AI Assisted Coding

Assignment – 6.5

Name: Md.Shahid

Batch: 21

HtNo: 2303A51421

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.”

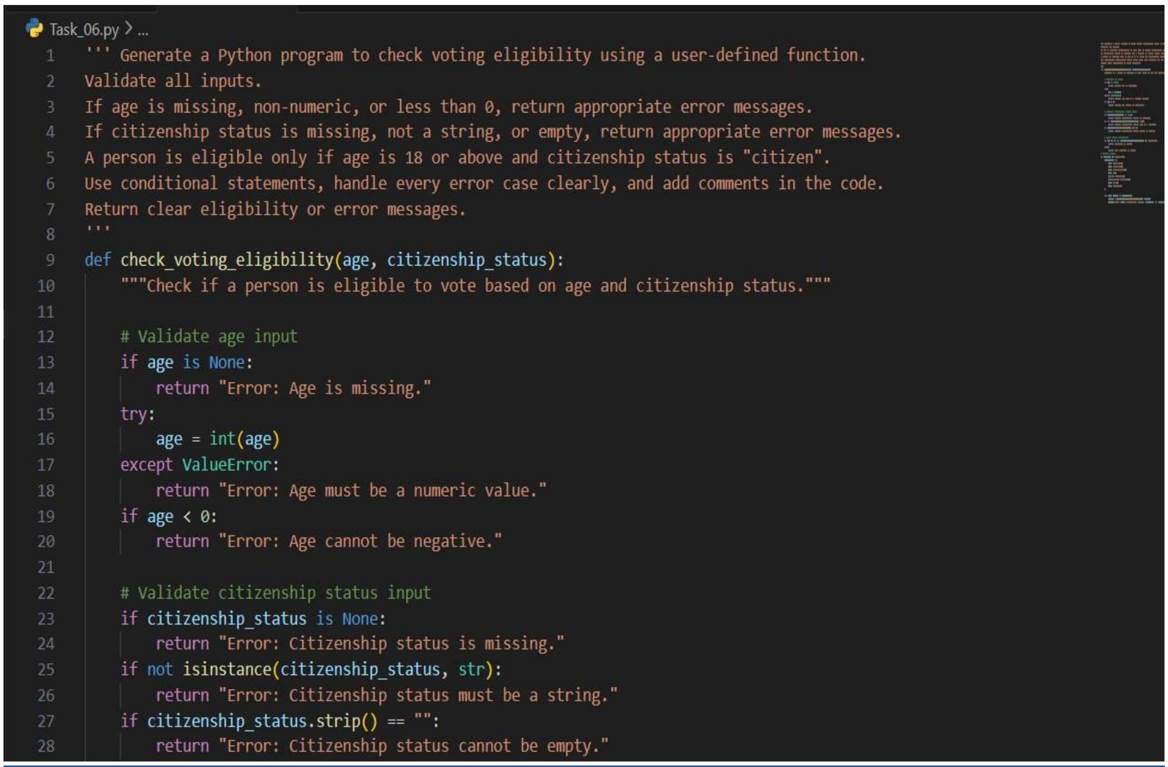
Expected Output:

• AI-generated conditional logic.

• Correct eligibility decisions.

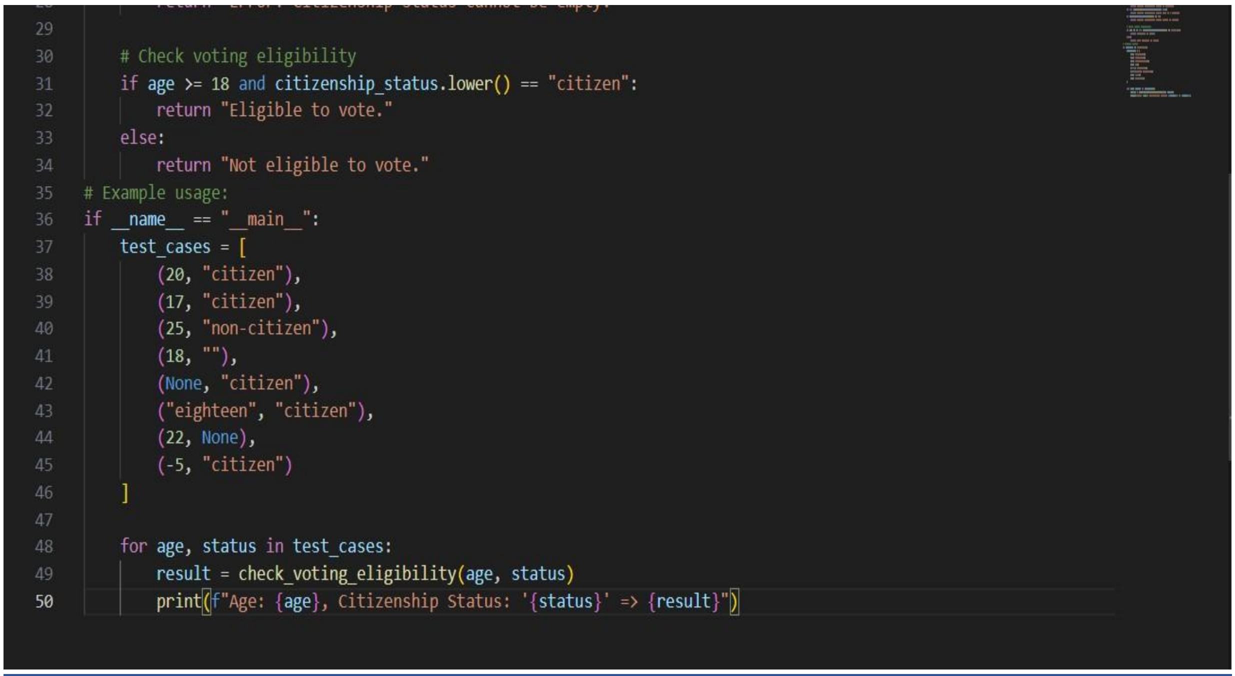
• Explanation of conditions.

