Assignment-6.5

Name: Adepu Tejaswini

Ht.no: 2303A51423

Batch. No: 21

Task Description #1 (AI-Based Code Completion for Conditional Eligibility Check)

Task: Use an AI tool to generate eligibility logic.

Prompt:

“Generate Python code to check voting eligibility based on age and citizenship.”

Expected Output:

• AI-generated conditional logic.

• Correct eligibility decisions.

• Explanation of conditions.

Prompt:

Generate Python code to check voting eligibility based on a person’s age and citizenship status using a user-defined function.

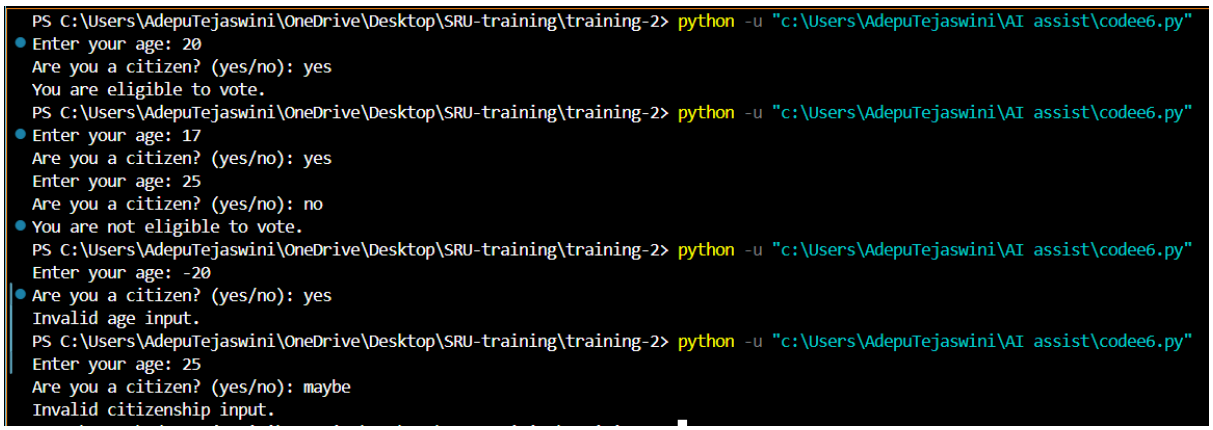
The code should take age and citizenship as input, to determine eligibility, and display whether the person is eligible to vote or not.

If the input is invalid like negative age or missing citizenship status, the code should handle these cases appropriately.

Code:

```python
def check_voting_eligibility(age, citizenship):
    if age < 0:
        return "Invalid age input."
    if citizenship not in ["yes", "no"]:
        return "Invalid citizenship input."
    if age >= 18 and citizenship == "yes":
        return "You are eligible to vote."
    else:
        return "You are not eligible to vote."
# Taking user input
try:
    age = int(input("Enter your age: "))
    citizenship = input("Are you a citizen? (yes/no): ").strip().lower()
    result = check_voting_eligibility(age, citizenship)
    print(result)
except ValueError:
    print("Please enter a valid number for age.")
```

Output:



Explanation:

This program uses a user-defined function to check voting eligibility based on age and citizenship.
Conditional statements ensure the age is non-negative, citizenship input is valid, and the person is at least 18 years old and a citizen.
Invalid inputs such as negative age, incorrect citizenship values, or non-numeric age are handled using condition checks and exception handling.

Task Description #2(AI-Based Code Completion for Loop-Based String Processing)

Task: Use an AI tool to process strings using loops.

Prompt:

“Generate Python code to count vowels and consonants in a string using a loop.”

Expected Output:

• AI-generated string processing logic.

• Correct counts.

• Output verification.

Prompt:

Generate a Python program to count the number of vowels and consonants in a given string using loops.

The program should use a user-defined function to perform the counting.

It should accept a string input from the user and iterate through each character using a loop.

Handle invalid inputs such as empty strings or non-string values by displaying appropriate error messages.

