**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**

**LIBRARY MANAGEMENT SYSTEM**

A library management system is a software tool that helps libraries organize and manage their resources efficiently. It simplifies tasks such as cataloging, circulation, and inventory management, making it easier for librarians to handle day-to-day operations. Here's a breakdown structure of a library management system:

1. **User Interface**:

User Registration: The system allows users, such as students, faculty, and staff, to register and obtain a unique identifier, such as a library card or username, to access library services.

Login/Authentication: Allows library staff to securely log in to the system using their credentials.

Dashboard: Provides an overview of the system and access to different modules and features.

Search and Browse: Enables users to search and browse the library catalog by title, author, keywords, or other criteria.

Borrowing and Returning: Facilitates the borrowing and returning of library materials by users, including checking availability and managing due dates.

User Management: Allows librarians to create and manage user accounts, update user information, and handle user-related operations.

2) **Cataloging and Inventory Management**:

This module is responsible for creating and maintaining a catalog of library resources, including books, journals, multimedia, and other materials. Each item is assigned a unique identifier and relevant information, such as title, author, subject, and publication details.

Item Entry: Provides a feature to enter new books or other materials into the library catalog, including information such as title, author, publisher, ISBN, and subject.

Classification and Categorization: Enables librarians to assign appropriate categories, subjects, and keywords to each item for easier searching and browsing.

Barcode Generation: Generates unique barcodes for each item to streamline circulation and inventory management processes.

Inventory Tracking: Tracks the location, availability, and status of library materials, including information on borrowed items, reserved items, and their due dates.

3) **Circulation Management:**

The circulation module manages the borrowing and returning of library items. It tracks the availability of items, handles due dates, and generates reminders or penalties for late returns.

Check Out/In: Allows librarians to check out items to borrowers and record due dates, while also facilitating the return process and managing late fees, if applicable.

Reservation and Holds: Enables users to reserve items that are currently unavailable and notifies them when the items become available for borrowing.

Renewals: Provides the option for borrowers to renew their borrowed items for an extended loan period, if allowed by library policies.

Overdue Management: Sends notifications and generates reports for overdue items, managing fines, and handling any necessary follow-up actions.

4) **Search and Discovery:**

Users can search the library's collection using keywords, author names, or subject categories.

Search functionality: The system should provide a powerful search function that allows users to search for resources based on various criteria, such as title, author, subject, keywords, or a combination of these.

Browsing and recommendations: The system should include browsing features that allow users to explore the library's collection by categories, genres, or other predefined criteria. Additionally, personalized recommendations can be generated based on the user's borrowing history, reading preferences, or similar users' behavior, suggesting relevant resources they may be interested in.

User-friendly interface: The search and discovery interface should be intuitive and user-friendly, enabling users to easily navigate and interact with the library management system. It gives organized presentation of search results with detailed resource information.

4) **Reporting and Analytics:**

The system generates various reports and statistics to help librarians analyze library usage, track overdue items, and make informed decisions for resource allocation.

Transaction Reports: Generates reports on circulation activities, including checkouts, returns, and renewals, to help analyze library usage and trends.

Financial Reports: Generates reports on fines collected, payments, and other financial aspects to assist in managing library finances.

Benefits:

**PROBLEM STATEMENT: LIBRARY MANAGEMENT SYSTEM**

Suppose you have a class called Library that contains a list of books. Each book has a title, author, and number of pages. Implement the Library class with the following methods:

* add\_book(title, author, num\_pages): adds a new book to the library with the given title, author, and number of pages.
* remove\_book(title): removes a book from the library with the given title.
* get\_books\_by\_author(author): returns a list of books by the given author.
* get\_total\_pages(): returns the total number of pages in the library.
* Using the Library class, perform the following operations:
* Add at least 5 books to the library.
* Use map to create a new list of books that contains only the titles of the books.
* Use filter to create a new list of books that contain more than 300 pages.
* Use reduce to calculate the total number of pages in the library.
* Use the get\_books\_by\_author method to get a list of books by a specific author

**AI Implementation**

The provided program is a simple implementation of a Library Management System. While the program itself does not directly demonstrate the use of AI, it showcases how automation and intelligent decision-making can be incorporated into a system. Here's how the program can be related to the use of AI:

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.

4.Recommendation System: Implement a recommendation system that suggests accounts to users based on their preferences, reading history, or other factors.

5.Text Analysis: Utilize text analysis techniques to extract insights from account details, or descriptions, such as sentiment analysis or topic modeling.

**PROGRAM FOR LIBRARY MANAGEMENT SYSTEM**

print(" LIBRARY MANAGEMENT SYSTEM ")

from functools import reduce

print("")

book\_list=[ ]

class Library:

def add\_book(self):

b=[ ]

title=input("Enter book title name :")

author=input("Enter book author name :")

num\_page=int(input("Enter number of pages in book :"))

b.append(title)

b.append(author)

b.append(num\_page)

book\_list.append(b)

print("--------------------------------------------")

def get\_books\_by\_author(self):

author\_name=input("Enter name of author :")

p=0

for i in book\_list:

if author\_name in i:

print(f"{i[0]} book is written by {author\_name}")

p=1

if p==0:

print("book with this author name is not present")

print("-------------------------------------------")

def total\_pages(self):

num\_list=[]

for j in book\_list:

num\_list.append(j[2])

def addition(x,y):

return x+y

total=reduce(addition,num\_list)

print("total number of pages in library : ", total)

print("------------------------------------------")

def table(self):

print("book\_names author\_names pages")

print("\_\_\_\_\_\_\_\_\_\_\_\_")

for i in book\_list:

print(f"{i[0]} | {i[1]} | {i[2]}")

print("\_\_\_\_\_\_\_\_\_")

lib=Library()

while True:

# Displaying the option menu

print("Enter 1 to add new book")

if len(book\_list) >= 1:

print("Enter 2 to display the book name")

if len(book\_list)>= 1:

print("Enter 3 to display the total number of pages in library")

if len(book\_list)>= 1:

print("Enter 4 to display the table of books")

inp = input("Enter your choice :- ")

if int(inp) == 1:

lib.add\_book()

elif int(inp)==2 and len(book\_list) >= 1:

lib.get\_books\_by\_author()

elif int(inp)==3 and len(book\_list)>= 1:

lib.total\_pages()

elif int(inp)==4 and len(book\_list)>= 1:

lib.table()

else:

print("Invalid Choice, Exiting from the program")

break

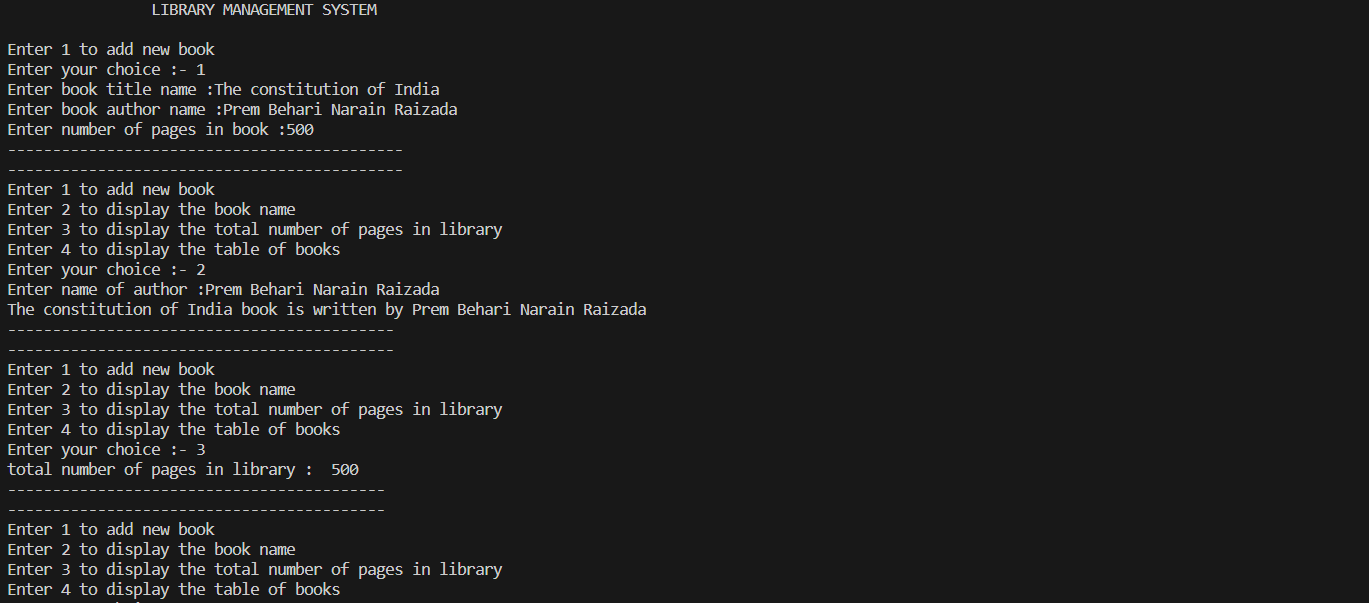
**Explanation of the code:**

This program is a basic implementation of a Library Management System. It allows users to perform various operations related to managing a library. Here's a brief explanation of the program:

1. The program starts by printing the title and a separator line.
2. The **‘book\_list’** variable is initialized as an empty list to store book information.
3. The **‘Library’** class is defined, which contains methods to perform different operations.
4. The **‘add\_book’** method allows the user to add a new book by providing the title, author name, and number of pages. The book details are stored as a list and appended to the **‘book\_list’**.
5. The **‘get\_books\_by\_author’** method prompts the user to enter an author's name and then searches the **‘book\_list’** for books written by that author. It prints the book titles along with the author's name.
6. The **‘total\_pages’** method calculates and prints the total number of pages in the library by summing up the number of pages for each book in the **‘book\_list’**.
7. The **‘table’** method displays a table of all the books in the book\_list, showing the **‘book title’**, author name, and number of pages.
8. An instance of the **‘Library’** class,**’lib’**, is created.
9. The program enters a while loop to continuously display the option menu and perform operations based on user input.
10. If the **‘book\_list’** is not empty, the user can choose options 2, 3, and 4 to display books by author, total number of pages, and the table of books, respectively.
11. The user can choose option 1 to add a new book by calling the **‘add\_book’** method.
12. If the user enters an invalid choice or selects an option without any books in the library, the program terminates.

Overall, this program provides basic functionality for adding books, searching books by author, calculating total pages, and displaying book information in a table format.

**Output of LIBRARY MANAGEMENT code:**

****