ASSIGNMENT-6.1

Name:N.Soumya

H NO:2303A51869

Batch-13

Task Description #1 (AI-Based Code Completion for Loops)

Task: Use an AI code completion tool to generate a loop-based Program.

Prompt:

“Generate Python code to print all even numbers between 1 and N using a loop.”

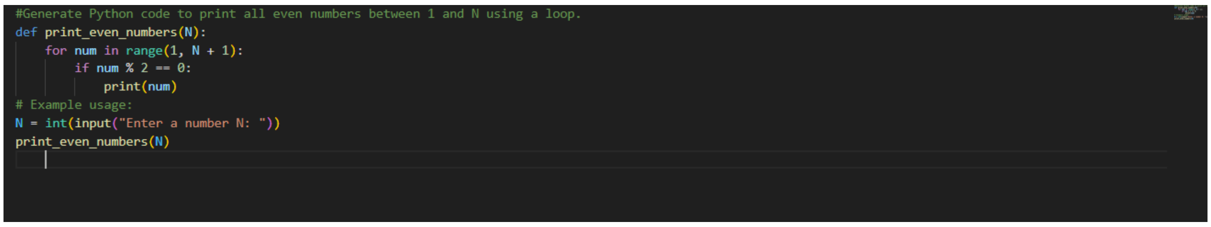
Expected Output:

• AI-generated loop logic.

• Identification of loop type used (for or while).

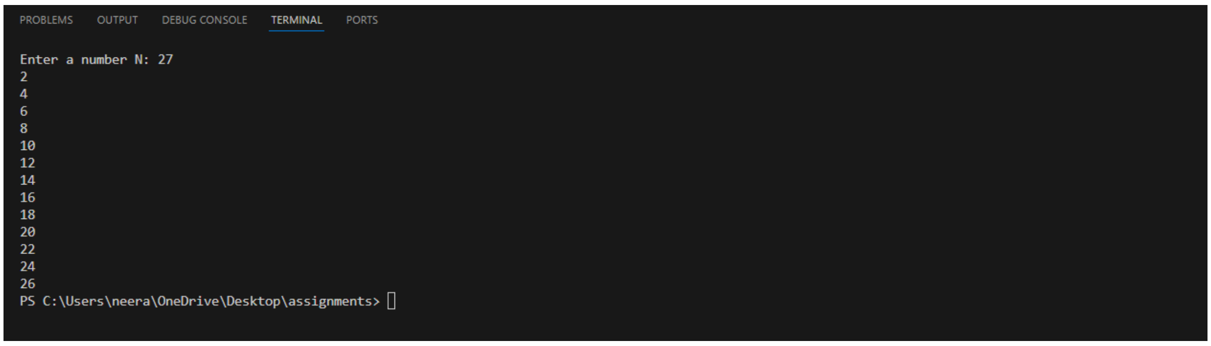
• Validation with sample inputs.

CODE:



```
#Generate Python code to print all even numbers between 1 and N using a loop.
def print_even_numbers(N):
    for num in range(1, N + 1):
        if num % 2 == 0:
            print(num)
# Example usage:
N = int(input("Enter a number N: "))
print_even_numbers(N)
```

OUTPUT:



```
Enter a number N: 27
2
4
6
8
10
12
14
16
18
20
22
24
26
PS C:\Users\neera\OneDrive\Desktop\assignments>
```

JUSTIFICATION:

This program uses a for loop to iterate through all numbers from 1 to N and applies a conditional statement to check whether each number is even. The modulo operator (%) helps determine divisibility by 2, ensuring only even numbers are printed. Taking N as user input makes the program flexible for different ranges. The logic is simple, efficient, and easy to understand. Overall, it clearly demonstrates the combined use of loops and conditionals in Python.

Task Description #2 (AI-Based Code Completion for Loop with Conditionals)

Task: Use an AI code completion tool to combine loops and conditionals.

Prompt:

“Generate Python code to count how many numbers in a list are even and odd.”

