AI ASSISTANT CODING

ASSIGNMENT-6.5

NAME : MOGILI KEERTHI

HT.NO : 2303A51427

BATCH : 21

Lab 6: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals

LO1. Use AI-based code completion tools to generate Python code involving classes, loops, and conditionals.

LO2. Interpret and explain AI-generated code line-by-line.

LO3. Identify errors, inefficiencies, or logical flaws in AI- suggested implementations.

LO4. Optimize AI-generated code for better readability and performance.

LO5. Demonstrate ethical and responsible use of AI tools in coding tasks.

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."

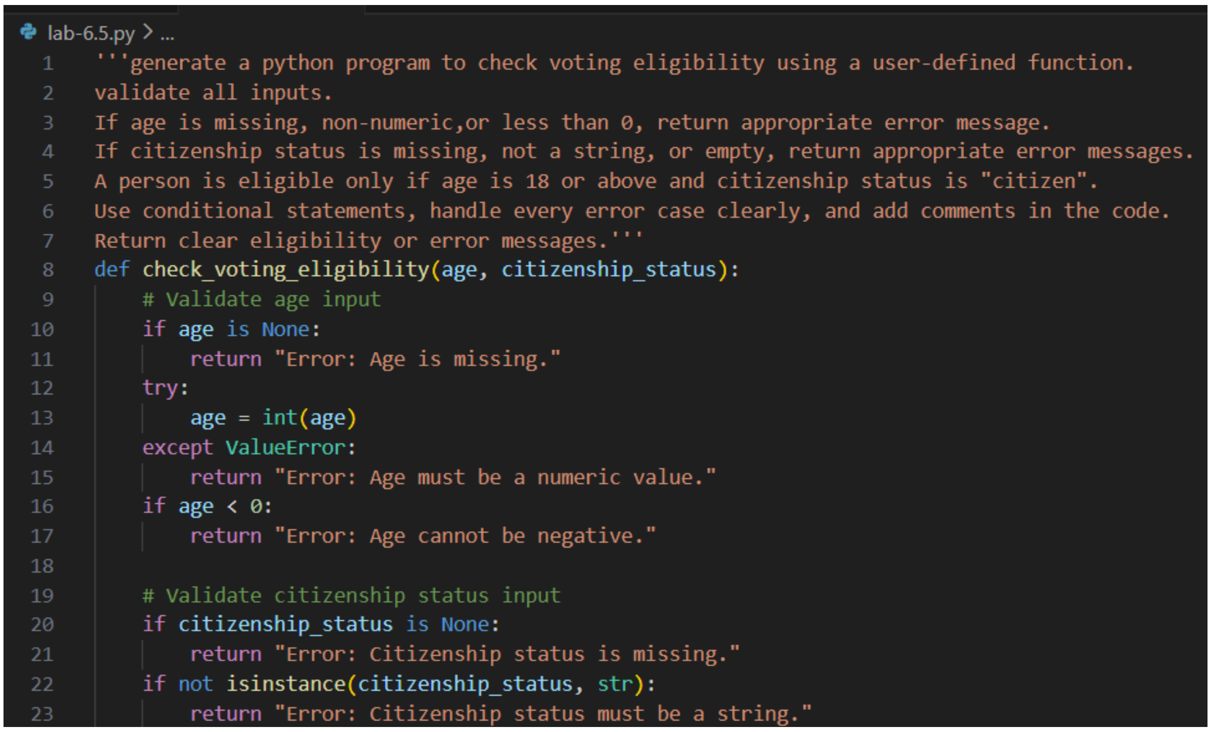
Expected Output:

• AI-generated conditional logic.

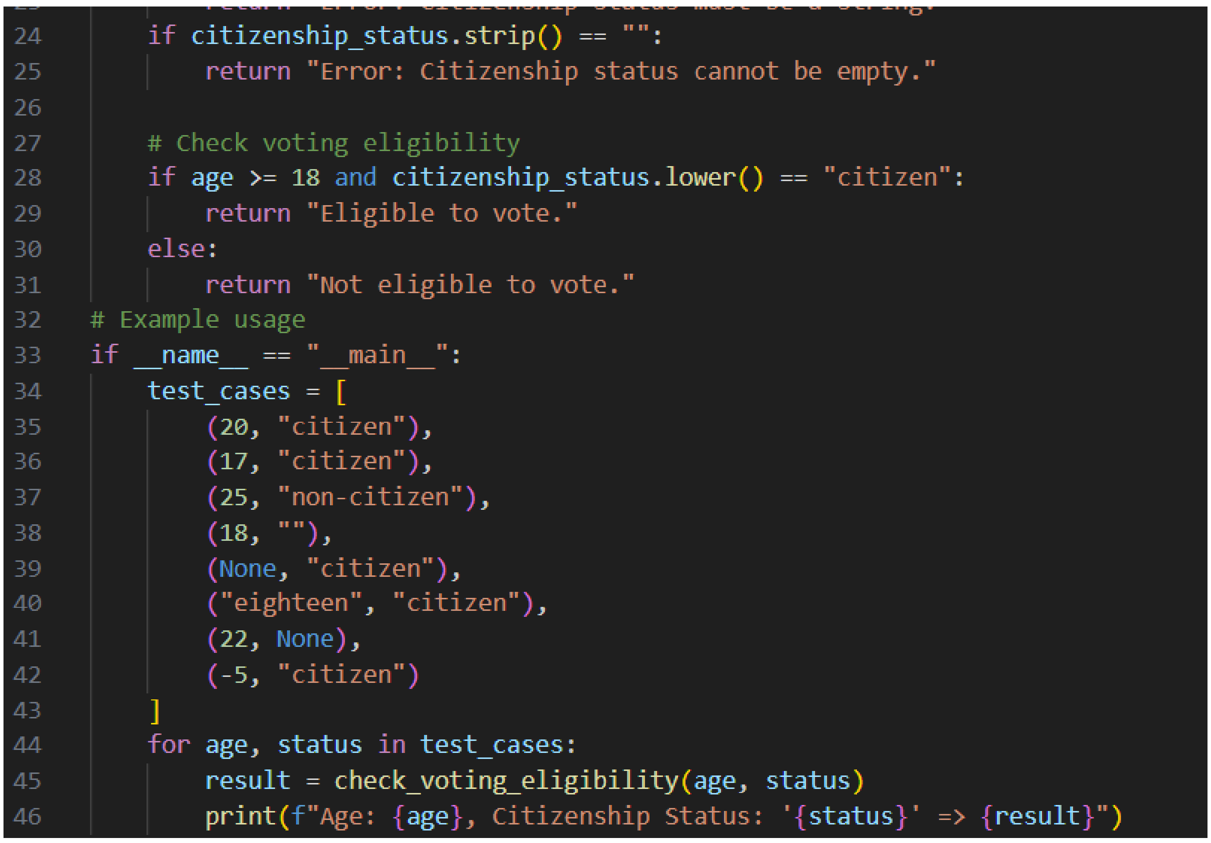
• Correct eligibility decisions.

• Explanation of conditions.

