G2

OBJECT ORIENTED PROGRAMMING

01

Introduction

Introduction about OOPs & its concepts, Classes & Objects, and uses of OOPs in python, with example.

02

Inheritance

Explanation about Inheritance, its types, diagrams, with example.

03

Polymorphism

Explanation about Polymorphism, method overriding, method overloading with example.

04

Encapsulation

Explanation about Encapsulation, access modifiers, with example.

05

Abstraction

Explanation about Abstraction, Abstract class, with example.

GROUP-2

INTRODUCTION

01

Introduction

In Python object-oriented Programming (OOPs) is a programming model that uses objects and classes to design applications and computer programs. Even though Python is not purely an OOP language, it is flexible enough and provides enough features to support OOP principles.

02

Concepts

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Abstraction

03

Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It has attributes(variables) and methods(functions).

04

Objects

The object is an entity that has a state and behavior associated with it. They are real instances of a class that are unique & can be created many as you want.

GROUP-2

INTRODUCTION

05 Benefits of OOPs

- OOP models complex things as reproducible, simple structures.
- Reusable, OOP objects can be used across programs.
- Polymorphism allows for class-specific behavior.
- Easier to debug, classes often contain all applicable information to them.
- Securely protects sensitive information through encapsulation.

06 Example

```
# Creating a class
class student:
    # defining methods
    def names(self,name):
        print("Name:",name)

# Creating object for student class
obj = student()

# calling names method of student class
obj.names("John")
```

GROUP-2

INHERITANCE

01 Definition

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. It provides the reusability of a code, so we don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

02 Types of Inheritance

1. **Single Inheritance**: Child class derived from a Single Parent class.
2. **Multilevel Inheritance**: Child class derived from a Parent class that is a Child to another Parent class.
3. **Hierarchical Inheritance**: Multiple Child class derived from a Single-Parent class.
4. **Multiple Inheritance**: Child class derived from Multiple Parent class.

GROUP-2

INHERITANCE

03

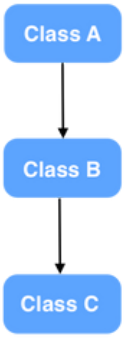
Diagrams

1. Single Inheritance:

Example:

```
class A:
    pass
class B(A):
    pass
```

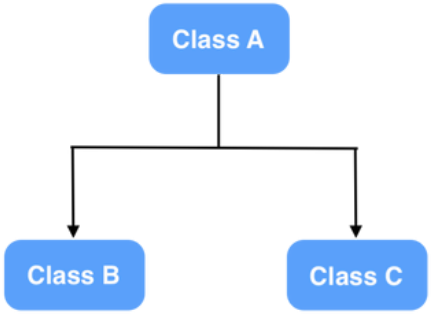
2. Multilevel Inheritance:



Example:

```
class A:
    pass
class B(A):
    pass
class C(B):
    pass
```

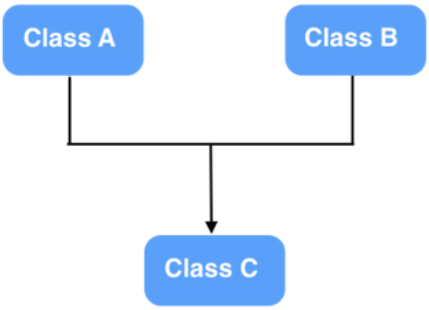
3. Hierarchical Inheritance:



Example:

```
class A:
    pass
class B(A):
    pass
class C(A):
    pass
```

4. Multiple Inheritance:



Example:

```
class A:
    pass
class B:
    pass
class C(A,B):
    pass
```

GROUP-2

INHERITANCE

04

Example

```
# Parent class
class school:
    def student(self):
        print("student")
    def teacher(self):
        print("teacher")
    def classes(self):
        print("classes")

# Child class
class names(school):
    def student_name(self):
        print("student_name")
    def teacher_name(self):
        print("teacher_name")
    def class_name(self):
        print("class_name")

# Creating an object of the child class
obj = names()

# Calling methods of the parent using the child class object
obj.student()
obj.teacher()
obj.classes()

# Calling methods of the child class
obj.student_name()
obj.teacher_name()
obj.class_name()
```

Output

```
\Inheritance.py"
student
teacher
classes

student_name
teacher_name
class_name
```

GROUP-2

POLYMORPHISM

01 Definition

Polymorphism simply means having many forms. In programming, polymorphism means the same function name being used for different types.
EX: len() function can be used to find the length of a string, list, tuple, etc.

02 Example

```
# len(variable or value), gives their length

var = "python"
print(len(var)) # 6

var = [1,2,3,4,5]
print(len(var)) # 5

var = (1,2,3)
print(len(var)) # 3

# we can see that len function worked for all the given data types,
# i.e one name many functions
```

03 Types of Polymorphism

1. **Method Overriding:** refers to defining a method in a subclass with the same name as a method in its superclass.