Code:



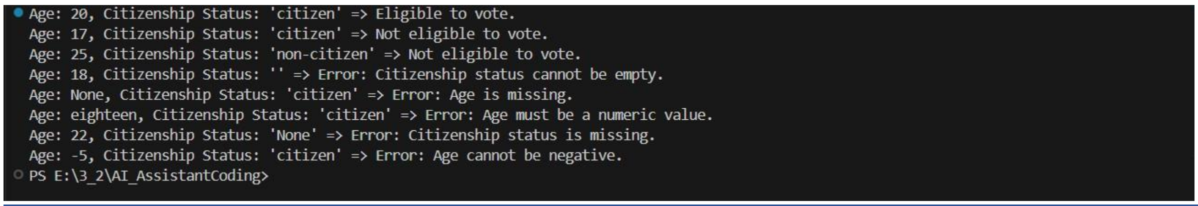
Task_06.py > ...

```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 messages.
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):
    """Check if a person is eligible to vote based on age and 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."
    if citizenship_status.strip() == "":
        return "Error: Citizenship status cannot be empty."
```



```python

    # 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:



```
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.
PS E:\3_2\AI_AssistantCoding>
```

Explanation:

The AI-generated Python program was developed to determine voting eligibility using structured input validation and precise decision logic. Each section of the code was analyzed systematically to verify how age and citizenship data were collected, validated, and processed, including safeguards against missing, invalid, or inconsistent inputs. Weak logic paths and potential runtime errors were identified and corrected through stronger conditional checks and proper exception handling. The program was then refactored to enhance clarity and maintainability by using descriptive variable names, organized validation steps, and well-documented comments. Rather than relying blindly on AI output, the logic was independently verified, errors were fixed, and the entire implementation was fully understood to ensure accuracy and reliability.

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."

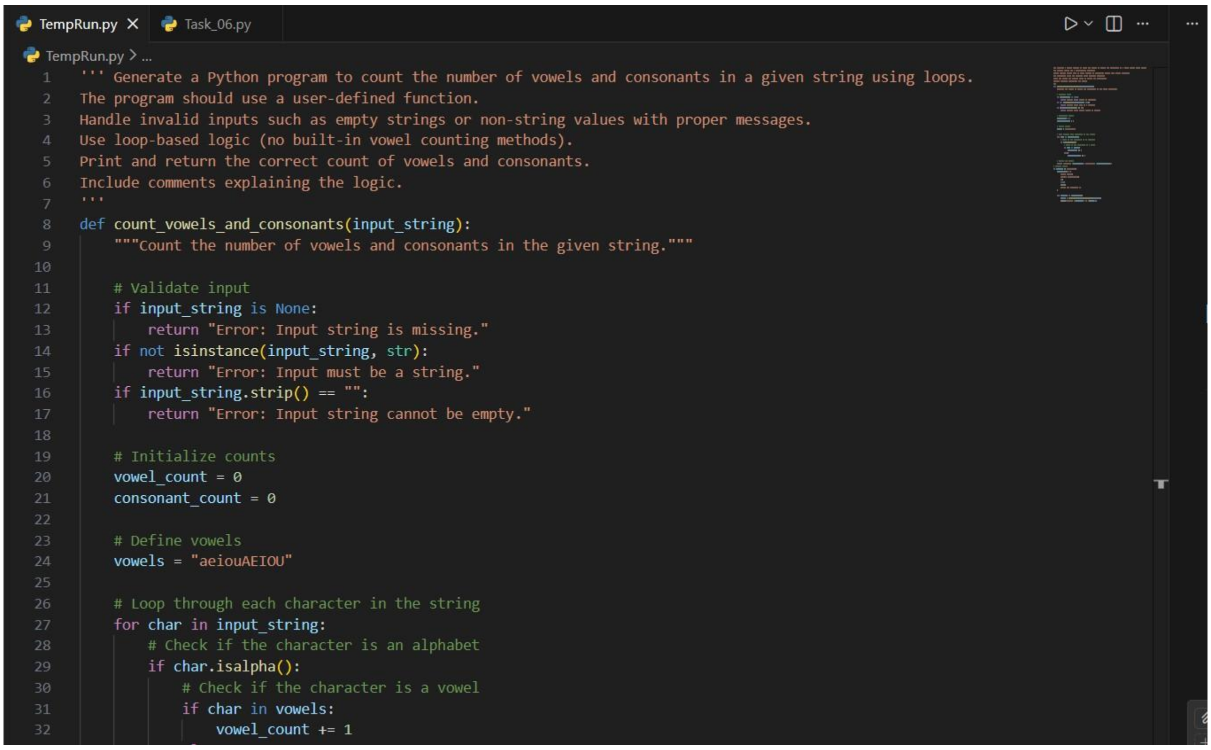
Expected Output:

• AI-generated string processing logic.

• Correct counts.

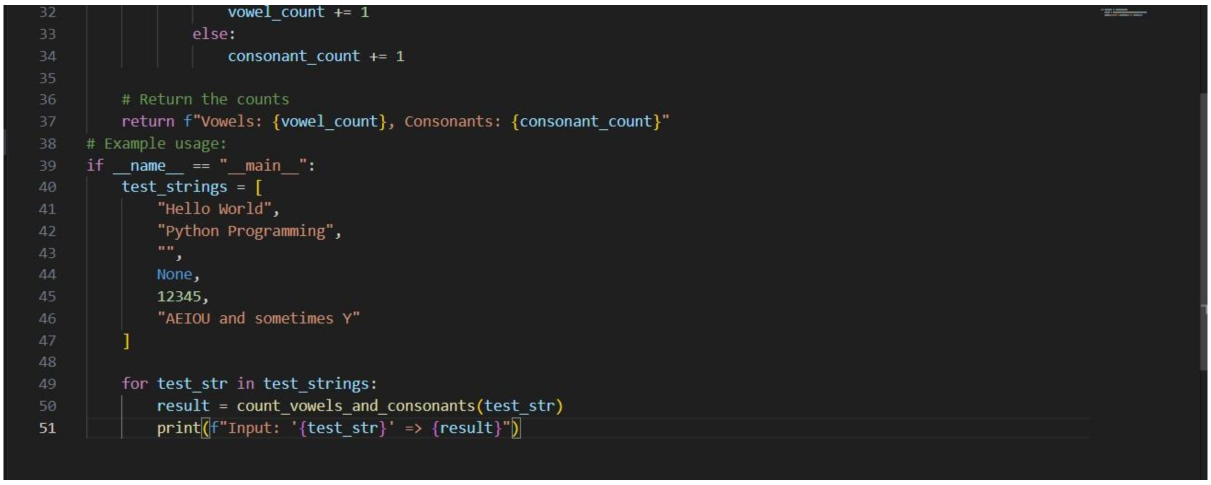
• Output verification.

Code:



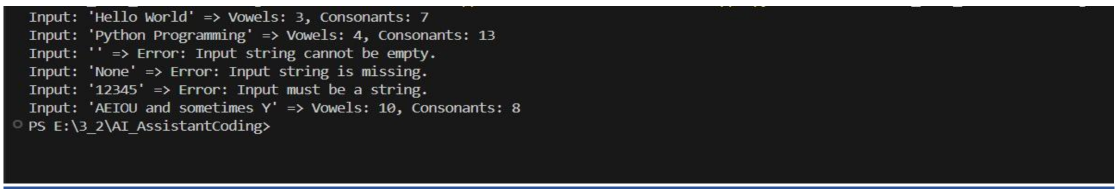
TempRun.py   Task_06.py