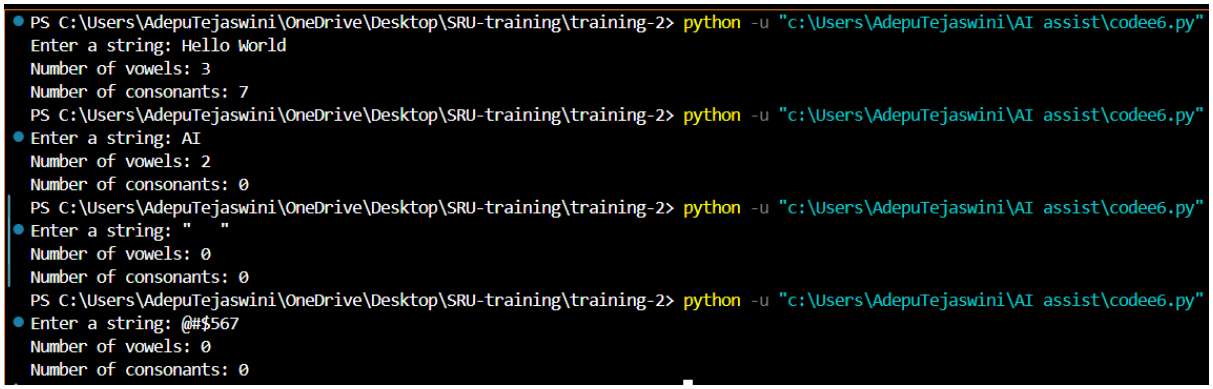
Do not use built-in vowel counting methods.

Print and return the correct count of vowels and consonants, and include comments explaining the logic.

Code:

```python
def count_vowels_consonants(input_string):
    # Initialize counts
    vowel_count = 0
    consonant_count = 0

    # Define vowels
    vowels = "aeiouAEIOU"

    # Check if the input string is empty
    if not input_string:
        return "Invalid input: The string is empty."

    # Iterate through each character in the string
    for char in input_string:
        # Check if the character is an alphabet
        if char.isalpha():
            # Check if the character is a vowel
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return vowel_count, consonant_count
# Taking user input
user_input = input("Enter a string: ")
result = count_vowels_consonants(user_input)
if isinstance(result, str):
    print(result)
else:
    vowels, consonants = result
    print(f"Number of vowels: {vowels}")
    print(f"Number of consonants: {consonants}")
```

Output:



Explanation:

This program uses a user-defined function to count vowels and consonants by iterating through each character using a loop.
It checks only alphabetic characters, classifying them as vowels or consonants while ignoring numbers and special symbols.
An empty string is handled as invalid input, ensuring the program produces correct and reliable output.

Task Description #3 (AI-Assisted Code Completion Reflection Task)

Task: Use an AI tool to generate a complete program using classes, loops, and conditionals.

Prompt:

“Generate a Python program for a library management system using classes, loops, and conditional statements.”

Expected Output:

• Complete AI-generated program.

• Review of AI suggestions quality.

• Short reflection on AI-assisted coding experience.

Prompt:

Generate a complete Python program for a library management system using classes.

The program should use loops and conditional statements for menu-driven operations.

