Programming Basics and Data Analytics with Python

simplilearn

OOPs Concepts with Python

Learning Objectives

By the end of this lesson, you will be able to:

- Explain OOP and its characteristics
- Identify objects and classes
- Describe methods, attributes, and access modifiers
- Define abstraction, encapsulation, inheritance, and polymorphism with real-life examples

DATA AND
ARTIFICIAL INTELLIGENCE

Object-Oriented Programming Language

What Is OOPs?

OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities such as inheritance, hiding, and polymorphism in programming.

Characteristics of OOPs

- OOPs uses a bottom-up approach.
- The program is divided into objects.
- OOPs uses access modifiers.
- OOPs is more secure than procedural languages.
- Objects can move freely within member functions.

OOPS Concepts

Abstraction

Encapsulation

Inheritance

Polymorphism

DATA AND
ARTIFICIAL INTELLIGENCE

Objects and Classes

Objects

Object-oriented programming (OOP) involves programming using objects. An object represents an entity in the real-world that can be distinctly identified.

An object consists of identity, state, and behavior.

- Identity: It gives a unique name to an object.
- State: It reflects the properties of an object.
- Behavior: It reflects the response of an object with other objects.

Example of an object: Dog

Identity	State/Attribute	Behavior
Name of the dog	Breed	Bark
	Age	Sleep
	Color	Eat

Classes

A class is a blueprint for an object. Objects with the same properties and methods are defined inside the class.

```
class Dog:
    pass
```

Here, we use the keyword **class** to define an empty **class Dog**.

Note

We construct instances from a class. An instance is a specific object created from a particular class.

DATA AND
ARTIFICIAL INTELLIGENCE

Methods and Attributes

Methods

Methods are functions defined inside a class. They are invoked by objects to perform actions on other objects.

__init__ is a method that is automatically called when memory is allocated to a new object.

```
# A sample class with init method
class Person:

    # init method or constructor
    def __init__(self, name):
        self.name = name
```

In the init method, **self** refers to the newly created object. In other class methods, it refers to the instance whose method was called.

Attributes

Attributes are defined within a class and outside any method. Attributes define the characteristics of any object.

Identity	Attributes
Name of dog	Breed
	Age
	Color

DATA AND
ARTIFICIAL INTELLIGENCE

Access Modifiers

Access Modifiers

A class in Python has three types of access modifiers.

- Public Access Modifiers
- Protected Access Modifiers
- Private Access Modifiers

Access Modifiers

Public Access Modifiers

Protected Access Modifiers

Private Access Modifiers

Members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.

Example:

```
class Dog:

    # constructor
    def __init__(self, name, age):

        # public access modifiers
        self.dogName = name
        self.dogage = age
```

Access Modifiers

- Public Access Modifiers
- Protected Access Modifiers
- Private Access Modifiers

Members of a class that are declared protected are only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore symbol (_) before the data member of that class.

Example:

```
class Dog:
    # protected access modifiers
    _name = None
    _age = None
    _breed = None
```

Access Modifiers

Public Access Modifiers

Protected Access Modifiers

Private Access Modifiers

Members of a class that are declared private are accessible within the class only. Private access modifier is the most secure access modifier. Data members of a class are declared private by adding a double underscore symbol (__) before the data member of that class.

Example:

```
class Dog:
      # private access modifiers
    __name = None
    __age = None
    __breed = None
```

Objects and Classes

Problem Statement: Write a program to demonstrate objects and classes using methods and attributes.

Steps to perform:

1. Create a class
2. Declare the attributes
3. Make a method
4. Initiate the objects
5. Access class attributes and method through objects

ASSISTED PRACTICE

DATA AND
ARTIFICIAL INTELLIGENCE

Abstraction

Abstraction

Abstraction is the property by virtue of which only the essential details are displayed to the user.

Example: When you press a key on the keyboard, the relevant character appears on the screen. It is not essential for you to know how exactly this works. This is called abstraction.