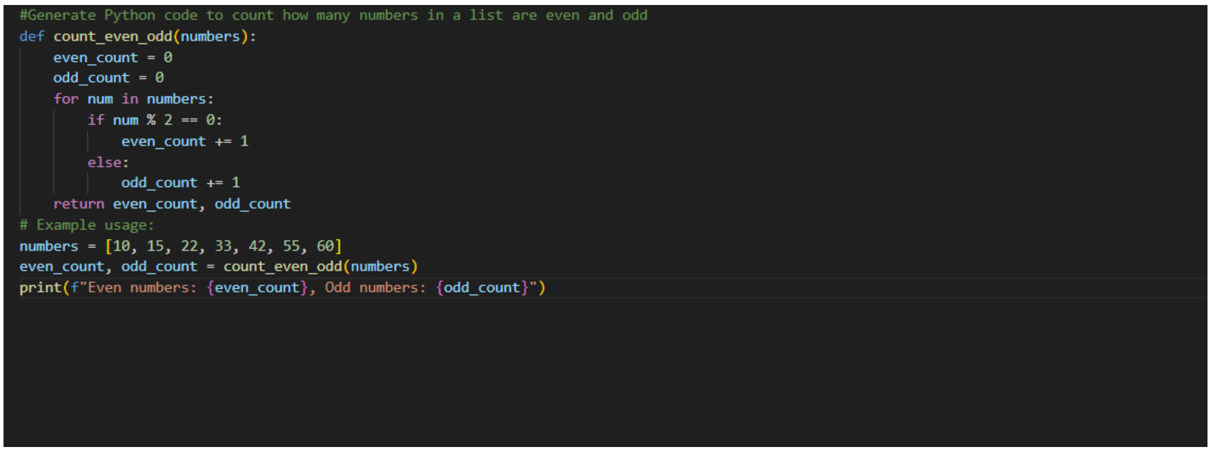
Expected Output:

• AI-generated code using loop and if condition.

• Correct count validation.

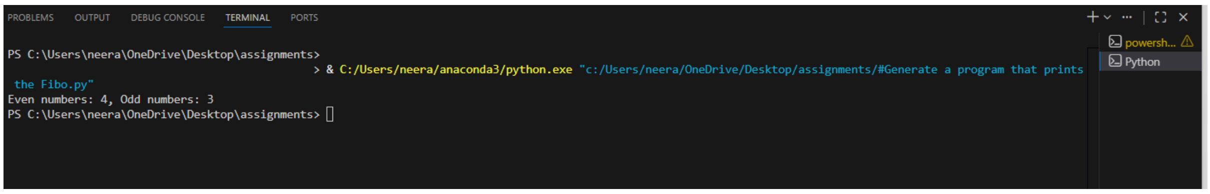
• Explanation of logic flow.

CODE:



```python
#Generate Python code to count how many numbers in a list are even and odd
def count_even_odd(numbers):
    even_count = 0
    odd_count = 0
    for num in numbers:
        if num % 2 == 0:
            even_count += 1
        else:
            odd_count += 1
    return even_count, odd_count
# Example usage:
numbers = [10, 15, 22, 33, 42, 55, 60]
even_count, odd_count = count_even_odd(numbers)
print(f"Even numbers: {even_count}, Odd numbers: {odd_count}")
```

OUTPUT:



```
PS C:\Users\neera\OneDrive\Desktop\assignments>
                                                > & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Generate a program that prints
 the Fibo.py"
Even numbers: 4, Odd numbers: 3
PS C:\Users\neera\OneDrive\Desktop\assignments> 
```

JUSTIFICATION:

The program begins by creating a User class that stores user details such as name, age, and email. The validate_age() method uses a conditional statement to verify whether the user’s age meets the minimum requirement of 18 years. Similarly, the validate_email() method applies conditions to check if the email contains valid characters like @ and . The validate_user() method then calls both validation functions and displays the results together. Overall, conditional statements play a key role in ensuring that user details are properly validated before acceptance.

Task Description #3 (AI-Based Code Completion for Class Attributes Validation)

Task: Use an AI tool to complete a Python class that validates user input.