```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_and_consonants(input_string):
    """Count the number of vowels and consonants in the given 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():
            # Check if the character is a vowel
            if char in vowels:
                vowel_count += 1
```



```python
                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_and_consonants(test_str)
        print(f"Input: '{test_str}' => {result}")
```

Output:



```
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: 'AEIOU and sometimes Y' => Vowels: 10, Consonants: 8
PS E:\3_2\AI_AssistantCoding>
```

Explanation :

The AI tool was used to generate a Python program that counts vowels and consonants in a string using a user-defined function, iterative logic, and conditional checks. The code was systematically reviewed to understand how inputs were validated, characters were processed, and counts were computed accurately. Edge cases such as empty strings, non-string inputs, and invalid characters were identified and handled through appropriate validation and safeguards. The implementation was then refined to improve clarity and efficiency by using descriptive variable names, organized control flow, and concise comments. Rather than relying blindly on the AI output, the solution was verified, corrected where necessary, and fully understood to ensure correctness and reliability.

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.”

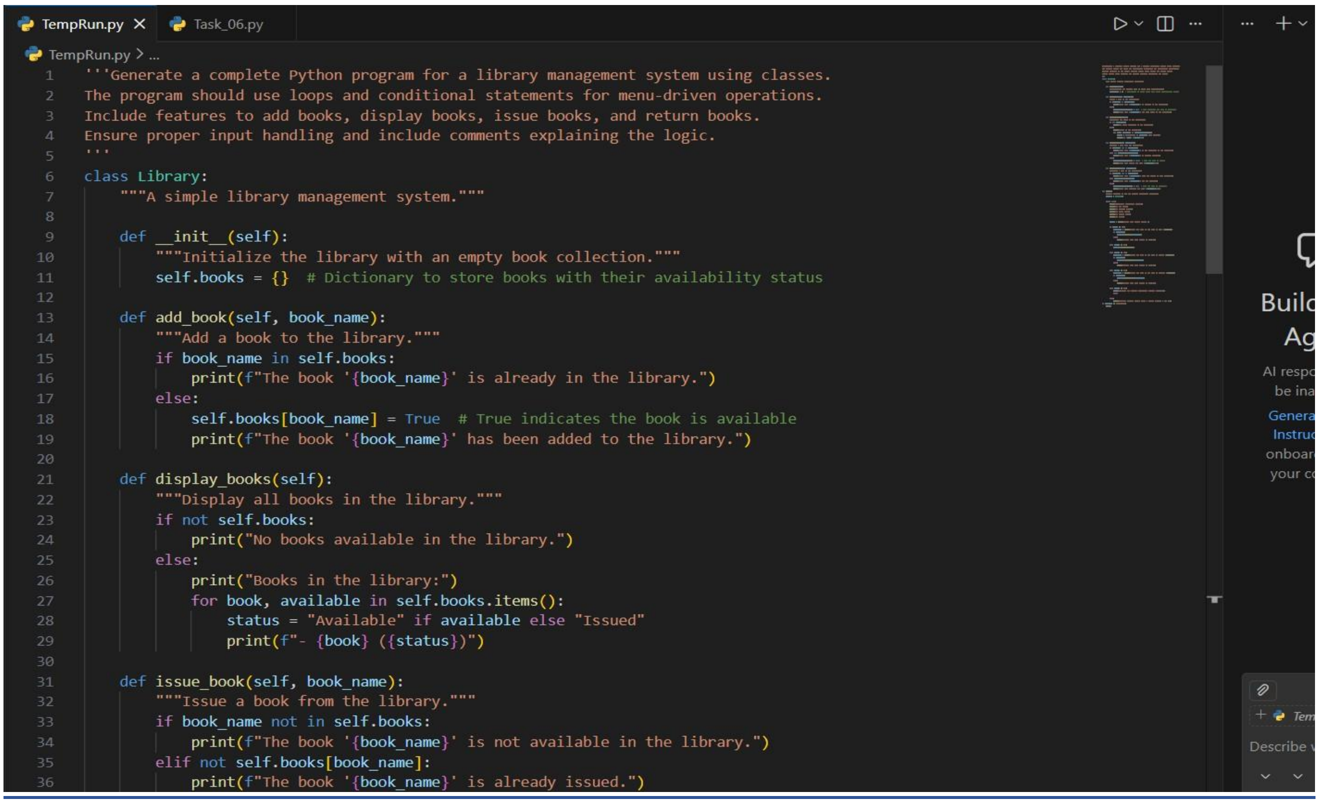
Expected Output:

• Complete AI-generated program.

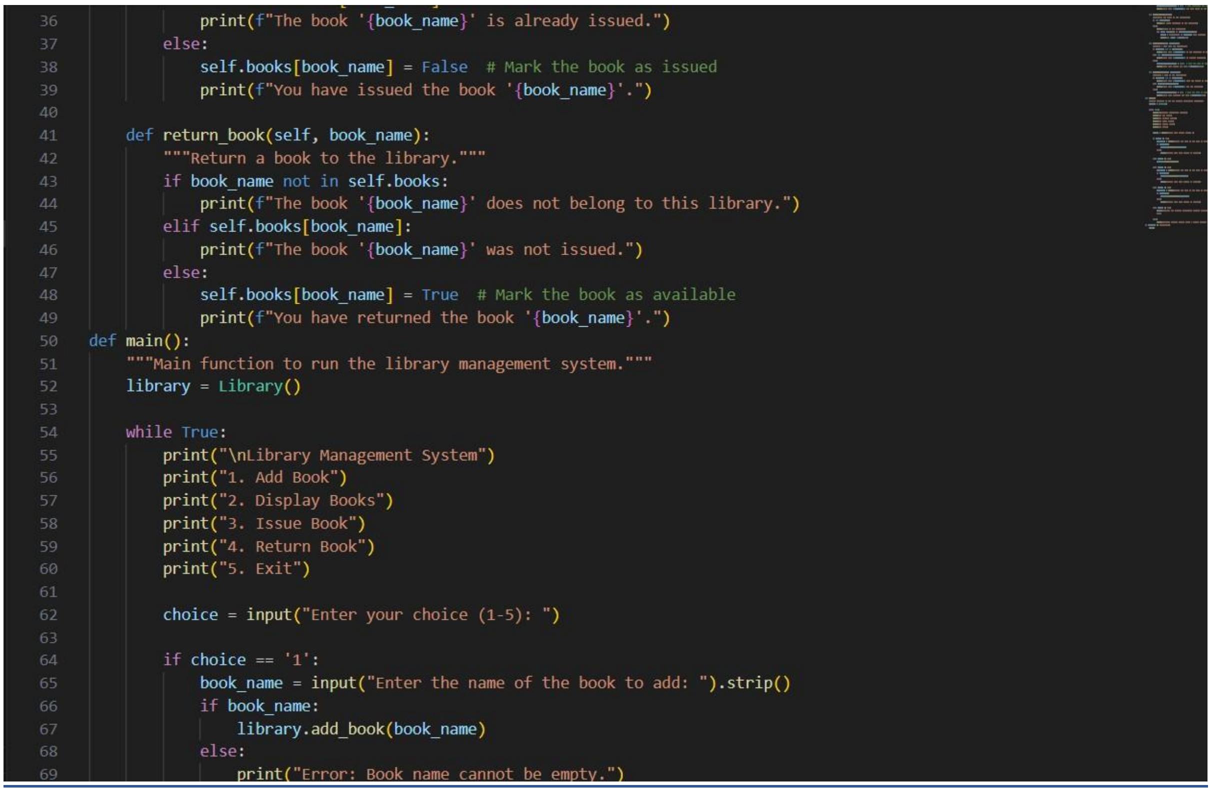
• Review of AI suggestions quality.

• Short reflection on AI-assisted coding experience

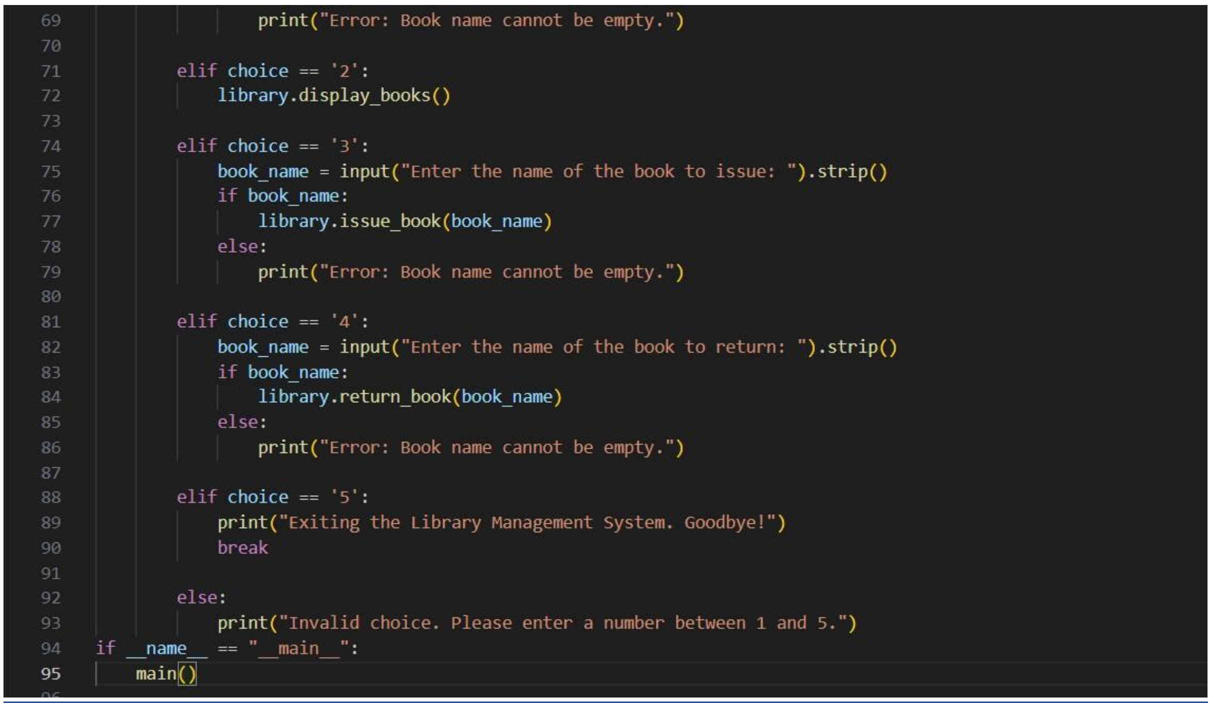
Code:



```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:
    """A simple library management system."""

    def __init__(self):
        """Initialize the library with an empty book collection."""
        self.books = {}  # Dictionary to store books with their availability status

    def add_book(self, book_name):
        """Add a book to the library."""
        if book_name in self.books:
            print(f"The book '{book_name}' is already in the library.")
        else:
            self.books[book_name] = True  # True indicates the book is available
            print(f"The book '{book_name}' has been 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 in the library:")
            for book, available in self.books.items():
                status = "Available" if available else "Issued"
                print(f"- {book} ({status})")

    def issue_book(self, book_name):
        """Issue a book from the library."""
        if book_name not in self.books:
            print(f"The book '{book_name}' is not available in the library.")
        elif not self.books[book_name]:
            print(f"The book '{book_name}' is already issued.")
```

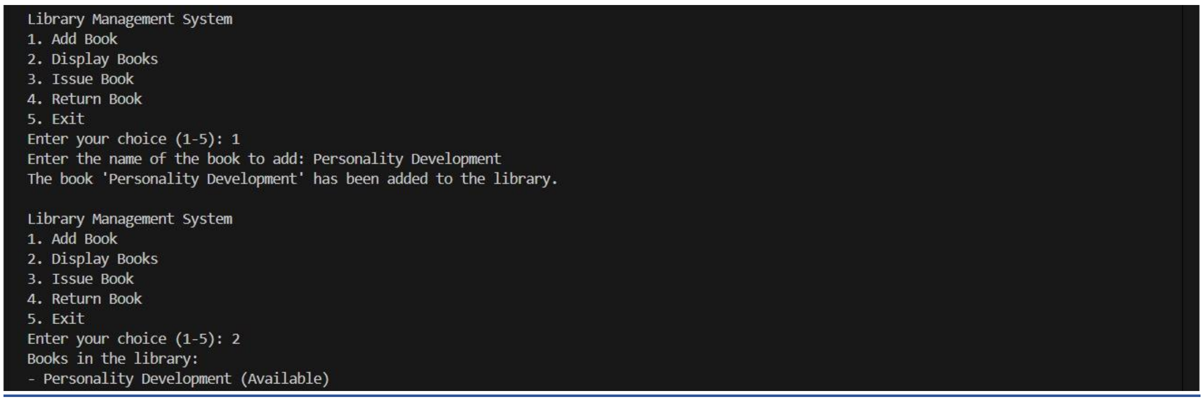


```python
            print(f"The book '{book_name}' is already issued.")
        else:
            self.books[book_name] = False  # Mark the book as issued
            print(f"You have issued the book '{book_name}'.")

    def return_book(self, book_name):
        """Return a book to the library."""
        if book_name not in self.books:
            print(f"The book '{book_name}' does not belong to this library.")
        elif self.books[book_name]:
            print(f"The book '{book_name}' was not issued.")
        else:
            self.books[book_name] = True  # Mark the book as available
            print(f"You have returned the book '{book_name}'.")
def main():
    """Main function to run the library management system."""
    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: ").strip()
            if book_name:
                library.add_book(book_name)
            else:
                print("Error: Book name cannot be empty.")
```



```python
                print("Error: Book name cannot be empty.")

        elif choice == '2':
            library.display_books()

        elif choice == '3':
            book_name = input("Enter the name of the book to issue: ").strip()
            if book_name:
                library.issue_book(book_name)
            else:
                print("Error: Book name cannot be empty.")

        elif choice == '4':
            book_name = input("Enter the name of the book to return: ").strip()
            if book_name:
                library.return_book(book_name)
            else:
                print("Error: Book name cannot be empty.")

        elif choice == '5':
            print("Exiting the Library Management System. Goodbye!")
            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: Personality Development
The book 'Personality Development' has been added to the library.