Code:

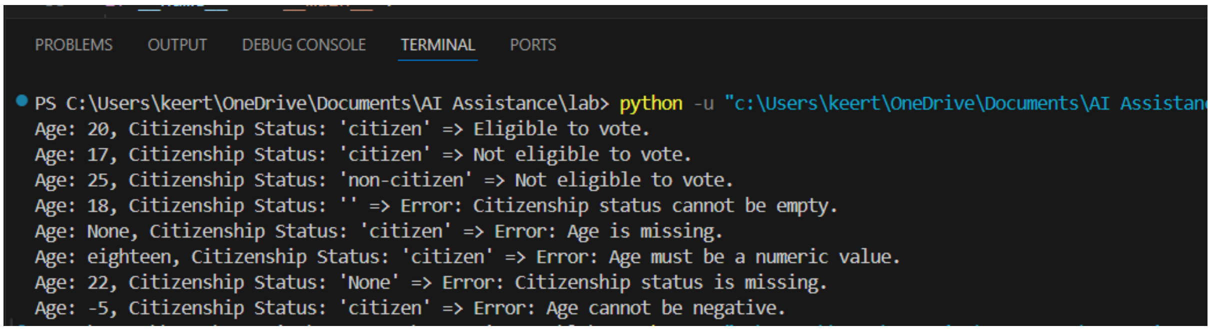


```python
'''generate a python program to check voting eligibility using a user-defined function.
validate all inputs.
If age is missing, non-numeric,or less than 0, return appropriate error message.
If citizenship status is missing, not a string, or empty, return appropriate error messages.
A person is eligible only if age is 18 or above and citizenship status is "citizen".
Use conditional statements, handle every error case clearly, and add comments in the code.
Return clear eligibility or error messages.'''
def check_voting_eligibility(age, citizenship_status):
    # Validate age input
    if age is None:
        return "Error: Age is missing."
    try:
        age = int(age)
    except ValueError:
        return "Error: Age must be a numeric value."
    if age < 0:
        return "Error: Age cannot be negative."

    # Validate citizenship status input
    if citizenship_status is None:
        return "Error: Citizenship status is missing."
    if not isinstance(citizenship_status, str):
        return "Error: Citizenship status must be a string."
```



```python
    if citizenship_status.strip() == "":
        return "Error: Citizenship status cannot be empty."

    # Check voting eligibility
    if age >= 18 and citizenship_status.lower() == "citizen":
        return "Eligible to vote."
    else:
        return "Not eligible to vote."
# Example usage
if __name__ == "__main__":
    test_cases = [
        (20, "citizen"),
        (17, "citizen"),
        (25, "non-citizen"),
        (18, ""),
        (None, "citizen"),
        ("eighteen", "citizen"),
        (22, None),
        (-5, "citizen")
    ]
    for age, status in test_cases:
        result = check_voting_eligibility(age, status)
        print(f"Age: {age}, Citizenship Status: '{status}' => {result}")
```

OUTPUT:



```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> python -u "c:\Users\keert\OneDrive\Documents\AI Assistan
Age: 20, Citizenship Status: 'citizen' => Eligible to vote.
Age: 17, Citizenship Status: 'citizen' => Not eligible to vote.
Age: 25, Citizenship Status: 'non-citizen' => Not eligible to vote.
Age: 18, Citizenship Status: '' => Error: Citizenship status cannot be empty.
Age: None, Citizenship Status: 'citizen' => Error: Age is missing.
Age: eighteen, Citizenship Status: 'citizen' => Error: Age must be a numeric value.
Age: 22, Citizenship Status: 'None' => Error: Citizenship status is missing.
Age: -5, Citizenship Status: 'citizen' => Error: Age cannot be negative.
```

Final Description:

The AI-generated Python program was used to design a voting eligibility check with proper input validation and clear decision logic. The code was carefully reviewed line by line to understand how age and citizenship inputs are validated, including handling missing, invalid, and incorrect values. Logical flaws and potential errors were identified and addressed by adding appropriate condition checks and exception handling. The program was further refined to improve readability and maintainability through meaningful variable names, structured validation, and clear comments. Responsible use of AI tools was ensured by verifying the logic, correcting mistakes, and fully understanding the code rather than copying the AI-generated output without evaluation.

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.”

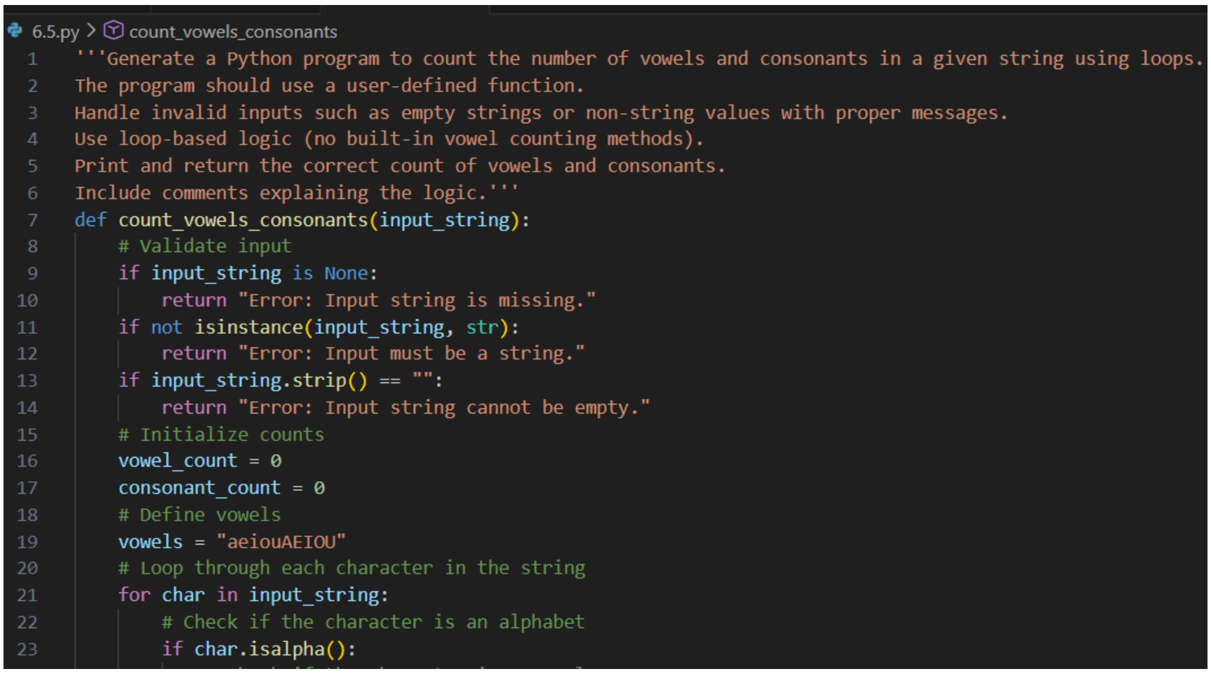
Expected Output:

• AI-generated string processing logic.