Prompt:

“Generate a Python class User that validates age and email using conditional statements.”

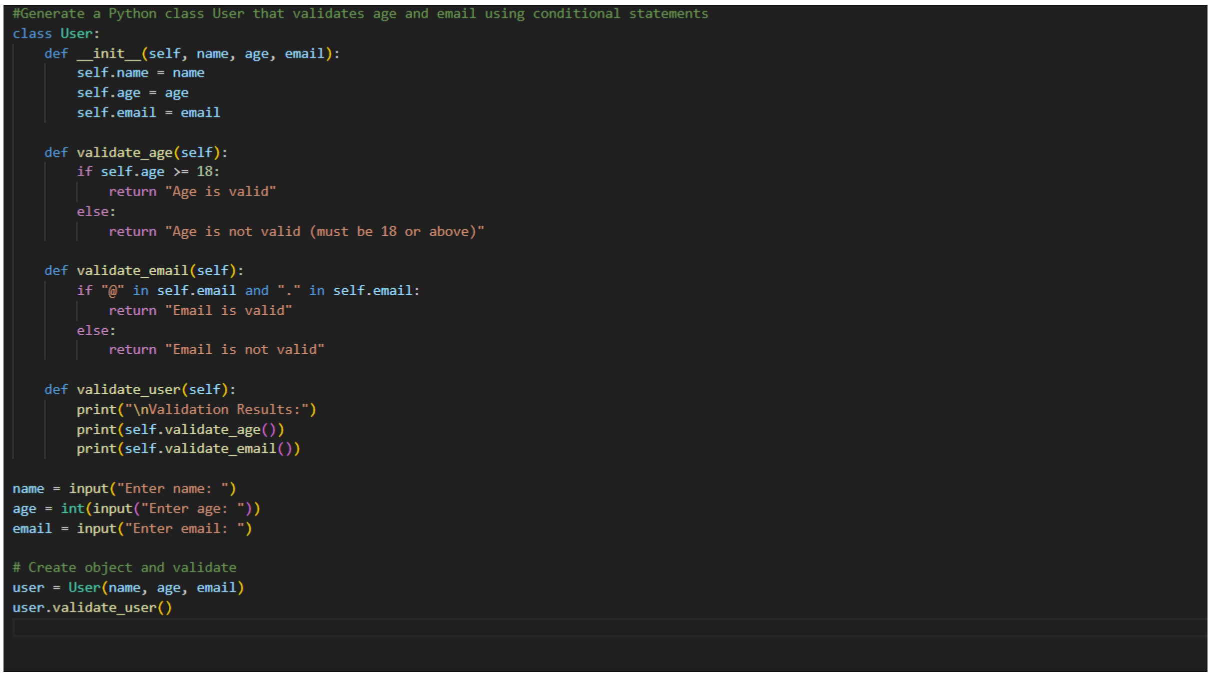
Expected Output:

• AI-generated class with validation logic.

• Verification of condition handling.

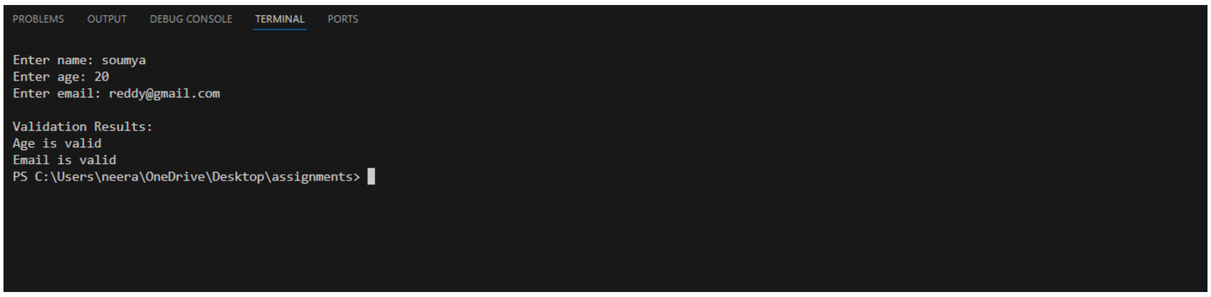
• Test cases for valid and invalid inputs.