Library Management System
1. Add Book
2. Display Books
3. Issue Book
4. Return Book
5. Exit
Enter your choice (1-5): 2
Books in the library:
- Personality Development (Available)
```

Explanation:

The AI-generated Python program was developed to implement a simple library management system using a class-based design, along with loops and conditional logic to support menu-driven operations. The code was analyzed to understand how object-oriented concepts—classes, methods, and instance variables—manage book records and track availability. Each operation, including adding, displaying, issuing, and returning books, was validated to ensure correct behavior and to prevent invalid actions such as issuing an already issued book. The program was refactored for improved readability and maintainability through descriptive method names, structured control flow, and clear input validation. Instead of accepting the AI output as-is, the logic was verified, errors were corrected, and the implementation was fully understood to ensure reliability.

- (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.”

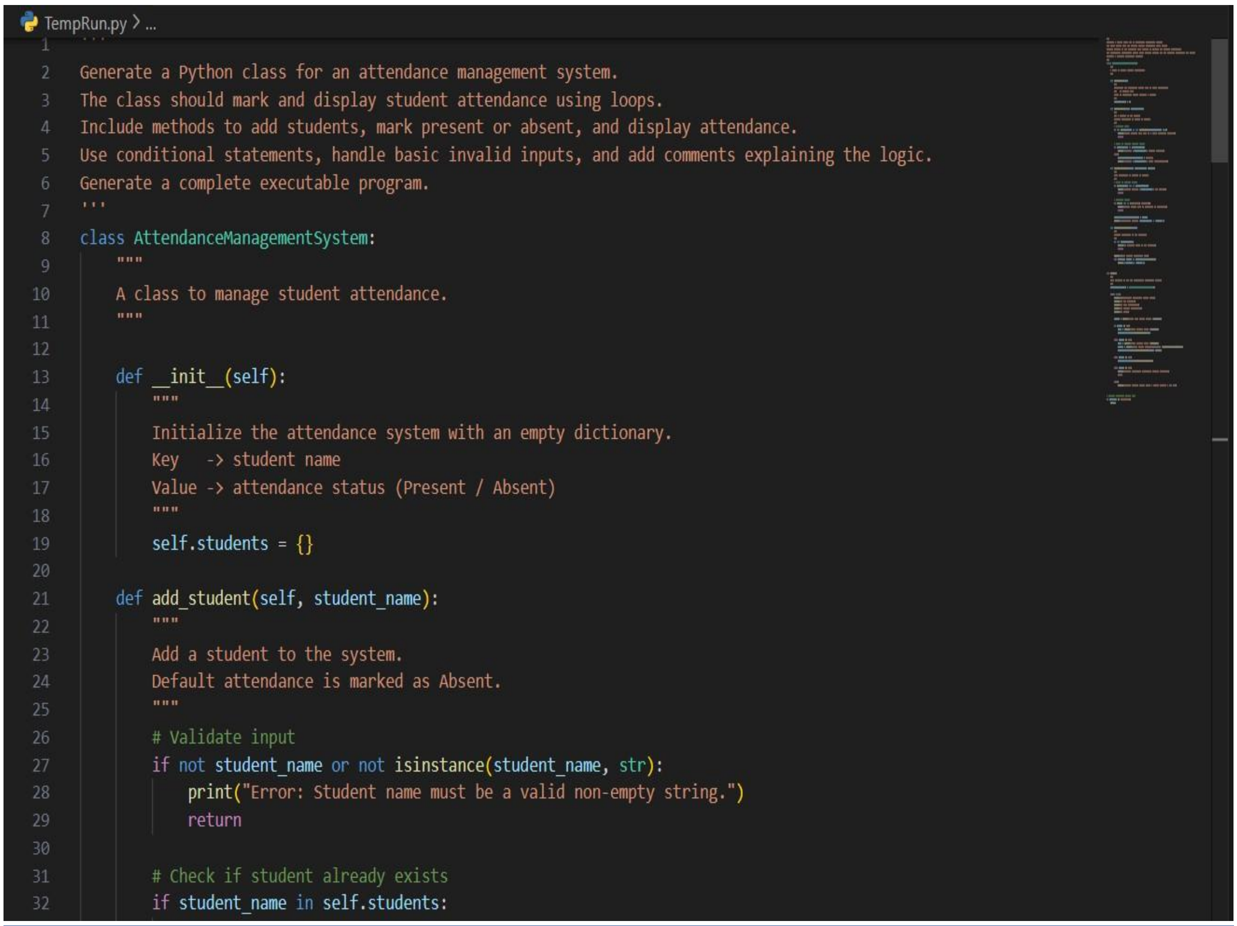
Expected Output:

• AI-generated attendance logic.

• Correct display of attendance.

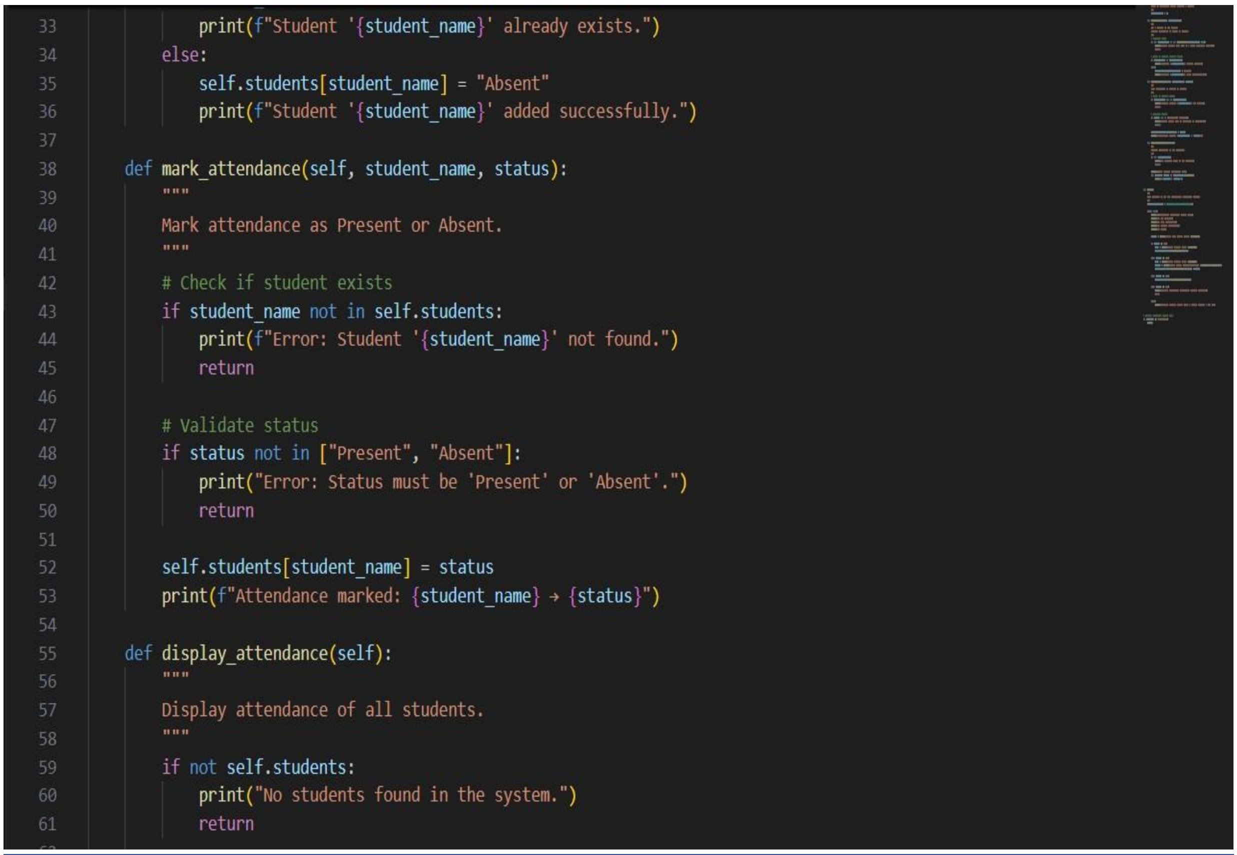
• Test cases

Code:

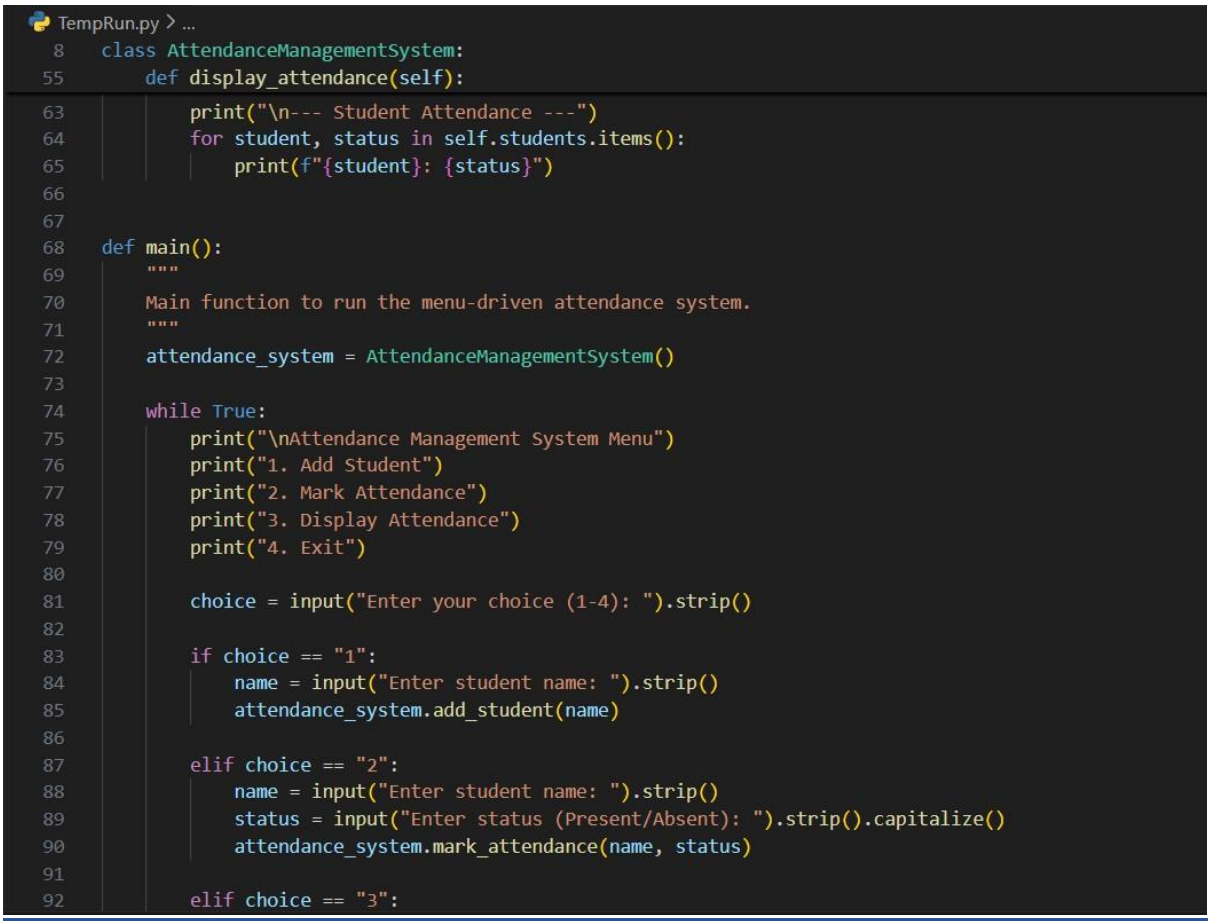


TempRun.py > ...

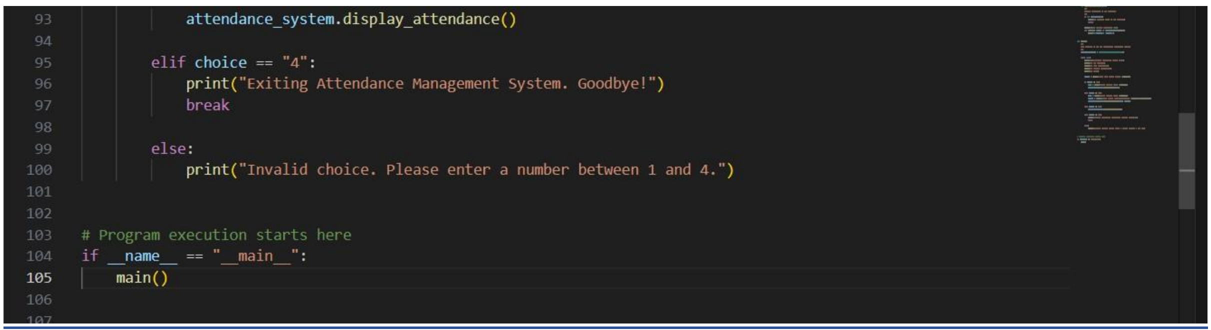
```python
'''
Generate a Python class for an attendance management system.
The class should mark and display student attendance using loops.
Include methods to add students, mark present or absent, and display attendance.
Use conditional statements, handle basic invalid inputs, and add comments explaining the logic.
Generate a complete executable program.
'''
class AttendanceManagementSystem:
    """
    A class to manage student attendance.
    """