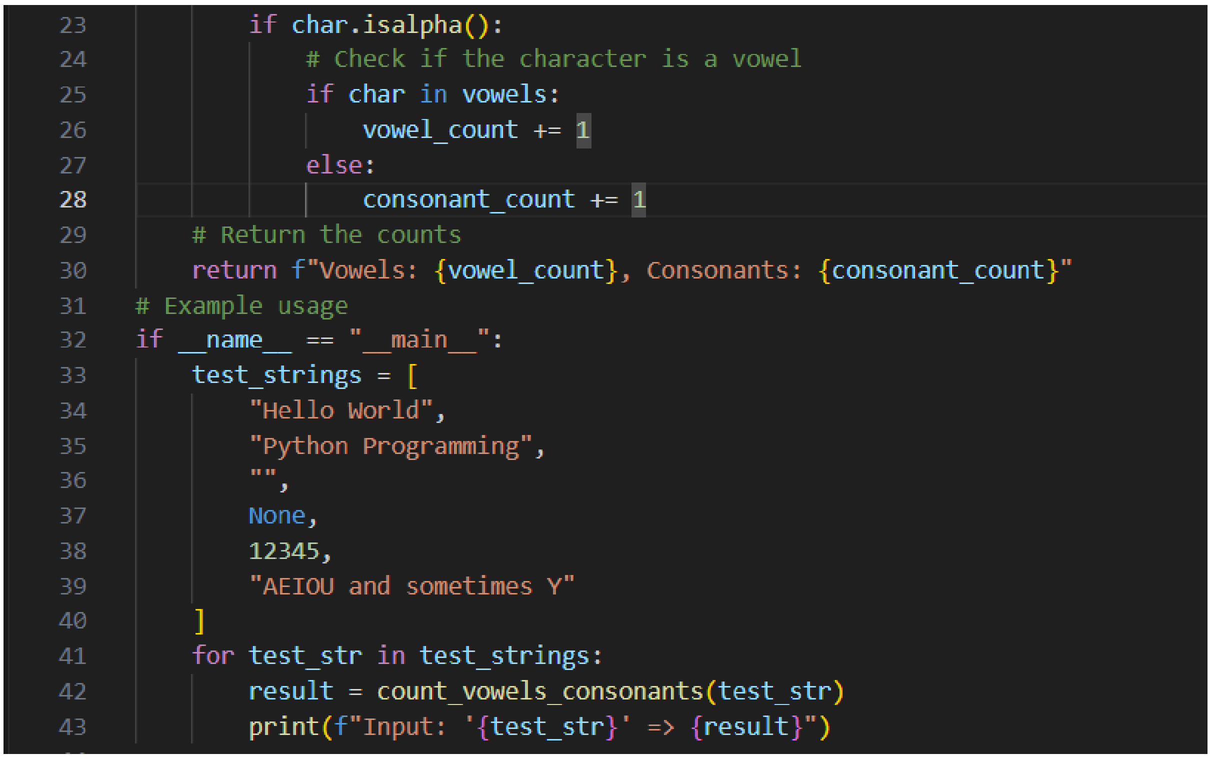
• Correct counts.

• Output verification.

Code:

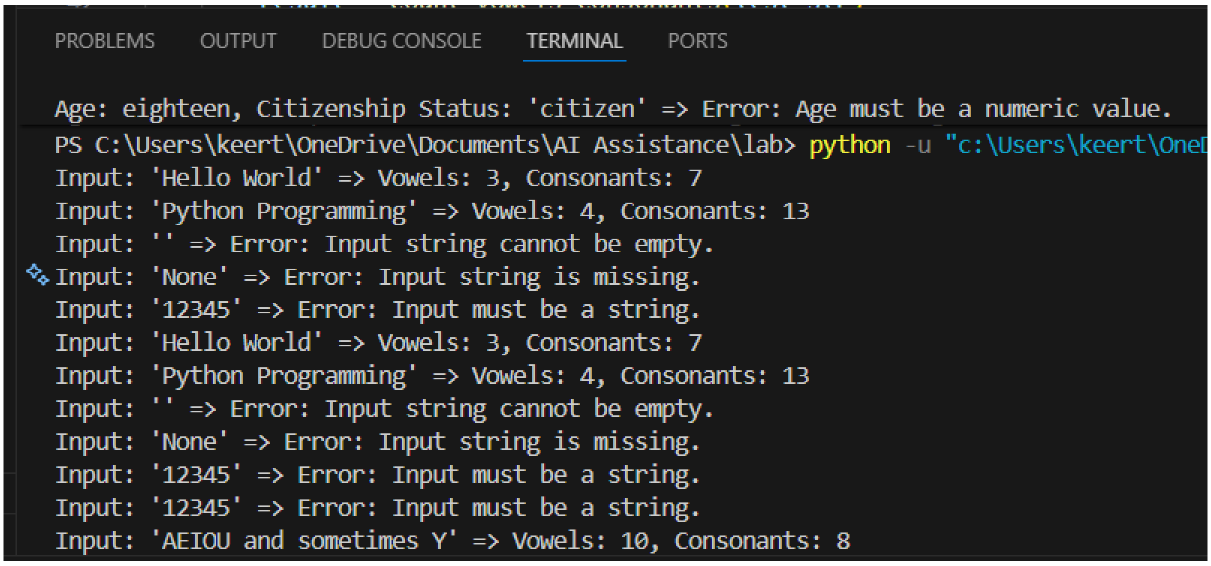


```python
'''Generate a Python program to count the number of vowels and consonants in a given string using loops.
The program should use a user-defined function.
Handle invalid inputs such as empty strings or non-string values with proper messages.
Use loop-based logic (no built-in vowel counting methods).
Print and return the correct count of vowels and consonants.
Include comments explaining the logic.'''
def count_vowels_consonants(input_string):
    # Validate input
    if input_string is None:
        return "Error: Input string is missing."
    if not isinstance(input_string, str):
        return "Error: Input must be a string."
    if input_string.strip() == "":
        return "Error: Input string cannot be empty."
    # Initialize counts
    vowel_count = 0
    consonant_count = 0
    # Define vowels
    vowels = "aeiouAEIOU"
    # Loop through each character in the string
    for char in input_string:
        # Check if the character is an alphabet
        if char.isalpha():
```



```python
        if char.isalpha():
            # Check if the character is a vowel
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
    # Return the counts
    return f"Vowels: {vowel_count}, Consonants: {consonant_count}"
# Example usage
if __name__ == "__main__":
    test_strings = [
        "Hello World",
        "Python Programming",
        "",
        None,
        12345,
        "AEIOU and sometimes Y"
    ]
    for test_str in test_strings:
        result = count_vowels_consonants(test_str)
        print(f"Input: '{test_str}' => {result}")
```

Output:



```
Age: eighteen, Citizenship Status: 'citizen' => Error: Age must be a numeric value.
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> python -u "c:\Users\keert\OneD
Input: 'Hello World' => Vowels: 3, Consonants: 7
Input: 'Python Programming' => Vowels: 4, Consonants: 13
Input: '' => Error: Input string cannot be empty.
Input: 'None' => Error: Input string is missing.
Input: '12345' => Error: Input must be a string.
Input: 'Hello World' => Vowels: 3, Consonants: 7
Input: 'Python Programming' => Vowels: 4, Consonants: 13
Input: '' => Error: Input string cannot be empty.
Input: 'None' => Error: Input string is missing.
Input: '12345' => Error: Input must be a string.
Input: '12345' => Error: Input must be a string.
Input: 'AEIOU and sometimes Y' => Vowels: 10, Consonants: 8
```

Final Description :

The AI tool was used to generate a Python program that applies a user-defined function, loop-based logic, and conditional statements to count vowels and consonants in a string. The generated code was carefully examined line by line to understand how input validation, character checking, and counting logic work together to produce correct results. During the review process, potential issues such as empty inputs, non-string values, and invalid characters were identified and handled appropriately to ensure logical correctness. The program was further refined to improve readability and efficiency through clear comments, structured conditions, and meaningful variable names. Responsible use of AI tools was demonstrated by validating the generated solution, correcting errors, and ensuring a clear understanding of the logic rather than using the output without evaluation.

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.”