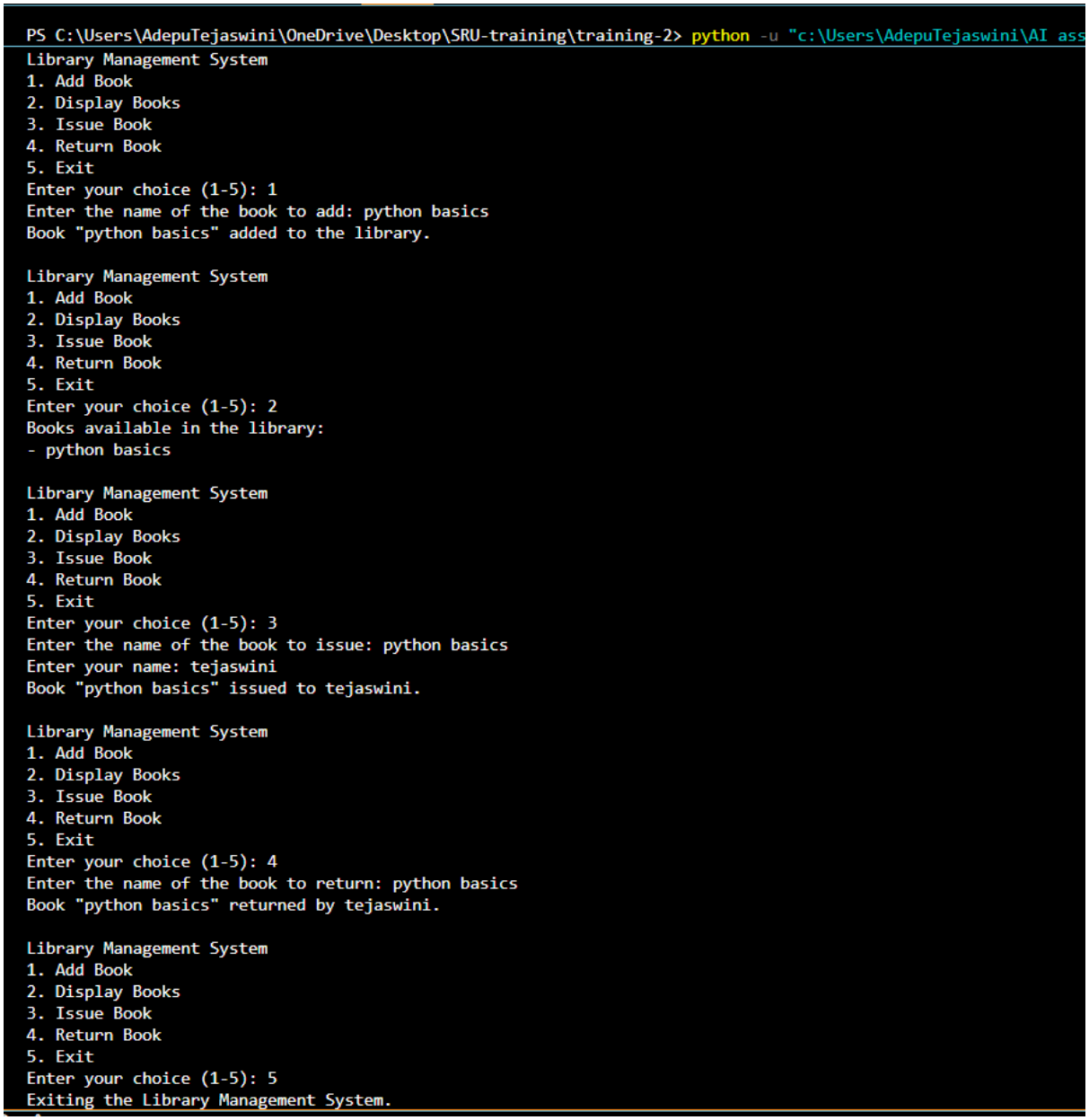
Include features to add books, display books, issue books, and return books.

Ensure proper input handling and include comments explaining the logic.

Code:

```python
class Library:
    def __init__(self):
        # Initialize an empty list to store books
        self.books = []
        self.issued_books = {}

    def add_book(self, book_name):
        # Add a book to the library
        self.books.append(book_name)
        print(f'Book "{book_name}" added to the library.')

    def display_books(self):
        # Display all books in the library
        if not self.books:
            print("No books available in the library.")
        else:
            print("Books available in the library:")
            for book in self.books:
                print(f"- {book}")

    def issue_book(self, book_name, user_name):
        # Issue a book to a user
        if book_name in self.books:
            self.books.remove(book_name)
            self.issued_books[book_name] = user_name
            print(f'Book "{book_name}" issued to {user_name}.')
        else:
            print(f'Book "{book_name}" is not available in the library.')

    def return_book(self, book_name):
        # Return a book to the library
        if book_name in self.issued_books:
            user_name = self.issued_books.pop(book_name)
            self.books.append(book_name)
            print(f'Book "{book_name}" returned by {user_name}.')
        else:
            print(f'Book "{book_name}" was not issued from this library.')
def main():
    library = Library()
    while True:
        print("\nLibrary Management System")
        print("1. Add Book")
        print("2. Display Books")
        print("3. Issue Book")
        print("4. Return Book")
        print("5. Exit")
        choice = input("Enter your choice (1-5): ")

        if choice == '1':
            book_name = input("Enter the name of the book to add: ")
            library.add_book(book_name)
        elif choice == '2':
            library.display_books()
        elif choice == '3':
            book_name = input("Enter the name of the book to issue: ")
            user_name = input("Enter your name: ")
            library.issue_book(book_name, user_name)
        elif choice == '4':
            book_name = input("Enter the name of the book to return: ")
            library.return_book(book_name)
        elif choice == '5':
            print("Exiting the Library Management System.")
            break
        else:
            print("Invalid choice. Please try again.")
if __name__ == "__main__":
    main()
```

Output:



Explanation

This program uses a Library class to manage books and issued records, applying object-oriented principles.
A loop-driven menu allows users to add, display, issue, and return books, with conditional statements controlling the flow.
Proper input handling and clear messages ensure correct operations and a smooth user experience.