    def __init__(self):
        """
        Initialize the attendance system with an empty dictionary.
        Key   -> student name
        Value -> attendance status (Present / Absent)
        """
        self.students = {}

    def add_student(self, student_name):
        """
        Add a student to the system.
        Default attendance is marked as Absent.
        """
        # Validate input
        if not student_name or not isinstance(student_name, str):
            print("Error: Student name must be a valid non-empty string.")
            return

        # Check if student already exists
        if student_name in self.students:
```



```python
            print(f"Student '{student_name}' already exists.")
        else:
            self.students[student_name] = "Absent"
            print(f"Student '{student_name}' added successfully.")

    def mark_attendance(self, student_name, status):
        """
        Mark attendance as Present or Absent.
        """
        # Check if student exists
        if student_name not in self.students:
            print(f"Error: Student '{student_name}' not found.")
            return

        # Validate status
        if status not in ["Present", "Absent"]:
            print("Error: Status must be 'Present' or 'Absent'.")
            return

        self.students[student_name] = status
        print(f"Attendance marked: {student_name} → {status}")

    def display_attendance(self):
        """
        Display attendance of all students.
        """
        if not self.students:
            print("No students found in the system.")
            return
```

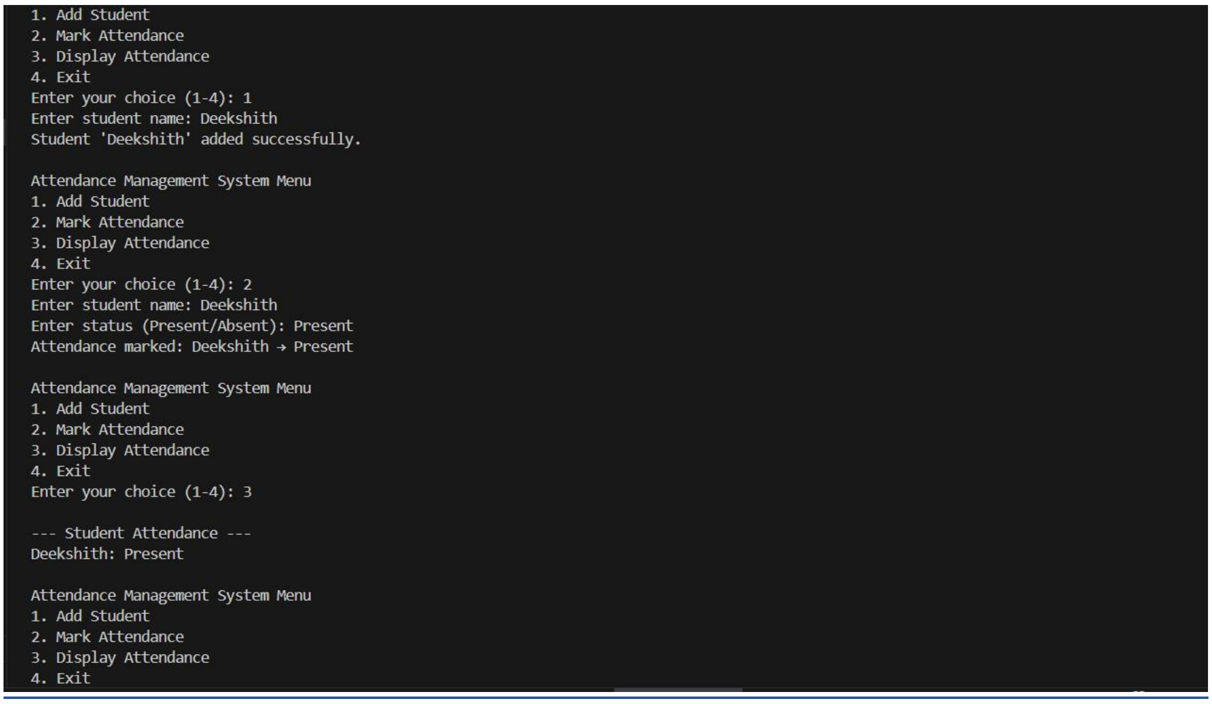


```python
TempRun.py > ...
class AttendanceManagementSystem:
    def display_attendance(self):
        print("\n--- Student Attendance ---")
        for student, status in self.students.items():
            print(f"{student}: {status}")


def main():
    """
    Main function to run the menu-driven attendance system.
    """
    attendance_system = AttendanceManagementSystem()

    while True:
        print("\nAttendance Management System Menu")
        print("1. Add Student")
        print("2. Mark Attendance")
        print("3. Display Attendance")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ").strip()

        if choice == "1":
            name = input("Enter student name: ").strip()
            attendance_system.add_student(name)

        elif choice == "2":
            name = input("Enter student name: ").strip()
            status = input("Enter status (Present/Absent): ").strip().capitalize()
            attendance_system.mark_attendance(name, status)

        elif choice == "3":
```



```python
            attendance_system.display_attendance()

        elif choice == "4":
            print("Exiting Attendance Management System. Goodbye!")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 4.")


# Program execution starts here
if __name__ == "__main__":
    main()
```

Output:



```
1. Add Student
2. Mark Attendance
3. Display Attendance
4. Exit
Enter your choice (1-4): 1
Enter student name: Deekshith
Student 'Deekshith' added successfully.