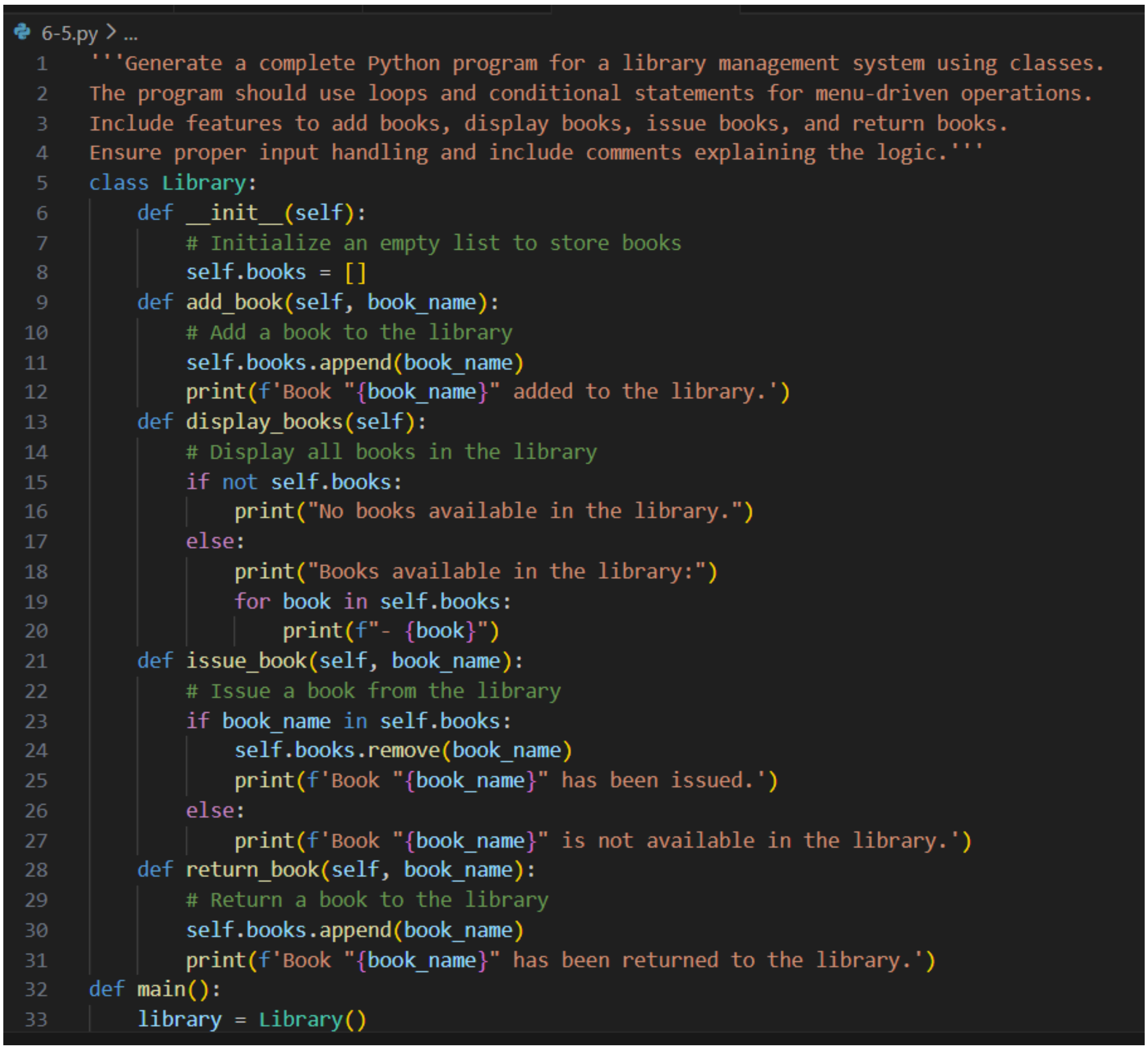
Expected Output:

• Complete AI-generated program.

• Review of AI suggestions quality.

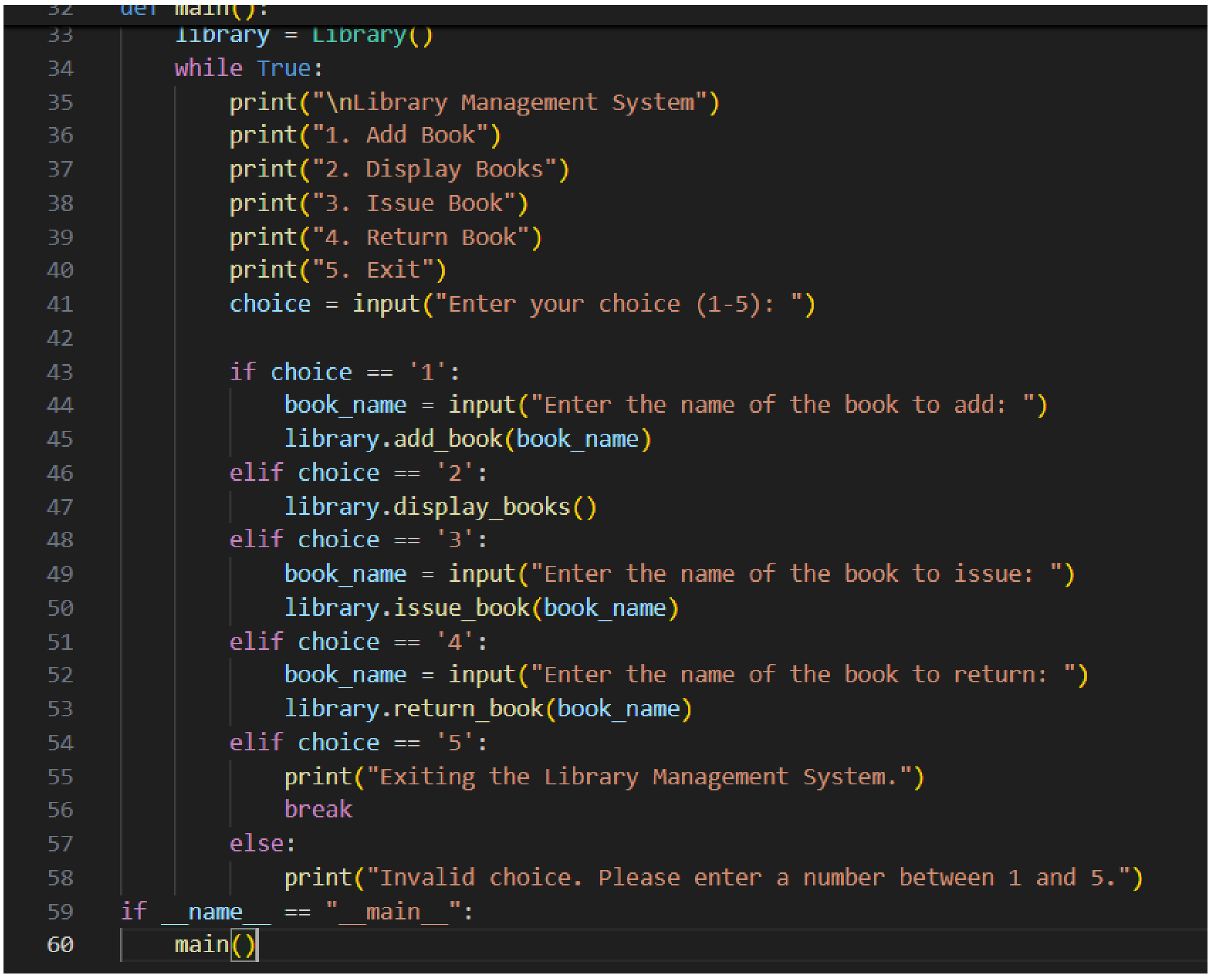
• Short reflection on AI-assisted coding experience.

