**OJT-1**

**Python Programing with OOP’s**

Python is a high-level, interpreted programming language that emphasizes code readability and simplicity. It was created by Guido van Rossum and first released in 1991. Python is known for its elegant syntax and easy-to-understand code, making it a popular choice for beginners and experienced developers alike.

Python is a versatile and popular programming language known for its simplicity and readability. It supports various programming paradigms, including procedural, functional, and object-oriented programming (OOP). Object-Oriented Programming is a powerful approach to software development that focuses on organizing code into reusable objects, enabling modular and maintainable code. This report provides a detailed overview of Python programming with an emphasis on OOP principles, concepts, and implementation

**Object-Oriented Programming (OOP):**

Object-Oriented Programming is a programming paradigm that provides a structured way to design and build software. It revolves around the concept of objects, which are instances of classes. A class serves as a blueprint or template for creating objects, defining their attributes (variables) and behaviors (methods).

**1. Classes and Objects**

In OOP, a class represents a real-world entity or concept. It defines the structure and behavior that objects of that class will possess. An object, on the other hand, is an instance of a class, representing a specific entity or instance of the concept described by the class.

To create a class in Python, you use the **class** keyword followed by the class name. Within the class, you can define attributes (data variables) and methods (functions) that describe the behavior of objects created from that class. Objects are created by calling the class as if it were a function, which invokes the class's constructor method and returns an object.

**2. Encapsulation**

Encapsulation is a fundamental principle of OOP that combines data and functions into a single unit called a class. It allows you to hide the internal details of a class and provide controlled access to the class members. This data hiding protects the integrity of the data and prevents direct manipulation from outside the class.

Python provides access modifiers like public, private, and protected to control the visibility and accessibility of class members. By convention, attributes and methods prefixed with a single underscore **\_** are considered protected, and those prefixed with double underscores **\_\_** are considered private.

Encapsulation promotes data abstraction, where the internal implementation details of a class are hidden and only the essential information and functionality are exposed to the user.

**3. Inheritance**

Inheritance is a mechanism that allows a class to inherit attributes and methods from another class, called the base class or parent class. The class inheriting from the base class is called the derived class or child class. Inheritance facilitates code reuse and promotes the concept of hierarchical classification.

To inherit from a base class in Python, you include the base class name in parentheses after the derived class name in the class definition. The derived class can then access the attributes and methods of the base class and can also override or extend them to provide specialized behavior.

Inheritance enables the creation of specialized classes that inherit and extend the functionality of more general classes, promoting code extensibility and flexibility.

**4. Polymorphism**

Polymorphism is the ability of objects of different classes to be treated as objects of a common base class. It allows you to write code that can work with objects of different types but treats them uniformly based on their shared interface or behavior.

Polymorphism in Python is achieved through method overriding and method overloading. Method overriding allows the derived class to provide its own implementation of a method inherited from the base class. This allows you to customize the behavior of a method based on the specific requirements of the derived class.

Method overloading, although not directly supported in Python, can be achieved by using default parameter values or variable-length arguments. This allows you to define multiple methods with the same name but different parameter lists, giving the appearance of method overloading.

**Implementation of OOP in Python**

Python provides a rich set of tools and syntax for implementing OOP concepts effectively.

**1. Class Definition**

In Python, a class is defined using the **class** keyword followed by the class name and a colon. The class body is indented, and it contains attribute and method definitions. Attributes are variables defined within a class, and methods are functions defined within a class that define its behavior.

**2. Constructor and Destructor**

A constructor is a special method that is automatically called when an object is created from a class. In Python, the constructor method is named **\_\_init\_\_()** and is used to initialize the attributes of the object. It allows you to set the initial state of the object and perform any necessary setup operations.

A destructor method, **\_\_del\_\_ ()**, can be defined to perform cleanup operations before an object is destroyed and memory is released. The destructor is automatically called when the object is no longer referenced or goes out of scope.

**3. Inheritance Syntax**

To create a derived class that inherits from a base class, you include the base class name in parentheses after the derived class name in the class definition. The derived class can then access the attributes and methods of the base class using the dot notation.

**4. Method Overriding**

Method overriding allows the derived class to provide its own implementation of a method inherited from the base class. In Python, this is achieved by defining a method with the same name in the derived class. When the method is called on an object of the derived class, the overridden method in the derived class is executed instead of the base class method.

To override a method in Python, you define a method with the same name in the derived class. The method signature (name and parameters) must match the method being overridden in the base class.Method overriding allows you to customize the behavior of a method based on the specific requirements of the derived class. It is a fundamental feature of object-oriented programming that supports code extensibility and flexibility

**5. Method Overloading**

Python does not support method overloading in the traditional sense, where multiple methods with the same name but different parameters are defined. However, you can achieve similar functionality by using default parameter values or variable-length arguments.