CODE:

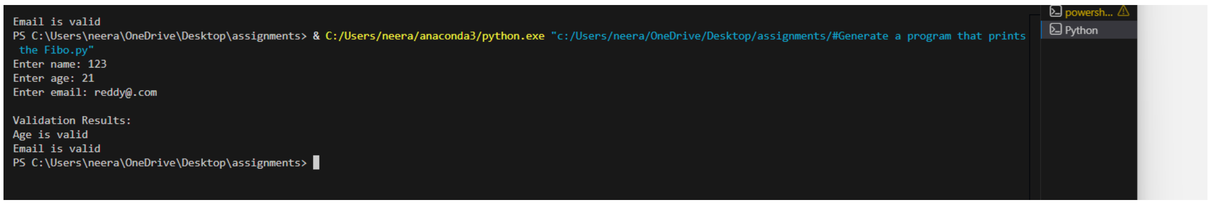


```python
#Generate a Python class User that validates age and email using conditional statements
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email

    def validate_age(self):
        if self.age >= 18:
            return "Age is valid"
        else:
            return "Age is not valid (must be 18 or above)"

    def validate_email(self):
        if "@" in self.email and "." in self.email:
            return "Email is valid"
        else:
            return "Email is not valid"

    def validate_user(self):
        print("\nValidation Results:")
        print(self.validate_age())
        print(self.validate_email())

name = input("Enter name: ")
age = int(input("Enter age: "))
email = input("Enter email: ")

# Create object and validate
user = User(name, age, email)
user.validate_user()
```

OUTPUT:



```
Enter name: soumya
Enter age: 20
Enter email: reddy@gmail.com

Validation Results:
Age is valid
Email is valid
PS C:\Users\neera\OneDrive\Desktop\assignments>
```



```
Email is valid
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Generate a program that prints
 the Fibo.py"
Enter name: 123
Enter age: 21
Enter email: reddy@.com

Validation Results:
Age is valid
Email is valid
PS C:\Users\neera\OneDrive\Desktop\assignments>
```

JUSTIFICATION:

This program justifies the use of object-oriented programming by encapsulating user data and validation logic within a class. Conditional statements are effectively used to validate age and email based on defined rules. Taking user input dynamically makes the program interactive and practical. AI-based code completion helps generate structured and error-free code efficiently. Overall, the task demonstrates the integration of classes, loops, and conditionals in Python.

Task Description #4 (AI-Based Code Completion for Classes)

Task: Use an AI code completion tool to generate a Python class for

managing student details.

Prompt:

“Generate a Python class Student with attributes (name, roll number,

marks) and methods to calculate total and average marks.”

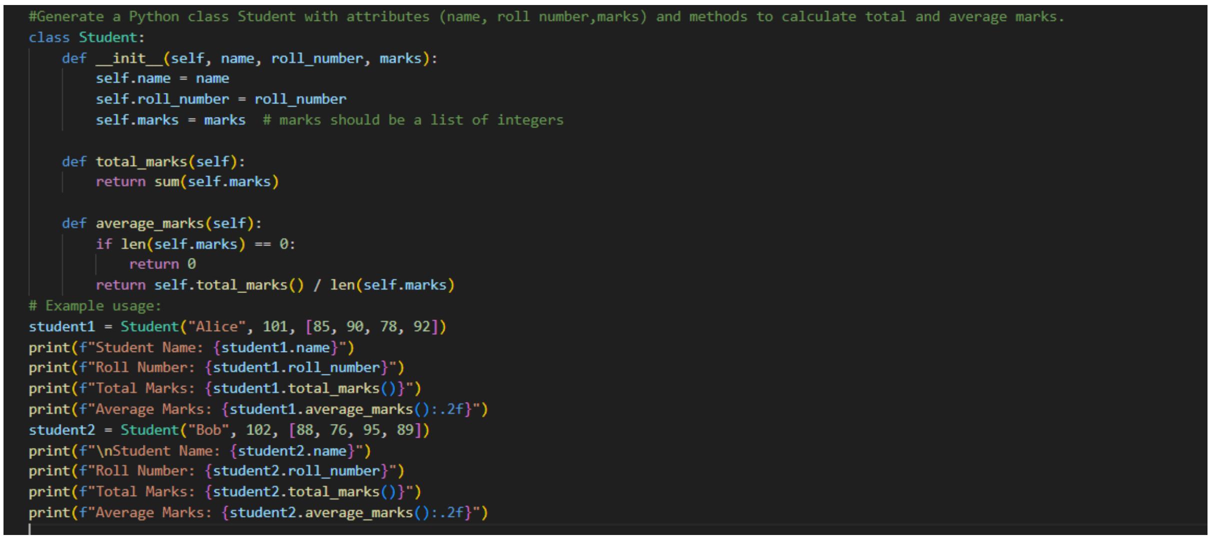
Expected Output:

• AI-generated class code.

• Verification of correctness and completeness of class structure.

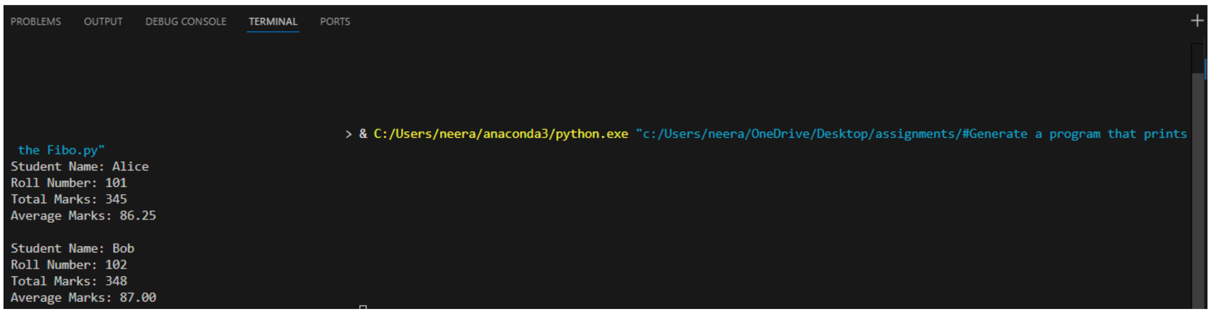
• Minor manual improvements (if needed) with justification.

CODE:



```python
#Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total and average marks.
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks  # marks should be a list of integers

    def total_marks(self):
        return sum(self.marks)

    def average_marks(self):
        if len(self.marks) == 0:
            return 0
        return self.total_marks() / len(self.marks)
# Example usage:
student1 = Student("Alice", 101, [85, 90, 78, 92])
print(f"Student Name: {student1.name}")
print(f"Roll Number: {student1.roll_number}")
print(f"Total Marks: {student1.total_marks()}")
print(f"Average Marks: {student1.average_marks():.2f}")
student2 = Student("Bob", 102, [88, 76, 95, 89])
print(f"\nStudent Name: {student2.name}")
print(f"Roll Number: {student2.roll_number}")
print(f"Total Marks: {student2.total_marks()}")
print(f"Average Marks: {student2.average_marks():.2f}")
```

OUTPUT:



```
                                   > & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Generate a program that prints
 the Fibo.py"
Student Name: Alice
Roll Number: 101
Total Marks: 345
Average Marks: 86.25

Student Name: Bob
Roll Number: 102
Total Marks: 348
Average Marks: 87.00
```

JUSTIFICATION:

This program demonstrates the use of object-oriented programming by defining a Student class that groups student data and related operations together. The methods total_marks() and average_marks() use built-in functions and conditional checks to compute results accurately. Handling an empty marks list prevents runtime errors and ensures robustness. Creating multiple student objects shows reusability of the class design. Overall, the code clearly illustrates how classes and methods simplify data management and calculations in Python.