Code



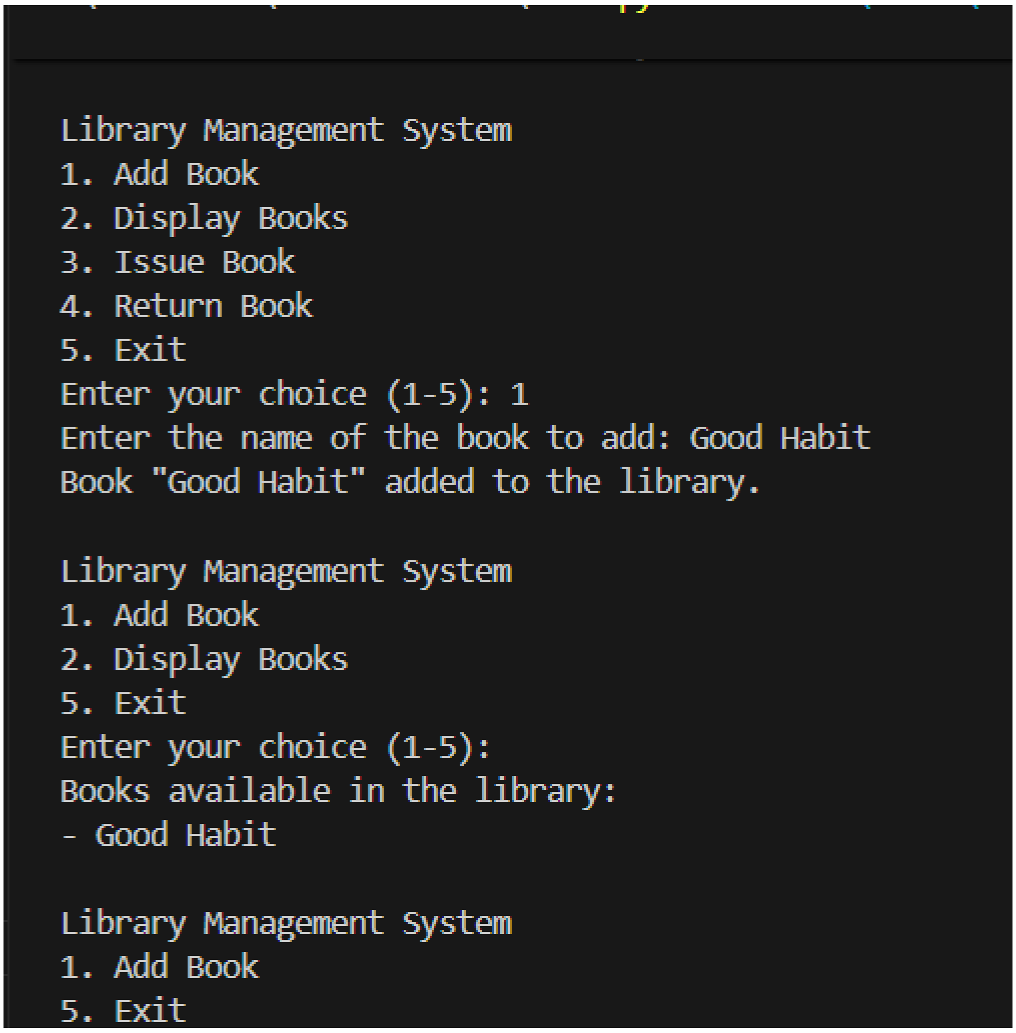
6-5.py > ...

```python
'''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.'''
class Library:
    def __init__(self):
        # Initialize an empty list to store books
        self.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):
        # Issue a book from the library
        if book_name in self.books:
            self.books.remove(book_name)
            print(f'Book "{book_name}" has been issued.')
        else:
            print(f'Book "{book_name}" is not available in the library.')
    def return_book(self, book_name):
        # Return a book to the library
        self.books.append(book_name)
        print(f'Book "{book_name}" has been returned to the library.')
def main():
    library = Library()
```



```python
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: ")
            library.issue_book(book_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 enter a number between 1 and 5.")
if __name__ == "__main__":
    main()
```

Output:



```
Library Management System
1. Add Book
2. Display Books
3. Issue Book
4. Return Book
5. Exit
Enter your choice (1-5): 1
Enter the name of the book to add: Good Habit
Book "Good Habit" added to the library.

Library Management System
1. Add Book
2. Display Books
5. Exit
Enter your choice (1-5):
Books available in the library:
- Good Habit

Library Management System
1. Add Book
5. Exit
```

Final Description:

The AI-generated Python program was used to design a simple library management system using a class-based approach along with loops and conditional statements for menu-driven operations. The code was carefully reviewed line by line to understand how object-oriented concepts, such as classes, methods, and instance variables, manage book records and availability status. Logical conditions were examined to ensure correct handling of operations like adding, displaying, issuing, and returning books, while also preventing invalid actions such as issuing an already issued book. The program was refined for better readability and maintainability through meaningful method names, clear comments, and structured input validation within the menu loop. Responsible use of AI tools was demonstrated by verifying the generated logic, handling user input errors properly, and ensuring a clear understanding of the program's functionality rather than relying on the AI output without evaluation.

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.”

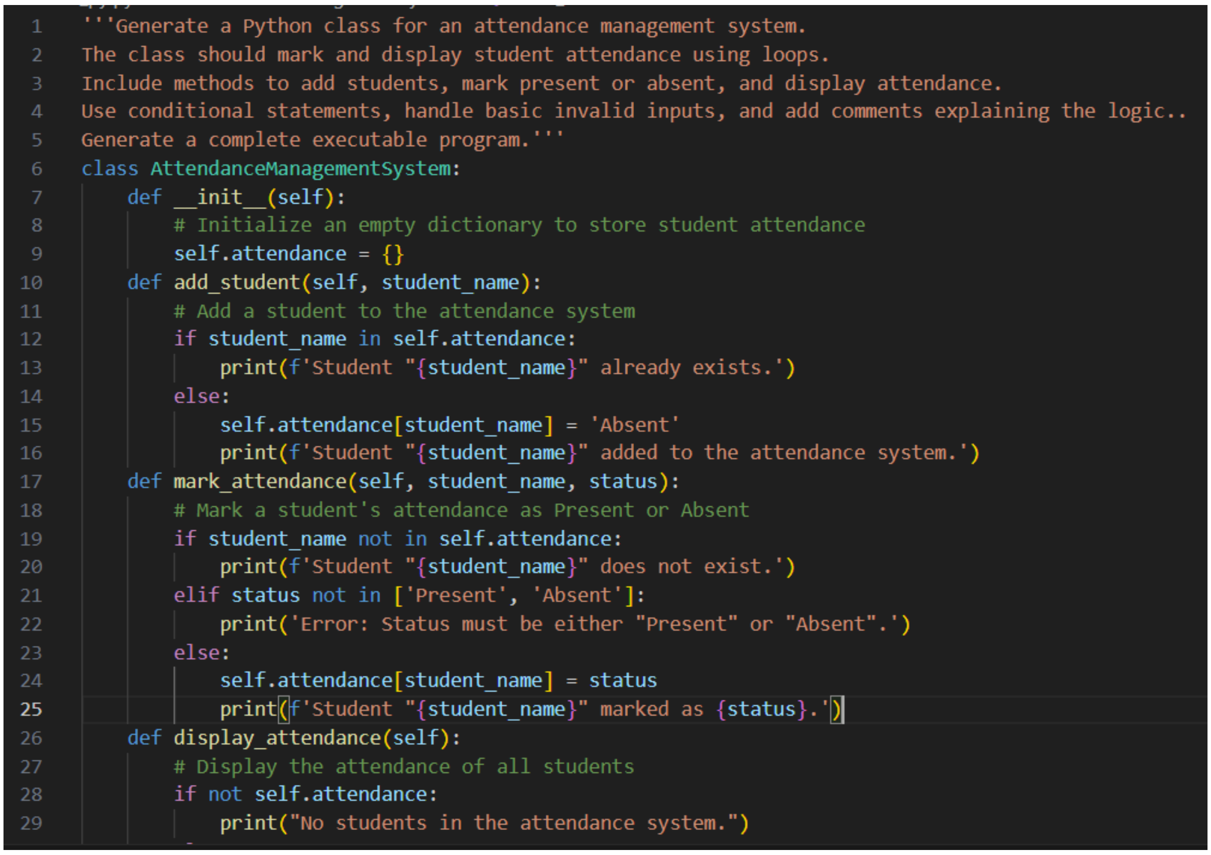
Expected Output:

• AI-generated attendance logic.