Default Parameter Values: You can define a method with default parameter values, allowing the method to be called with different numbers of arguments Variable-Length Arguments: Python provides the **\*args** and **\*\*kwargs** syntax to handle variable-length arguments. The **\*args** allows you to pass a variable number of non-keyword arguments, while **\*\*kwargs** allows you to pass a variable number of keyword arguments. This enables you to define methods that can accept different numbers of arguments

**Benefits of OOP in Python**

Using OOP in Python offers several advantages:

**1. Reusability:**

OOP promotes reusability by allowing the creation of reusable objects and classes. Objects can be instantiated from classes and reused in different parts of the program or in different programs altogether. This reduces code duplication and improves development efficiency.

**2. Modularity:**

OOP enables the modular organization of code. Classes encapsulate data and related methods into self-contained units. This modular structure makes code easier to understand, test, and maintain. It also allows for easier collaboration among developers working on different parts of a project.

**3. Flexibility and Extensibility:**

Inheritance, a key feature of OOP, allows for easy modification and extension of existing code. New classes can be created that inherit and reuse the functionality of base classes. This promotes code extensibility and reduces development effort by building upon existing code rather than starting from scratch.

**4. Encapsulation and Information Hiding:**

Encapsulation, a core principle of OOP, encapsulates data and methods within a class, hiding the internal implementation details. This provides data security and prevents direct manipulation of class members from outside the class. Encapsulation also allows for better code maintenance and updates, as the internal implementation can be modified without affecting the code using the class.

**5. Improved Code Organization and Design:**

OOP promotes better code organization and design by providing clear structures for managing complexity. Classes and objects help break down complex systems into smaller, more manageable components. This enhances code readability, understandability, and maintainability.

**6. Polymorphism and Code Flexibility:**

Polymorphism, another important concept in OOP, allows objects of different types to be treated uniformly based on their shared interface or behavior. This promotes code flexibility and modularity, as different objects can be used interchangeably in code that relies on their common interface. Polymorphism simplifies code design and enhances code reusability.

**7. Improved Collaboration and Code Maintenance:**

OOP facilitates collaboration among developers in large-scale projects. By dividing the project into classes and objects, different team members can work on different parts of the project independently. Changes or updates to one class do not affect other classes, as long as the interface remains unchanged. This improves code maintenance, scalability, and team productivity.

Overall, OOP provides a powerful and efficient approach to software development, offering benefits such as reusability, modularity, flexibility, code organization, and collaboration. These benefits contribute to improved code quality, development productivity, and maintainability of software systems.

**Important Function of Python.**

**1.Map**

The **map()** function in Python is used to apply a given function to each item in an iterable (such as a list) and returns an iterator containing the results.The **map()** function takes each item from the **iterable**, applies the **function** to it, and returns an iterator that yields the results. It is commonly used to transform or modify the elements of a list in a concise and efficient way.

**2. Filter:**

The **filter()** function in Python is used to filter out elements from an iterable based on a specified condition. It returns an iterator that contains the elements for which the condition is True.The **filter()** function applies the **function** to each element in the **iterable** and retains only the elements for which the **function** returns True. It effectively filters out elements that do not satisfy the specified condition.

**3. Reduce:**

The **reduce()** function is part of the **functools** module in Python. It is used to apply a specified function to the elements of an iterable in a cumulative way. The **reduce()** function performs a repetitive operation on pairs of elements until a single value is obtained.The **reduce()** function starts by applying the **function** to the first two elements of the **iterable**. It then takes the result and combines it with the next element, repeating the process until all the elements are processed. The final output is a single value that represents the cumulative result.

**4. Lambda Functions:**

A lambda function is a small, anonymous function in Python. It is defined using the **lambda** keyword and can take any number of arguments but can only have one expression. Lambda functions are typically used when a function is required for a short duration and does not need to be defined using a regular **def** statement.Lambda functions are often used in conjunction with higher-order functions like **map()**, **filter()**, and **reduce()** to provide a concise and inline way of defining functions without the need for a separate function definition.

Lambda functions are useful in scenarios where a simple function is required, such as when the function logic is short and straightforward, or when a function is used as an argument to another function.

These functional programming tools (map, filter, reduce, and lambda) in Python provide powerful and concise ways to manipulate data and perform operations on iterable objects. They enhance code readability and enable more expressive and efficient programming.

**Use case-1**

**Salary Increase and Elderly Employee Identification**

In a company, there is a list of employees with their names, ages, and salaries. The HR department wants to perform two operations on the employee data: increase the salary of each employee by 10% and identify the elderly employees (those with an age of 60 or greater).

The code snippet above represents a solution to this use case. Here's how it works:

1. The employees list is created, containing four Employee objects with different attributes (name, age, and salary).
2. Using the map function, the salary of each employee is increased by 10% by applying a lambda function that calls the get\_salary\_increase method on each employee.
3. The filter function is used to obtain a list of only the elderly employees by applying a lambda function that checks if the employee's age is 60 or greater.
4. The filtered list of elderly employees is stored in the elderly\_employees variable.
5. Finally, the details (name, age, and salary) of each elderly employee are printed using a loop.

This use case demonstrates how the Employee class and the map and filter functions can be utilized to efficiently manipulate and analyze employee data in a company setting.