Task Description #4 (AI-Assisted Code Completion for Class-Based Attendance System)

Task: Use an AI tool to generate an attendance management class.

Prompt: “Generate a Python class to mark and display student attendance using loops.”

Expected Output:

• AI-generated attendance logic.

• Correct display of attendance.

• Test cases.

Prompt:

Generate a complete Python program for an Attendance Management System using a class.

The class should store student names and their attendance status.

Use loops to add students, mark them as present or absent, and display the attendance list.

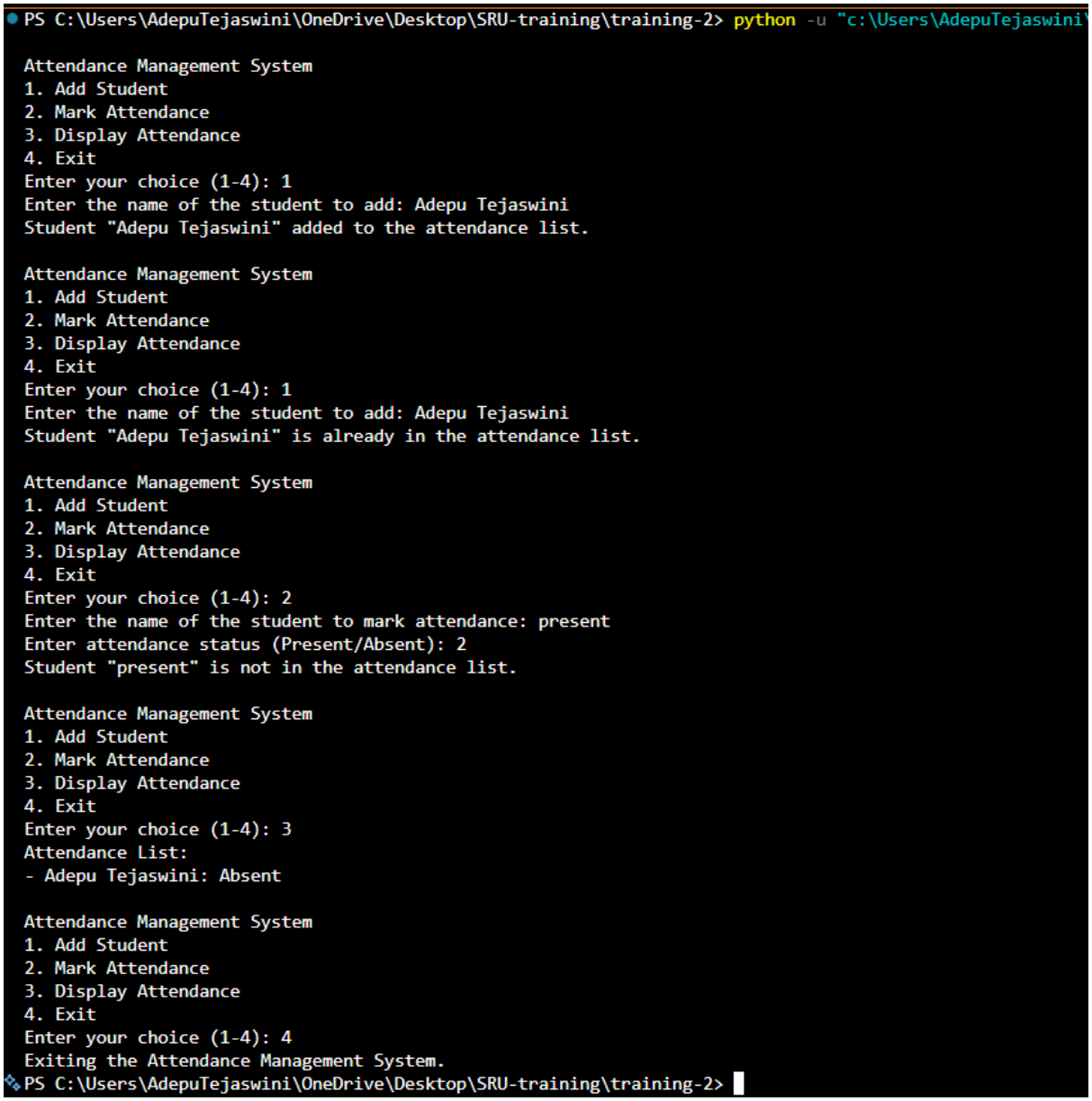
Apply conditional statements to handle basic invalid inputs such as duplicate or missing student names.