2. **Method Overloading:** a way to create multiple methods with the same name but different arguments, this is not directly possible to achieve in Python.

GROUP-2

POLYMORPHISM

04

Example Method Overriding

```
# Defining parent class
class Bird:
    # Defining methods
    def intro(self):
        print("There are many types of birds.")
    def flight(self):
        print("Most of the birds can fly but some cannot.")

# Defining child classes of Bird class
class sparrow(Bird):
    # Overriding flight method
    def flight(self):
        print("Sparrows can fly.")

# Creating object for Bird class
obj_bird = Bird()
# Calling methods of Bird class
obj_bird.intro()
# Before Overriding
obj_bird.flight()

# Creating object for sparrow class
obj_spr = sparrow()
# Calling methods of sparrow class
obj_spr.intro()
# After Overriding
obj_spr.flight()
```

Output

```
\Method_Overriding.py"
There are many types of birds.
Most of the birds can fly but some cannot.

There are many types of birds.
Sparrows can fly.
```

GROUP-2

POLYMORPHISM

04

Example Method Overloading

```
# *args is a function to take multiple arguments
def add(datatype, *args):

    # if datatype is int initialize answer as 0
    if datatype == 'int':
        answer = 0

    # if datatype is str initialize answer as ''
    if datatype == 'str':
        answer = ''

    # calling each arguments separately
    for x in args:
        # this will do addition or concatenation according to the datatype
        answer = answer + x

    print(answer)


# Integer
add('int', 5, 6)

# String
add('str', 'Hi ', 'Python')
```

Output

```
\Method_Overloading.py"
11
Hi Python
```

GROUP-2

ENCAPSULATION

01 Definition

Encapsulation describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. It uses access modifiers to achieve this.

Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members.
ex: class

02 Types of Access Modifiers

1. **Public Access Modifier:** public methods and attributes can be accessed directly by any class.

2. **Protected Access Modifier:** protected methods and attributes can be accessed within the same class it is declared and its subclass. A single underscore "_" is used to describe a protected data member or method of the class.

3. **Private Access Modifier:** private methods and attributes can be only accessed within the same class it is declared. Double underscore "__" is used to describe a private data member or method of the class.

GROUP-2

ENCAPSULATION

03

Example

```
# Creating a Parent class
class Base:
    def __init__(self):
        self.a = "public" # no underscore
        self._b = "protected" # single underscore
        self.__c = "private" # two underscores

# Creating a Child class
class Derived(Base):
    def __init__(self):
        # Calling constructor of Base class
        Base.__init__(self)
        # accessing public variable
        print("Calling public member of base class: ",self.a)
        # accessing protected variable
        print("Calling protected member of base class: ",self._b)
        # accessing private variable, shows error message
        print("Calling private member of base class: ",self.__c)

# Creating an object of the child class
obj = Derived()
```

Output

```
Calling public member of base class:  public
Calling protected member of base class:  protected
Traceback (most recent call last):
  File "c:\PRGM\python\python luminar\class work\seminar\Encapsulation.py", line 21, in <module>
    obj = Derived()
```

GROUP-2

ABSTRACTION

01 Definition

Data abstraction in Python is used to hide irrelevant details from the user and show the details that are relevant to the users. It enables programmers to hide complex implementation details while just showing users the most crucial data and functions. This abstraction makes it easier to design modular and well-organized code, makes it simpler to understand and maintain, promotes code reuse, and improves developer collaboration. It hides unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.

02 Abstract Class

A class is said to be an abstract class if it cannot be instantiated, that is you can have an object of an abstract class. You can however use it as a base or parent class for constructing other classes.To create an abstract class in Python, it must inherit the ABC class that is defined in the ABC module.

GROUP-2

ABSTRACTION

03

Example

```
# importing modules to use abstraction
from abc import ABC,abstractmethod

# creating an abstract class, cannot be called directly
class Animal(ABC):
    # defining an abstract method in the abstract class
    @abstractmethod
    def move(self):
        pass

# creating child classes inheriting from the abstract class
class Dog(Animal):
    # overriding the abstract method in the child class
    def move(self):
        print("I protect my home.")

# creating child classes inheriting from the abstract class
class Cat(Animal):
    # overriding the abstract method in the child class
    def move(self):
        print("I do nothing.")

# creating objects of Dog & Cat class and calling the move method
obj_dog = Dog()
obj_dog.move()
obj_cat = Cat()
obj_cat.move()
```

Output

```
\Abstract_Class.py"
I protect my home.
I do nothing.
```

GROUP-2