**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**

**Music Player Application**

This use case demonstrates the usage of the MusicPlayer class to create a music player application. The application allows users to manage their playlist, play, pause, resume, stop songs, and add new songs to the playlist.

**Use Case Steps:**

* The user launches the music player application.
* The application initializes the MusicPlayer class with an existing playlist.
* The application presents the user with a menu of actions to choose from.
* The user selects an action:
* Play a Song: The user provides the song number from the playlist. The application checks if the song exists in the playlist and plays it if found.
* Pause the Current Song: The application pauses the currently playing song, if any.
* Resume the Current Song: The application resumes the currently paused song, if any.
* Stop the Current Song: The application stops the currently playing or paused song, if any.
* Add a Song to the Playlist: The user provides the name of the song to add. The application checks if the song already exists in the playlist and adds it if not.
* Exit: The user exits the music player application.
* The application performs the selected action and displays the appropriate message or performs the corresponding functionality.
* The application loops back to the menu, allowing the user to select another action.
* The application continues to process user actions until the user chooses to exit.

**Problem Statement: Music Player Application**

You have been tasked with developing a music player application that allows users to manage their playlist and control the playback of songs. The application should provide a user-friendly interface with various options to play, pause, resume, stop songs, and add new songs to the playlist.

**Requirements:**

1. The application should implement a MusicPlayer class that encapsulates the functionality of the music player.
2. The MusicPlayer class should have the following attributes:

* playlist: A list containing the names of songs in the playlist.
* current\_song: A string representing the name of the currently playing song.

1. The MusicPlayer class should have the following methods:

* play(song): Plays the specified song if it exists in the playlist.
* pause(): Pauses the currently playing song, if any.
* resume(): Resumes the currently paused song, if any.
* stop(): Stops the currently playing or paused song, if any.
* add\_song(song): Adds a new song to the playlist if it is not already present.
* get\_playlist(): Returns the current playlist.

1. The application should allow the user to interact with the music player through a menu-driven interface.
2. The menu should provide the following options:

* Play a Song: Allows the user to enter the song number from the playlist to play.
* Pause the Current Song: Pauses the currently playing song.
* Resume the Current Song: Resumes the currently paused song.
* Stop the Current Song: Stops the currently playing or paused song.
* Add a Song to the Playlist: Allows the user to enter the name of a new song to add to the playlist.
* Exit: Terminates the application.

1. The application should validate user input and provide appropriate error messages for invalid choices or inputs.

**Constraints:**

* The playlist can contain any number of songs.
* Song names are unique within the playlist.
* The application should handle cases where the user tries to perform actions on a non-existent song or when no song is currently playing or paused.

**AI implementation**

In this AI-generated implementation, the code defines a MusicPlayer class that represents a simple music player. It has methods to play a song, pause the current song, resume a paused song, stop the current song, and add a new song to the playlist.

The code creates an instance of the MusicPlayer class with an initial playlist. Then, it enters a while loop that presents a menu of options to the user and performs the selected action based on the user's input.

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

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.

2. 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.

3.Natural Language Processing (NLP): Use NLP techniques to enhance search capabilities, allowing users to search for bank accounts using natural language queries.

**CODE OF MusicPlayer SYSTEM**

class MusicPlayer:

def \_\_init\_\_(self, playlist):

self.playlist = playlist

self.current\_song = ''

def play(self, song):

if song in self.playlist:

self.current\_song = song

print("Now playing: " + song)

else:

print(song + " is not in the playlist.")

def pause(self):

if self.current\_song:

print("Paused: " + self.current\_song)

else:

print("No song is currently playing.")

def resume(self):

if self.current\_song:

print("Resuming: " + self.current\_song)

else:

print("No song is currently paused.")

def stop(self):

if self.current\_song:

print("Stopped: " + self.current\_song)

self.current\_song = ''

else:

print("No song is currently playing.")

def add\_song(self, song):

if song not in self.playlist:

self.playlist.append(song)

print("Added " + song + " to the playlist.")

else:

print(song + " is already in the playlist.")

# Create an instance of MusicPlayer

playlist = ['song1.mp3', 'song2.mp3', 'song3.mp3']

player = MusicPlayer(playlist)

while True:

print("Choose an action:")

print("1. Play a song")

print("2. Pause the current song")

print("3. Resume the current song")

print("4. Stop the current song")

print("5. Add a song to the playlist")

print("0. Exit")

choice = input("Enter your choice: ")

if choice == "1":

song\_index = int(input("Enter the song number to play: ")) - 1

if song\_index >= 0 and song\_index < len(playlist):

player.play(playlist[song\_index])

else:

print("Invalid song number.")

elif choice == "2":

player.pause()

elif choice == "3":

player.resume()

elif choice == "4":

player.stop()

elif choice == "5":

new\_song = input("Enter the name of the song to add: ")

player.add\_song(new\_song)

elif choice == "0":

break

else:

print("Invalid choice. Please try again.\n")

**Explaning of the code:**

1. The MusicPlayer class is defined with an \_\_init\_\_ method, which initializes the player with a playlist and sets the current\_song to an empty string.
2. The play method takes a song as input. If the song is present in the playlist, it sets the current\_song to the input song and prints a message indicating that the song is being played. Otherwise, it prints a message stating that the song is not in the playlist.
3. The pause method checks if there is a current song playing. If so, it prints a message indicating that the current song has been paused. If no song is currently playing, it prints a message stating that there is no song playing.
4. The resume method checks if there is a current song that has been paused. If so, it prints a message indicating that the current song has been resumed. If no song is currently paused, it prints a message stating that there is no song paused.
5. The stop method checks if there is a current song playing. If so, it prints a message indicating that the current song has been stopped and sets the current\_song to an empty string. If no song is currently playing, it prints a message stating that there is no song playing.
6. The add\_song method takes a song as input. If the song is not already in the playlist, it appends the song to the playlist and prints a message indicating that the song has been added. If the song is already in the playlist, it prints a message stating that the song is already in the playlist.
7. An instance of the MusicPlayer class is created with a predefined playlist.
8. The code enters an infinite loop using while True. It displays a menu of options for the user to choose from.
9. The user's choice is captured through the input function and stored in the choice variable.
10. Based on the user's choice, the corresponding action is performed:

* If the choice is "1", the user is prompted to enter the song number to play. If the input is a valid song number within the range of the playlist, the play method is called with the corresponding song from the playlist.
* If the choice is "2", the pause method is called.
* If the choice is "3", the resume method is called.
* If the choice is "4", the stop method is called.
* If the choice is "5", the user is prompted to enter the name of the song to add, and the add\_song method is called with the input song.
* If the choice is "0", the loop is terminated, and the program exits.
* If the choice is not a valid option, an error message is printed.
* After each action, the loop repeats, displaying the menu again for the user to choose another action.

**Output of LIBRARY MANAGEMENT code**