Include comments throughout the code to clearly explain the logic and flow of the program.

Code:

```python
class AttendanceManagementSystem:
    def __init__(self):
        # Initialize an empty dictionary to store student attendance
        self.attendance = {}

    def add_student(self, student_name):
        # Add a student to the attendance list
        if student_name in self.attendance:
            print(f'Student "{student_name}" is already in the attendance list.')
        else:
            self.attendance[student_name] = 'Absent'
            print(f'Student "{student_name}" added to the attendance list.')

    def mark_attendance(self, student_name, status):
        # Mark a student's attendance as Present or Absent
        if student_name in self.attendance:
            if status in ['Present', 'Absent']:
                self.attendance[student_name] = status
                print(f'Student "{student_name}" marked as {status}.')
            else:
                print('Invalid status. Please enter "Present" or "Absent".')
        else:
            print(f'Student "{student_name}" is not in the attendance list.')

    def display_attendance(self):
        # Display the attendance list
        if not self.attendance:
            print("No students in the attendance list.")
        else:
            print("Attendance List:")
            for student, status in self.attendance.items():
                print(f"- {student}: {status}")
def main():
    attendance_system = AttendanceManagementSystem()
    while True:
        print("\nAttendance Management System")
        print("1. Add Student")
        print("2. Mark Attendance")
        print("3. Display Attendance")
        print("4. Exit")
        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            student_name = input("Enter the name of the student to add: ")
            attendance_system.add_student(student_name)
        elif choice == '2':
            student_name = input("Enter the name of the student to mark attendance: ")
            status = input("Enter attendance status (Present/Absent): ")
            attendance_system.mark_attendance(student_name, status)
        elif choice == '3':
            attendance_system.display_attendance()
        elif choice == '4':
            print("Exiting the Attendance Management System.")
            break
        else:
            print("Invalid choice. Please try again.")
if __name__ == "__main__":
    main()
```

Output:



Explanation

This program uses a class to manage student attendance by storing names and their status in a dictionary.
Loops and conditional statements allow users to add students, mark them as present or absent, and display attendance while handling invalid inputs.
Clear messages and comments ensure correct attendance tracking and easy understanding of the program flow.

Task Description #5 (AI-Based Code Completion for Conditional Menu Navigation)

Task: Use an AI tool to complete a navigation menu.

Prompt: “Generate a Python program using loops and conditionals to simulate an ATM menu.”

Expected Output:

• AI-generated menu logic.

• Correct option handling.

• Output verification.

Prompt:

Generate a Python program to simulate an ATM menu using loops and conditional statements.

The menu should include options for balance inquiry, deposit, withdrawal, and exit.

Use a loop to keep the menu running until the user exits.