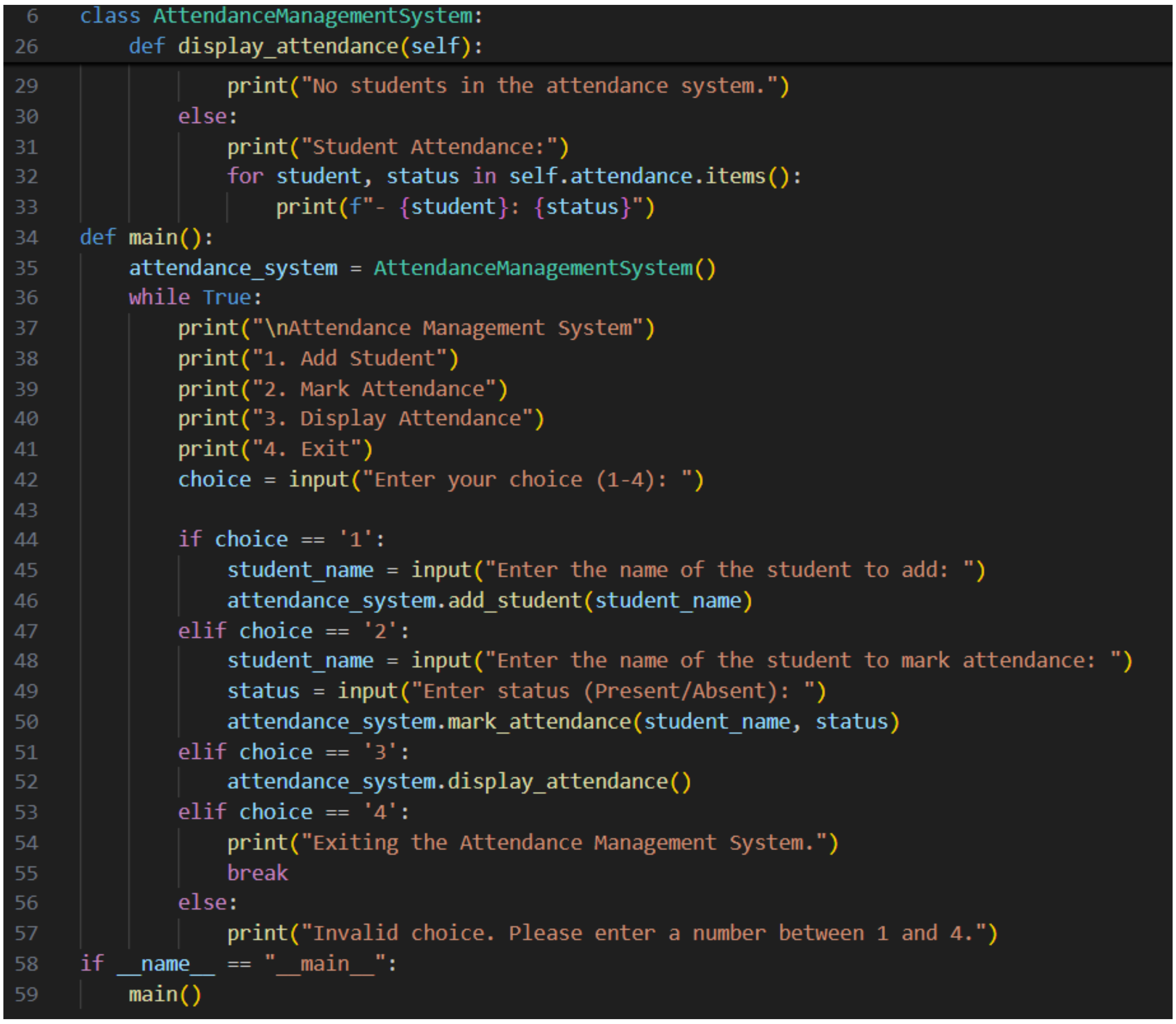
• Correct display of attendance.

• Test cases.

Code:

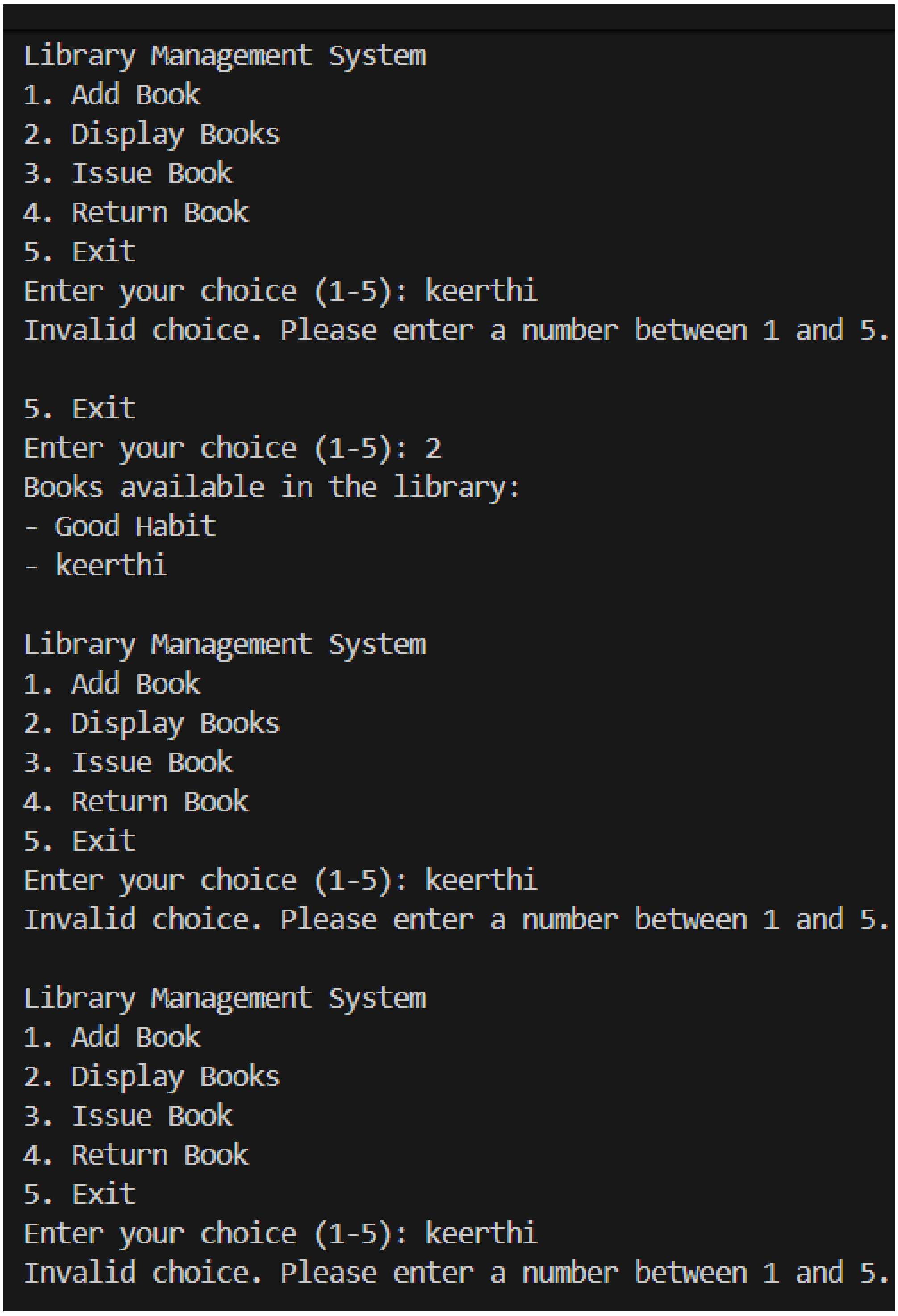


```python
'''Generate a Python class for an attendance management system.
The class should mark and display student attendance using loops.
Include methods to add students, mark present or absent, and display attendance.
Use conditional statements, handle basic invalid inputs, and add comments explaining the logic..
Generate a complete executable program.'''
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 system
        if student_name in self.attendance:
            print(f'Student "{student_name}" already exists.')
        else:
            self.attendance[student_name] = 'Absent'
            print(f'Student "{student_name}" added to the attendance system.')
    def mark_attendance(self, student_name, status):
        # Mark a student's attendance as Present or Absent
        if student_name not in self.attendance:
            print(f'Student "{student_name}" does not exist.')
        elif status not in ['Present', 'Absent']:
            print('Error: Status must be either "Present" or "Absent".')
        else:
            self.attendance[student_name] = status
            print(f'Student "{student_name}" marked as {status}.')
    def display_attendance(self):
        # Display the attendance of all students
        if not self.attendance:
            print("No students in the attendance system.")
```



```python
class AttendanceManagementSystem:
    def display_attendance(self):
                print("No students in the attendance system.")
            else:
                print("Student Attendance:")
                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 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 enter a number between 1 and 4.")
if __name__ == "__main__":
    main()
```