Abstraction

Problem Statement: Write a program to demonstrate abstraction using classes, objects, and methods.

Steps to Perform:

1. Create a class called Bill
2. Define a function to initialize variables
3. Create a class called Debit Card Payment
4. Create an object from the class Debit Card Payment
5. Create an object from the class Bill
6. Check whether the object is an instance of class Bill or not

ASSISTED PRACTICE

DATA AND
ARTIFICIAL INTELLIGENCE

Encapsulation

Encapsulation

Encapsulation is a process of binding data members and member functions into a single unit. Encapsulation hides the state of a structured data object inside a class, preventing unauthorized access to an unauthorized person.

Example: Only the chemist in a drug store has access to medicines. This reduces the risk of unauthorized people taking any unintended medicines.

Example:

```
class Encapsulation:
    def __init__(self, a, b,c):
        self.public = a
        self._protected = b
        self.__private = c
```

Encapsulation

Problem Statement: Write a program to demonstrate encapsulation using classes, objects, and methods.

Steps to Perform:

1. Create a class called Employee
2. Declare variables in the initiation function
3. Make a method to print the variables
4. Create an object for the class Employee
5. Call the display method

ASSISTED PRACTICE

DATA AND
ARTIFICIAL INTELLIGENCE

Inheritance

Inheritance

Inheritance is the process of forming a new class from an existing class or a base class. The base class is also known as parent class or super class. The new class that is formed is called derived class.

Example: A family has three members: father, mother, and son

Father (Base class)
Tall
Dark

Mother (Base class)
Short
Fair

Son (Derived class)
Tall
Fair

The son is tall and fair. This indicates that he has inherited the features of his father and mother, respectively.

Types of Inheritance

Single inheritance:
A class can inherit from only one class.

Multilevel inheritance:
A derived class is created from another derived class.

Multiple inheritance:
A class can inherit from more than one class.

Hierarchical inheritance:
More than one sub class is inherited from a single base class.

Inheritance

Problem Statement: Write a program to demonstrate inheritance using classes, objects, and methods.

Steps to Perform:

1. Create a base class
2. Create a derived class
3. Call the constructor of the base class
4. Multiply the variable of the base class and the derived class