Task Description 5 (AI-Assisted Code Completion Review)

Task: Use an AI tool to generate a complete Python program using

classes, loops, and conditionals together.

Prompt:

“Generate a Python program for a simple bank account system using

class, loops, and conditional statements.”

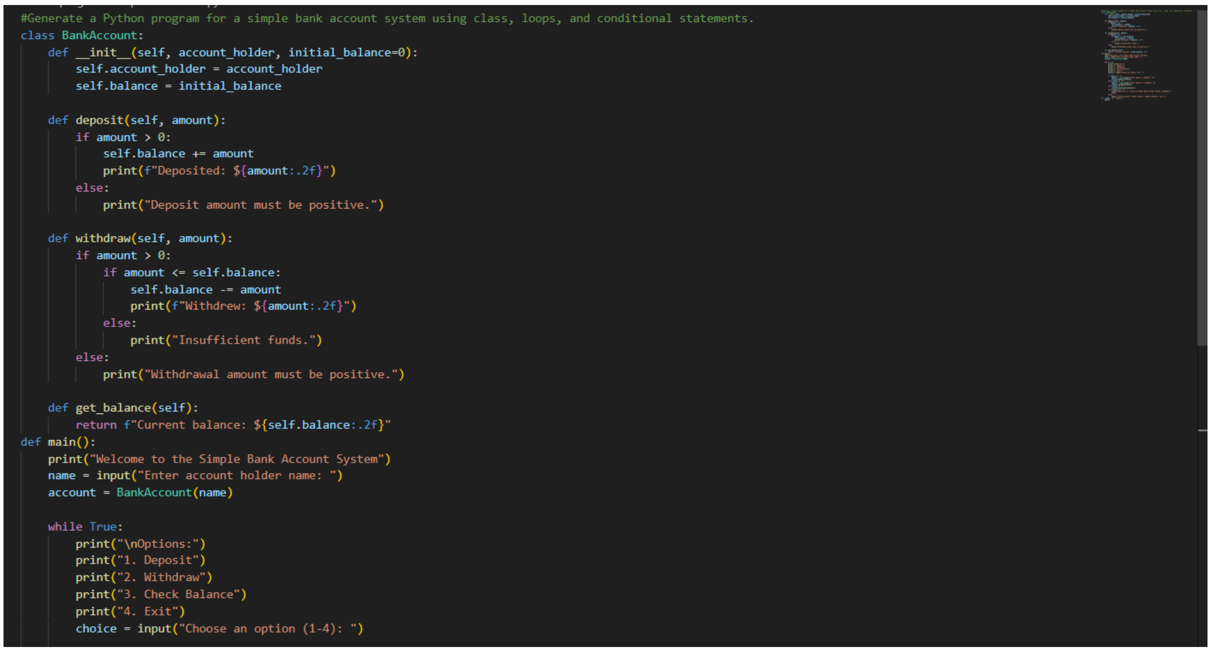
Expected Output:

• Complete AI-generated program.

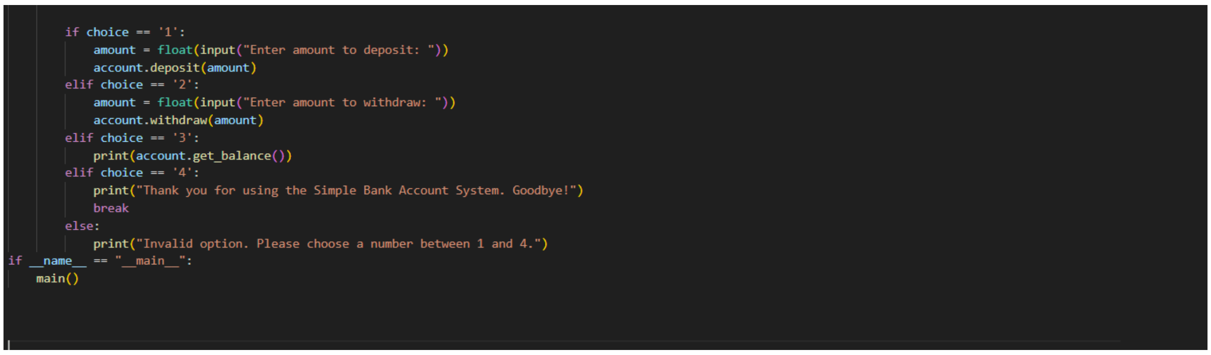
• Identification of strengths and limitations of AI suggestions.