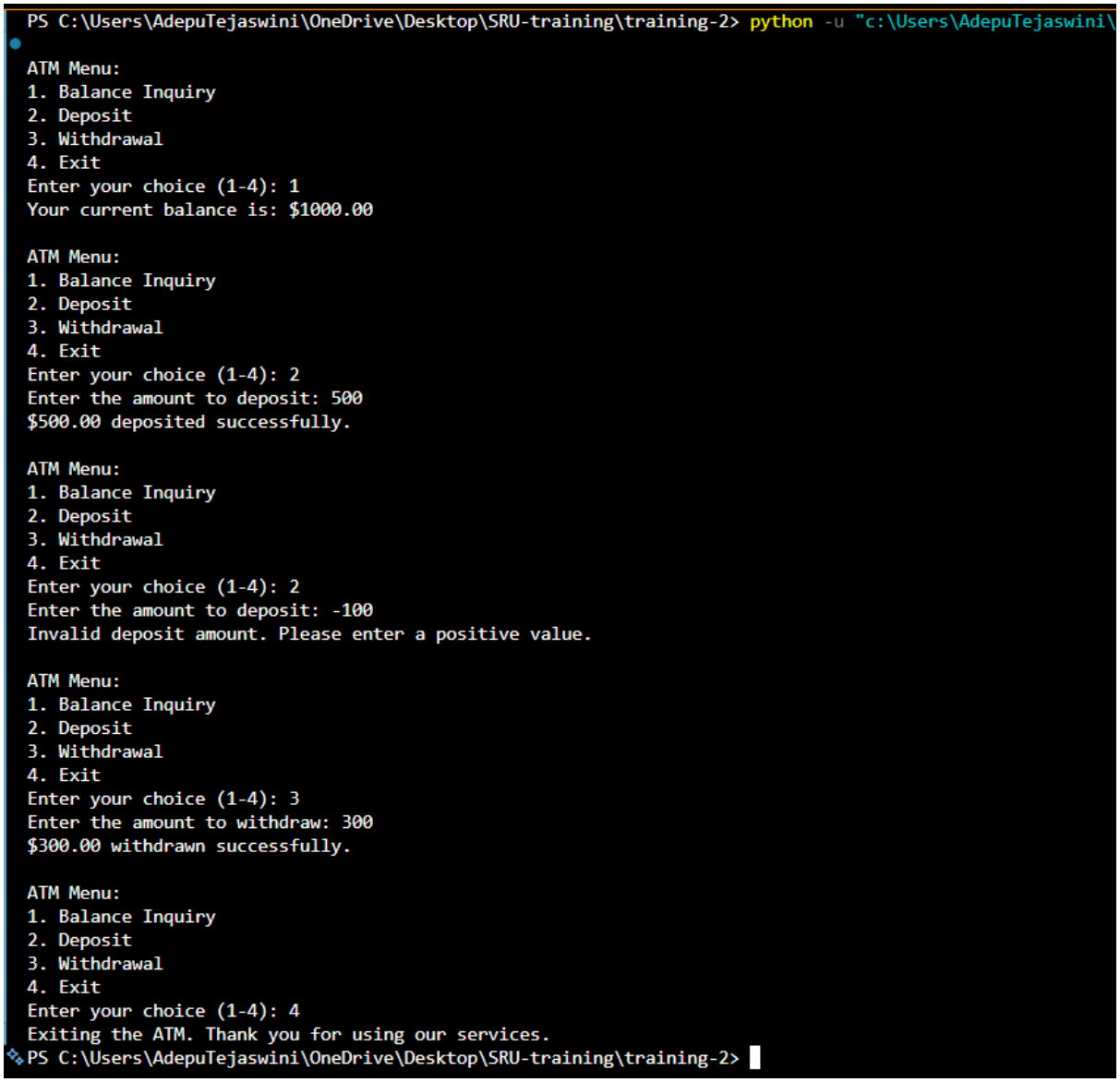
Handle invalid menu choices and invalid transaction amounts.

Generate a complete executable program with comments.

Code:

```python
class ATM:
    def __init__(self, initial_balance=0):
        # Initialize the ATM with an initial balance
        self.balance = initial_balance

    def display_menu(self):
        # Display the ATM menu options
        print("\nATM Menu:")
        print("1. Balance Inquiry")
        print("2. Deposit")
        print("3. Withdrawal")
        print("4. Exit")

    def balance_inquiry(self):
        # Display the current balance
        print(f"Your current balance is: ${self.balance:.2f}")

    def deposit(self, amount):
        # Deposit an amount into the account
        if amount > 0:
            self.balance += amount
            print(f"${amount:.2f} deposited successfully.")
        else:
            print("Invalid deposit amount. Please enter a positive value.")

    def withdrawal(self, amount):
        # Withdraw an amount from the account
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"${amount:.2f} withdrawn successfully.")
            else:
                print("Insufficient balance for this withdrawal.")
        else:
            print("Invalid withdrawal amount. Please enter a positive value.")
def main():
    atm = ATM(initial_balance=1000)  # Starting with an initial balance of $1000
    while True:
        atm.display_menu()
        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            atm.balance_inquiry()
        elif choice == '2':
            try:
                amount = float(input("Enter the amount to deposit: "))
                atm.deposit(amount)
            except ValueError:
                print("Please enter a valid number for the deposit amount.")
        elif choice == '3':
            try:
                amount = float(input("Enter the amount to withdraw: "))
                atm.withdrawal(amount)
            except ValueError:
                print("Please enter a valid number for the withdrawal amount.")
        elif choice == '4':
            print("Exiting the ATM. Thank you for using our services.")
            break
        else:
            print("Invalid choice. Please try again.")
if __name__ == "__main__":
    main()
```

Output:



Explanation

This program simulates an ATM using a loop-driven menu and conditional statements to handle user choices.
It allows balance inquiry, deposit, and withdrawal operations while validating transaction amounts and menu selections.
Clear messages and input validation ensure correct operations and reliable output verification.