Output:



```
Library Management System
1. Add Book
2. Display Books
3. Issue Book
4. Return Book
5. Exit
Enter your choice (1-5): keerthi
Invalid choice. Please enter a number between 1 and 5.

5. Exit
Enter your choice (1-5): 2
Books available in the library:
- Good Habit
- keerthi

Library Management System
1. Add Book
2. Display Books
3. Issue Book
4. Return Book
5. Exit
Enter your choice (1-5): keerthi
Invalid choice. Please enter a number between 1 and 5.

Library Management System
1. Add Book
2. Display Books
3. Issue Book
4. Return Book
5. Exit
Enter your choice (1-5): keerthi
Invalid choice. Please enter a number between 1 and 5.
```

Final Description:

The AI-generated Python program was used to develop an attendance management system using a class-based structure along with loops and conditional statements for menu-driven operations. The code was reviewed in detail to understand how methods are used to add students, mark attendance as present or absent, and display attendance records using dictionary-based storage. Conditional checks were analysed to ensure proper handling of invalid inputs such as empty names, incorrect attendance status, and non-existing students. The implementation was refined to improve readability and maintainability by using meaningful method names, clear comments, and structured control flow within the loop. Responsible use of AI tools was demonstrated by validating the generated logic, handling edge cases correctly, and ensuring a clear understanding of the program's functionality rather than relying on the AI output without verification.

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."

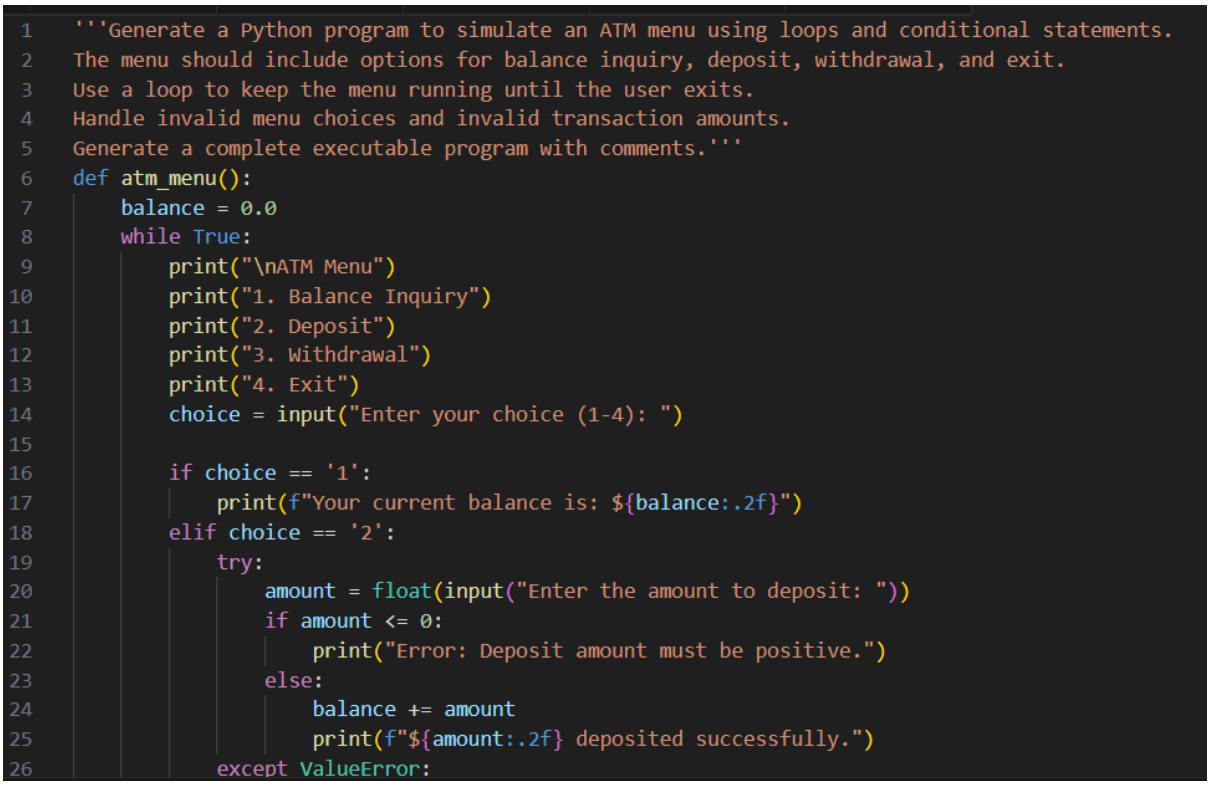
Expected Output:

• AI-generated menu logic.