• Reflection on how AI assisted coding productivity.

CODE:

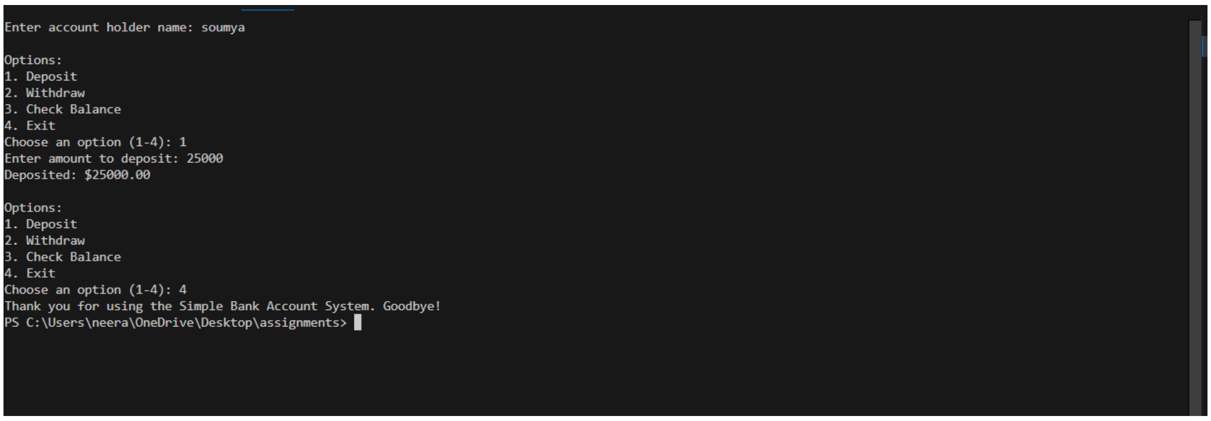


```python
#Generate a Python program for a simple bank account system using class, loops, and conditional statements.
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: ${amount:.2f}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def get_balance(self):
        return f"Current balance: ${self.balance:.2f}"
def main():
    print("Welcome to the Simple Bank Account System")
    name = input("Enter account holder name: ")
    account = BankAccount(name)

    while True:
        print("\nOptions:")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Exit")
        choice = input("Choose an option (1-4): ")
```



```python
        if choice == '1':
            amount = float(input("Enter amount to deposit: "))
            account.deposit(amount)
        elif choice == '2':
            amount = float(input("Enter amount to withdraw: "))
            account.withdraw(amount)
        elif choice == '3':
            print(account.get_balance())
        elif choice == '4':
            print("Thank you for using the Simple Bank Account System. Goodbye!")
            break
        else:
            print("Invalid option. Please choose a number between 1 and 4.")
if __name__ == "__main__":
    main()
```

OUTPUT:



```
Enter account holder name: soumya

Options:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 1
Enter amount to deposit: 25000
Deposited: $25000.00

Options:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 4
Thank you for using the Simple Bank Account System. Goodbye!
PS C:\Users\neera\OneDrive\Desktop\assignments>
```

JUSTIFICATION:

This program justifies the use of object-oriented programming by modeling a real-world bank account using a BankAccount class. Conditional statements ensure valid deposits, withdrawals, and prevent overdrafts, maintaining data integrity. The while loop enables continuous user interaction through a menu-driven system. Separating operations into methods improves code readability and reusability. Overall, the program effectively integrates classes, loops, and conditionals to build a simple yet functional banking system.