal(self):
        print("The salary of ",self.name," is ",self.__sal,)

# creating instance of Company class
c = Company("Stark Industries", "Mark 4")
# creating instance of Employee class
e = Emp("Steve", 9999999, c.name, c._proj)
print("Welcome to ", c.name)
print("Here",e.name,"is working on",e._proj)
# to show the value of __sal we have created a public function show_sal()
e.show_sal()
```

Output:

```
Welcome to Stark Industries
Here Steve is working on Mark 4
The salary of Steve is 9999999
```