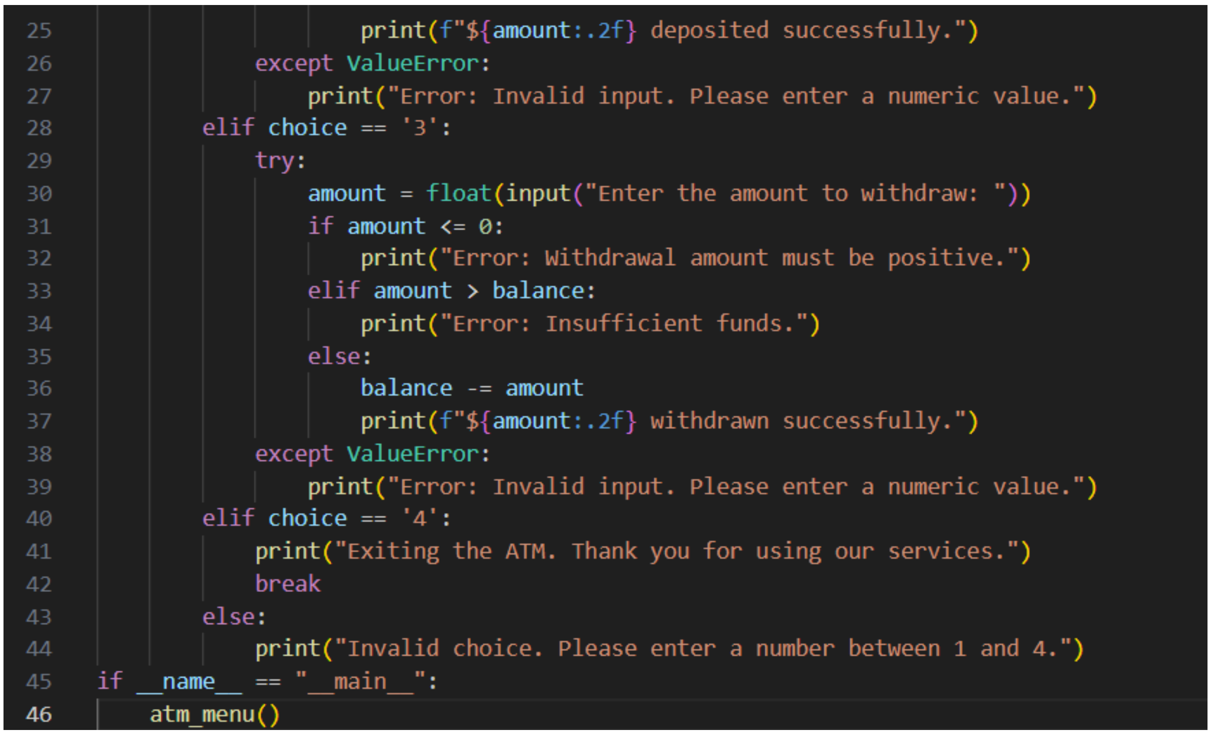
• Correct option handling.

• Output verification.

Code:

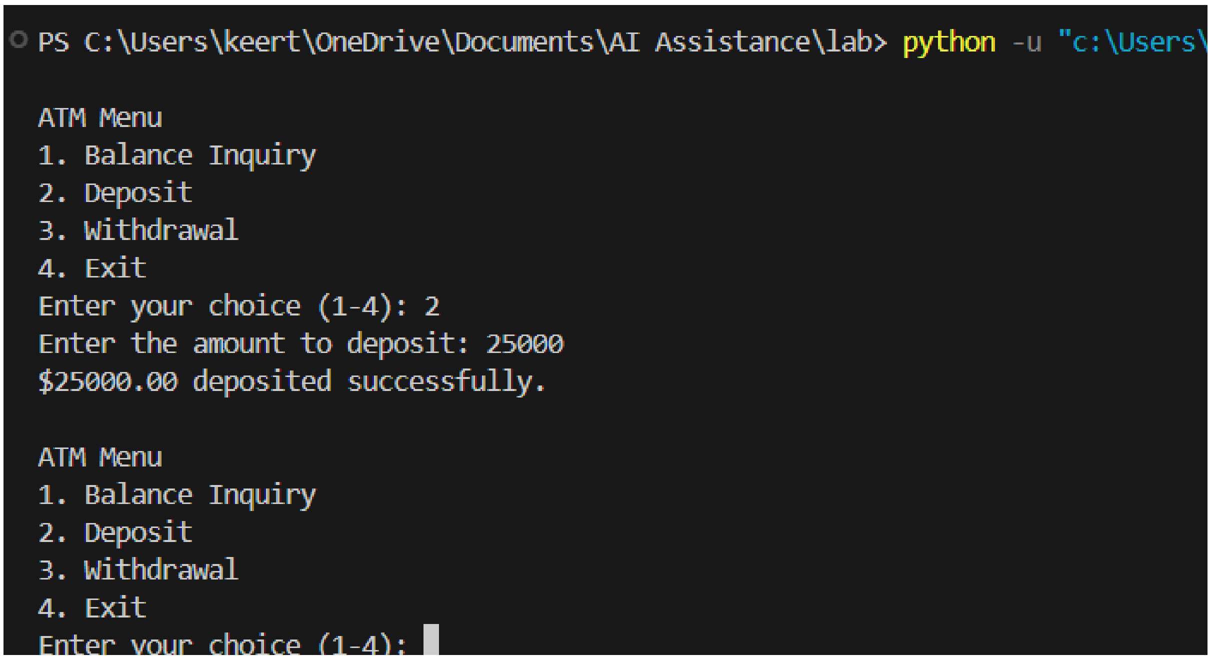


```python
'''Generate a Python program to simulate an ATM menu using loops and conditional statements.
The menu should include options for balance inquiry, deposit, withdrawal, and exit.
Use a loop to keep the menu running until the user exits.
Handle invalid menu choices and invalid transaction amounts.
Generate a complete executable program with comments.'''
def atm_menu():
    balance = 0.0
    while True:
        print("\nATM Menu")
        print("1. Balance Inquiry")
        print("2. Deposit")
        print("3. Withdrawal")
        print("4. Exit")
        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            print(f"Your current balance is: ${balance:.2f}")
        elif choice == '2':
            try:
                amount = float(input("Enter the amount to deposit: "))
                if amount <= 0:
                    print("Error: Deposit amount must be positive.")
                else:
                    balance += amount
                    print(f"${amount:.2f} deposited successfully.")
            except ValueError:
```



```python
                    print(f"${amount:.2f} deposited successfully.")
            except ValueError:
                print("Error: Invalid input. Please enter a numeric value.")
        elif choice == '3':
            try:
                amount = float(input("Enter the amount to withdraw: "))
                if amount <= 0:
                    print("Error: Withdrawal amount must be positive.")
                elif amount > balance:
                    print("Error: Insufficient funds.")
                else:
                    balance -= amount
                    print(f"${amount:.2f} withdrawn successfully.")
            except ValueError:
                print("Error: Invalid input. Please enter a numeric value.")
        elif choice == '4':
            print("Exiting the ATM. Thank you for using our services.")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
if __name__ == "__main__":
    atm_menu()
```

Output:



```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> python -u "c:\Users\

ATM Menu
1. Balance Inquiry
2. Deposit
3. Withdrawal
4. Exit
Enter your choice (1-4): 2
Enter the amount to deposit: 25000
$25000.00 deposited successfully.

ATM Menu
1. Balance Inquiry
2. Deposit
3. Withdrawal
4. Exit
Enter your choice (1-4):
```

Final Description:

The AI-generated Python program was used to simulate an ATM system using loops and conditional statements to provide a menu-driven interface. The code was carefully examined to understand how the loop keeps the menu running until the user chooses to exit and how conditional branches handle balance inquiry, deposit, and withdrawal operations. Input validation was analysed to ensure that invalid menu selections, non-numeric inputs, negative amounts, and insufficient balance cases are handled correctly. The program structure was reviewed to identify and prevent logical errors, while clear comments and meaningful variable names were used to improve readability and maintainability. Responsible use of AI tools was demonstrated by verifying the generated logic, testing different transaction scenarios, and ensuring correct behaviour rather than relying on the AI output without evaluation**.**