ASSISTED PRACTICE

DATA AND
ARTIFICIAL INTELLIGENCE

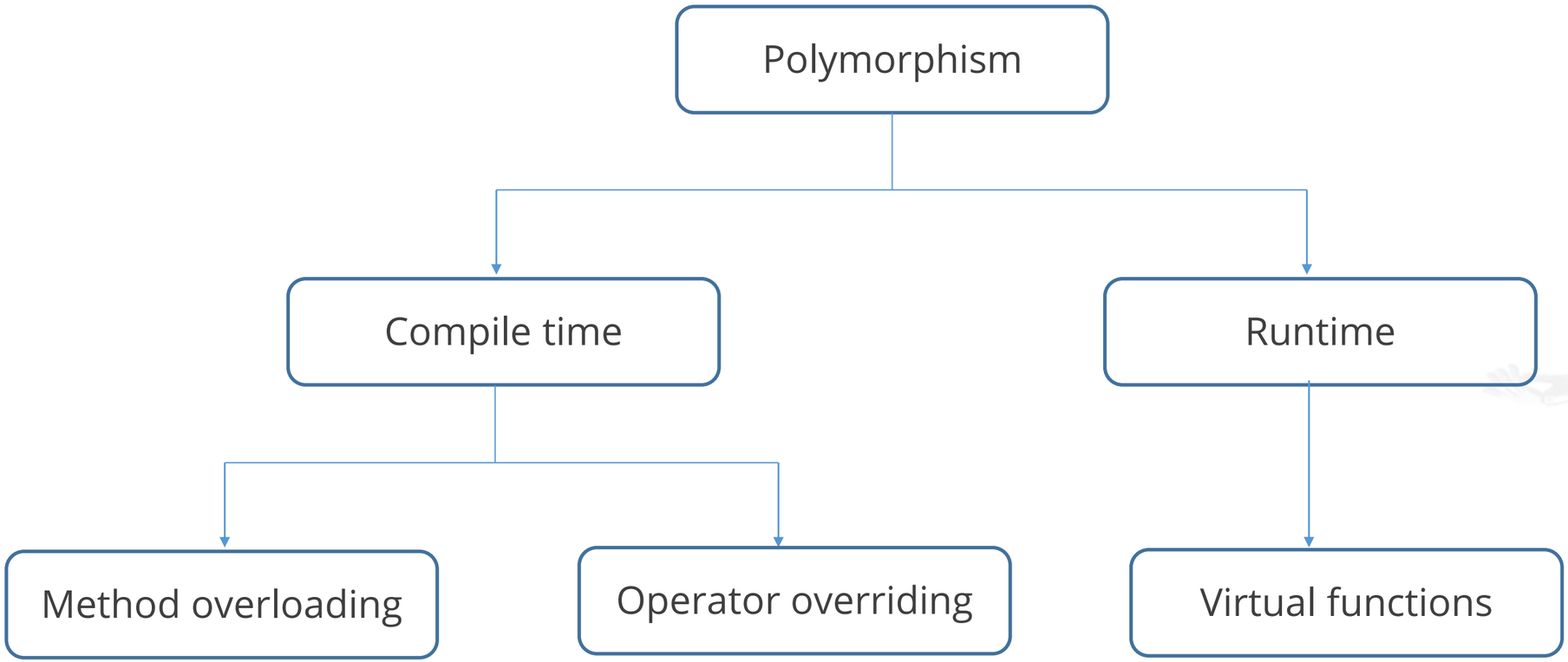
Polymorphism

Polymorphism

Polymorphism is a Greek word that means *many shaped*. Polymorphism is the ability of a message to be displayed in more than one form.

Example: A woman can be a mother, a wife, a daughter, a teacher, and an employee at the same time.

Types of Polymorphism



Polymorphism

Problem Statement: Write a program to demonstrate polymorphism using classes, objects, and methods.

Steps to Perform:

1. Create a class called Ohio
2. Make a method to print a statement
3. Create a class called California
4. Make a method to print a statement
5. Make two objects of the two classes having the same method

ASSISTED PRACTICE

Knowledge Check

Knowledge Check

1

An object is an instance of a(n) _____.

a. Method

b. Attribute

c. Class

d. Function

Knowledge Check

1

An object is an instance of a(n) _____.

a. Method

b. Attribute

c. Class

d. Function

The correct answer is **c**

An object is an instance of a class.

Knowledge Check

2

Which of the following is NOT an OOPs concept?

a. Inheritance

b. Compilation

c. Abstraction

d. Encapsulation

Knowledge Check

2

Which of the following is NOT an OOPs concept?

a. Inheritance

b. Compilation

c. Abstraction

d. Encapsulation

The correct answer is **b**

There are four OOPS concepts: Inheritance, Encapsulation, Polymorphism, and Abstraction.

Knowledge Check

3

Which of the following is a type of polymorphism?

a. Compile time polymorphism

b. Runtime polymorphism

c. Multiple polymorphism

d. Multilevel polymorphism

Knowledge Check

3

Which of the following is a type of polymorphism? (Select all that apply)

a. Compile time polymorphism

b. Runtime polymorphism

c. Multiple polymorphism

d. Multilevel polymorphism

The correct answer is **a and b**

The types of polymorphism are compile time polymorphism and runtime polymorphism.

Key Takeaways

- Object-oriented programming aims to implement real-world entities such as inheritance, hiding, and polymorphism in programming.
- An object is an instance of a class.
- A class is a blueprint for an object. A class is a definition of objects with the same properties and methods.
- A class in Python has three types of access modifiers: public, protected, and private.