**Problem Statement: Employee Salary Increase and Elderly Employees**

You are tasked with developing a Python program to handle employee data for a company. The program should provide functionality to increase the salary of each employee by a given percentage and identify the elderly employees based on their age.

You need to implement an Employee class with the following attributes and methods: Attributes:

name (str): The name of the employee.

age (int): The age of the employee.

salary (float): The salary of the employee.

Methods:

\_\_init\_\_(self, name, age, salary): Initializes a new employee with the given name, age, and salary.

get\_name(self): Returns the name of the employee.

get\_age(self): Returns the age of the employee.

get\_salary(self): Returns the salary of the employee.

get\_salary\_increase(self, percentage): Increases the salary of the employee by the specified percentage.

is\_elderly(self): Returns True if the employee's age is 60 or greater, and False otherwise.

You also need to provide a sample code snippet that demonstrates the usage of the class by creating a list of Employee objects and performing various operations on them using map and filter. Specifically, you should use map to increase the salary of each employee by 10% and use filter to obtain a list of only the elderly employees (those with an age of 60 or greater).

Ensure that the implementation is modular, efficient, and follows best practices for object-oriented programming in Python.

Your program should be able to handle any number of employees and provide accurate results for salary increases and identification of elderly employees.

Write a detailed problem statement that clearly explains the requirements and expected behavior of the program.

**AI implementation**

1. Intelligent Decision-Making: The program allows users to add books, display books by author, calculate the total number of pages, display a table of books, and remove books. These actions
2. involve decision-making based on user input and data manipulation. Although the program does not employ advanced AI techniques, it demonstrates basic decision-making capabilities.
3. Data Analysis: The program maintains a list of books and their details, such as titles, authors, and page counts. AI algorithms could be employed to analyze this data, identify patterns, and generate insights. For instance, data analysis techniques could help identify popular authors, genres, or trends in reading habits.
4. Natural Language Processing (NLP): Use NLP techniques to enhance search capabilities, allowing users to search for bank accounts using natural language queries.
5. Recommendation System: Implement a recommendation system that suggests accounts to users based on their preferences, reading history, or other factors.
6. In this AI implementation, the code creates an Employee class with attributes name, age, and salary. It also includes methods to get employee information, increase the salary by a given percentage, and check if an employee is elderly (age 60 or above).
7. The code then creates a list of employees and applies a salary increase to each employee using map() and a lambda function. After that, it filters the list to obtain only the elderly employees using filter() and another lambda function.
8. Finally, it prints the information (name, age, and salary) of the elderly employees using a for loop and the get\_name(), get\_age(), and get\_salary() methods.

**Code for Employee Salary Increase and Elderly Employees :**

class Employee:

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

def get\_name(self):

return self.name

def get\_age(self):

return self.age

def get\_salary(self):

return self.salary

def get\_salary\_increase(self, percentage):

self.salary \*= (1 + percentage / 100)

def is\_elderly(self):

return self.age >= 60

employees = [

Employee("John Doe", 45, 5000.0),

Employee("Jane Smith", 55, 6000.0),

Employee("Bob Johnson", 65, 7000.0),

Employee("Alice Williams", 70, 8000.0),

]

salary\_increase\_percentage = 10

list(map(lambda emp: emp.get\_salary\_increase(salary\_increase\_percentage), employees))

elderly\_employees = list(filter(lambda emp: emp.is\_elderly(), employees))

for emp in elderly\_employees:

print("Name:", emp.get\_name())

print("Age:", emp.get\_age())

print("Salary:", emp.get\_salary())

print()

**Explanation of code :**

The Employee class represents an employee of a company and has the following attributes:

* name (str): The name of the employee.
* age (int): The age of the employee.
* salary (float): The salary of the employee.

The class also provides the following methods:

* \_\_init\_\_(self, name, age, salary): This is the constructor method that initializes a new employee with the given name, age, and salary.
* get\_name(self): Returns the name of the employee.
* get\_age(self): Returns the age of the employee.
* get\_salary(self): Returns the salary of the employee.
* get\_salary\_increase(self, percentage): Increases the salary of the employee by the specified percentage. It takes a percentage as a parameter, converts it to a decimal value, and updates the salary attribute accordingly.
* is\_elderly(self): Returns True if the employee's age is 60 or greater, and False otherwise.

1. The sample code snippet demonstrates the usage of the Employee class by performing operations on a list of Employee objects.
2. A list called employees is created, containing four Employee objects with different attributes (name, age, and salary).
3. The map function is used to increase the salary of each employee in the employees list by 10%. A lambda function is applied to each employee, calling the get\_salary\_increase method with the specified percentage.
4. The filter function is used to create a new list called elderly\_employees, which contains only the elderly employees from the original list. A lambda function is applied to each employee, checking if their age is 60 or greater using the is\_elderly method.
5. Finally, a loop is used to iterate over the elderly\_employees list and print the details (name, age, and salary) of each elderly employee.

The code snippet demonstrates how the Employee class can be used to manage employee data. It showcases the use of map to apply a salary increase to each employee and filter to obtain a subset of employees based on a condition. The output displays the details of the elderly employees after the salary increase.

**Output for code :**

