**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.

**UseCase-1**

**Bank Account Management System**

**Description:**

The bank account management system allows customers to perform deposit and withdrawal operations on their bank accounts. It provides an interface for users to interact with their accounts and maintains a list of bank account numbers

**Actors:**

- User

- Bank

**Preconditions:**

- The Bank Account Management System must be operational.

**Flow of Events:**

1. User launches the Bank Account Management System.

2. User is prompted to enter their bank account number.

3. User enters their bank account number.

4. The system creates an instance of the BankAccount class with the provided account number and initializes the account balance to 0.

5. The system adds the account number to the list of bank accounts.

6. The system displays a success message indicating that the account has been created.

7. The system presents the user with the available actions:

- Deposit

- Withdraw

8. User selects an action by entering the corresponding choice:

- If the user selects "Deposit":

- The system prompts the user to enter the deposit amount.

- User enters the amount to deposit.

- The system calls the `deposit()` method of the BankAccount instance, passing the deposit amount.

- The system updates the account balance accordingly.

- The system displays a success message with the updated balance.

- If the user selects "Withdraw":

- The system prompts the user to enter the withdrawal amount.

- User enters the amount to withdraw.

- The system calls the `withdraw()` method of the BankAccount instance, passing the withdrawal amount.

- The system checks if the account has sufficient funds for the withdrawal:

- If the account balance is greater than or equal to the withdrawal amount:

- The system deducts the withdrawal amount from the account balance.

- The system displays a success message with the updated balance.

- If the account balance is less than the withdrawal amount:

- The system displays an error message indicating insufficient funds.

9. The system retrieves the current balance of the account using the `get\_balance()` method.

10. The system displays the account number and the current balance.

11. The system terminates.

**Postconditions:**

- The user can create a bank account and perform deposit or withdrawal operations on that account.

- The Bank Account Management System maintains a list of bank account numbers for reference.

- The system provides the user with the current balance of the account.

This use case outlines the basic flow of events for the Bank Account Management System. It covers the process of creating a bank account, performing deposit and withdrawal operations, and retrieving the account balance. The system also maintains a list of bank account numbers for reference. Additional features like account balance inquiries, transaction history, and user authentication could be added to enhance the functionality of the system.

**Problem Statement for:** **Bank Account Management System**

You are tasked with implementing a Bank Account Management System using the Python programming language. The system should allow users to perform various operations on their bank accounts, including depositing and withdrawing funds, as well as checking the current balance.

**Functional Requirements:**

1. Create BankAccount class:

* The BankAccount class should have the following attributes:
* account\_number (integer): A unique identifier for the bank account.
* balance (float): The current balance in the account.
* The BankAccount class should have the following methods:
* \_\_init\_\_(self, account\_number): Initializes a new bank account with the given account number and a balance of 0.
* deposit(self, amount): Deposits the specified amount into the account and updates the balance accordingly.
* withdraw(self, amount): Withdraws the specified amount from the account, if the account has sufficient funds, and updates the balance accordingly.
* get\_balance(self): Returns the current balance in the account.

1. Create an empty list bankaccount to store bank account numbers.
2. Prompt the user to enter their account number and store it in the account\_number variable.
3. Create an instance of the BankAccount class with the provided account number.
4. Add the account number to the bankaccount list.
5. Present the user with the following options:

* Deposit: Prompt the user to enter the amount to deposit. Call the deposit() method of the BankAccount instance with the deposit amount.
* Withdraw: Prompt the user to enter the amount to withdraw. Call the withdraw() method of the BankAccount instance with the withdrawal amount.
* Cancel: Display a message indicating that the transaction has been canceled.
* Invalid choice: Display a message indicating that the user has made an invalid choice.

1. Display the account number and the current balance using the get\_balance() method.

**Non-functional Requirements:**

* The account number should be an integer.
* The balance should be a float and should not be allowed to go below 0.
* The program should handle invalid inputs, such as non-numeric values or negative amounts, and provide appropriate error messages to the user.
* The program should gracefully handle exceptions and prevent any unexpected crashes.
* The program should be easy to understand, with clear variable and method names and appropriate comments.

**AI Implementation**

AI, which stands for artificial intelligence, involves the use of algorithms and models to mimic human intelligence and perform tasks that require human-like understanding or decision-making. Without incorporating any AI-specific functionality or components, your code does not have an AI element.

1. Natural Language Processing (NLP): Use NLP techniques to enhance search capabilities, allowing users to search for books using natural language queries.
2. Recommendation System: Implement a recommendation system that suggests books to users based on their preferences, reading history, or other factors.
3. Text Analysis: Utilize text analysis techniques to extract insights from book titles, authors, or descriptions, such as sentiment analysis or topic modeling.
4. First it ask for user to enter the numbers how many accounts user want to create, It create automatically the account numbers based on user input.
5. It will react to user inputs

**PROGRAM FOR Bank Account Management**

print()

class BankAccount :

# account\_number=int()

# balance=float(0.0)

def \_\_init\_\_(self,account\_number):

self.account\_number =account\_number

self.balance=0.0

def deposit(self,amount):

self.balance +=amount

if self.deposit:

print(f"Amount deposited successfully to - {self.account\_number}",f"balance : {self.balance}")

else:

print("You have canceld Thank You!")

def withdraw(self,amount):

if self.balance >= amount:

self.balance -= amount

print(f"Amount withdraw successfully from -{self.account\_number} ")

else :

print(f"Insaficient found in Account: {self.account\_number}")

def get\_balance(self):

return self.balance

num = int(input("Enter account numbers: "))

account\_numbers=1000

bankaccount = []

for i in range(num):

accNum=account\_numbers + (i+1)

num1=BankAccount(accNum)

bankaccount.append(num1)

print(f" \* Account created successfully:{accNum}")

print()

opt = input("Enter choice (1-deposit, 2-withdraw,3-check balance 9-cancel): ")

if opt == "1":

deposit\_amount = float(input("Enter the amount to deposit: "))

num1.deposit(deposit\_amount)

elif opt == "2":

withdrawal\_amount = int(input("Enter the amount to withdraw: "))

num1.withdraw(withdrawal\_amount)

elif opt == "3":

print(f"Your balance is : {num1.get\_balance()}")

elif opt == "9":

print("Transaction canceled")

else:

print("Invalid choice")

break

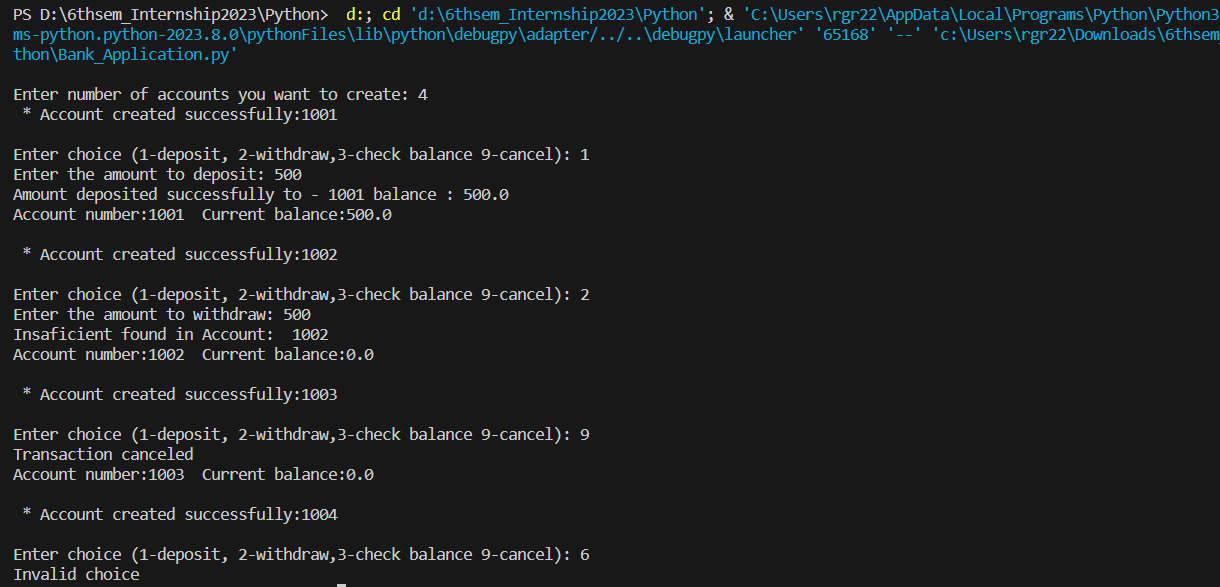
print(f"Accountnumber:{num1.account\_number} ",f"Current balance:{num1.get\_balance()}")

print()

**Explanation of code:**

1. The BankAccount class is defined, which will represent a bank account. It has two attributes, account\_number and balance. The account\_number is initialized with the value passed to the constructor, and the balance is set to 0.0 by default.
2. The \_\_init\_\_ method is the constructor of the BankAccount class. It takes an account\_number parameter and initializes the account\_number and balance attributes accordingly.
3. The deposit method is used to deposit an amount into the account. It takes an amount parameter, adds the amount to the current balance, and prints a success message along with the updated balance.
4. The withdraw method is used to withdraw an amount from the account. It takes an amount parameter, checks if the account has sufficient balance, subtracts the amount from the balance if possible, and prints a success message. If the balance is insufficient, it prints a corresponding message.
5. The get\_balance method returns the current balance of the account
6. The code prompts the user to enter the number of accounts they want to create (num). It also initializes the account\_numbers variable with a starting value of 1000.
7. A loop is executed num times to create num bank accounts. Inside the loop, an account number is generated by adding the current value of account\_numbers with the loop index (i+1). An instance of BankAccount is created with the generated account number, and it is appended to the bankaccount list.
8. After creating an account, the user is prompted to enter their choice: 1 for deposit, 2 for withdrawal, 3 for checking the balance, or 9 to cancel the transaction.
9. Depending on the user's choice, the corresponding action is performed on the num1 account (the current account being processed in the loop). If the choice is 1, the user is prompted to enter the amount to deposit, and the deposit method of the account is called. If the choice is 2, the user is prompted to enter the amount to withdraw, and the withdraw method of the account is called. If the choice is 3, the get\_balance method is called and the current balance is printed. If the choice is 9, a transaction canceled message is printed.
10. Finally, after performing the user's selected action, the account number and current balance of the num1 account are printed.

**Output for Bank Account Management Code:**

****