Attendance Management System Menu
1. Add Student
2. Mark Attendance
3. Display Attendance
4. Exit
Enter your choice (1-4): 2
Enter student name: Deekshith
Enter status (Present/Absent): Present
Attendance marked: Deekshith → Present

Attendance Management System Menu
1. Add Student
2. Mark Attendance
3. Display Attendance
4. Exit
Enter your choice (1-4): 3

--- Student Attendance ---
Deekshith: Present

Attendance Management System Menu
1. Add Student
2. Mark Attendance
3. Display Attendance
4. Exit
```

Explanation:

The AI-generated Python program was developed to implement an attendance management system using a class-based design, with loops and conditional logic to support menu-driven operations. The structure was analyzed to understand how methods add students, record attendance as present or absent, and store records using dictionaries for efficient access. Input validation was strengthened to handle edge cases such as empty names, invalid attendance values, and non-existent students. The code was then refactored to improve clarity and maintainability through descriptive method names, organized control flow, and concise comments. Rather than relying blindly on the AI output, the logic was verified, corrected where necessary, and fully understood to ensure accuracy and reliability.

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.”

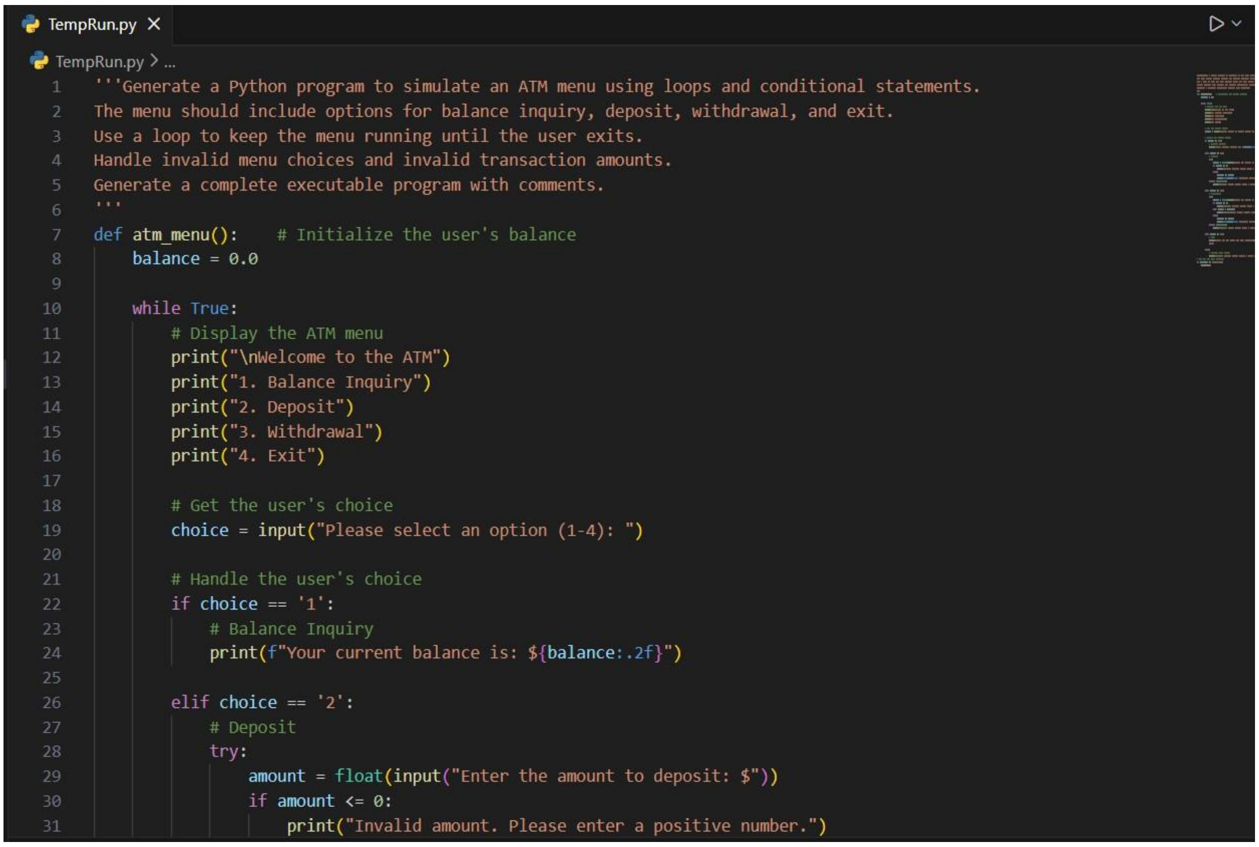
Expected Output:

• AI-generated menu logic.

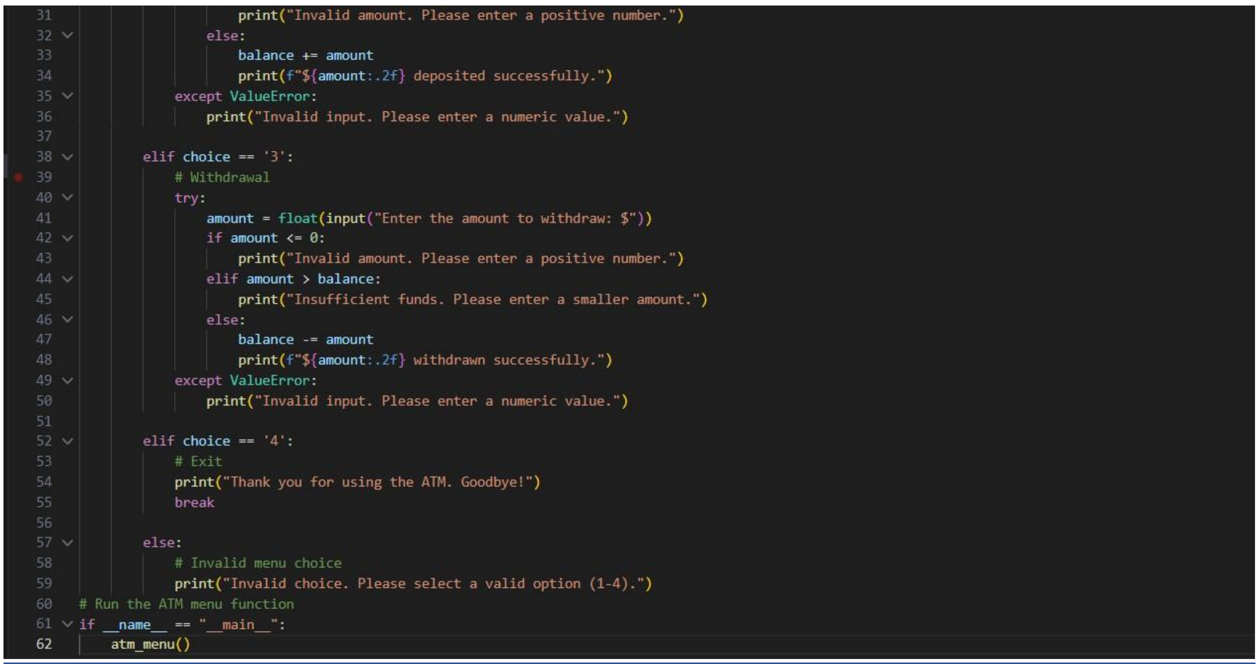
• Correct option handling.

• Output verification.

Code:

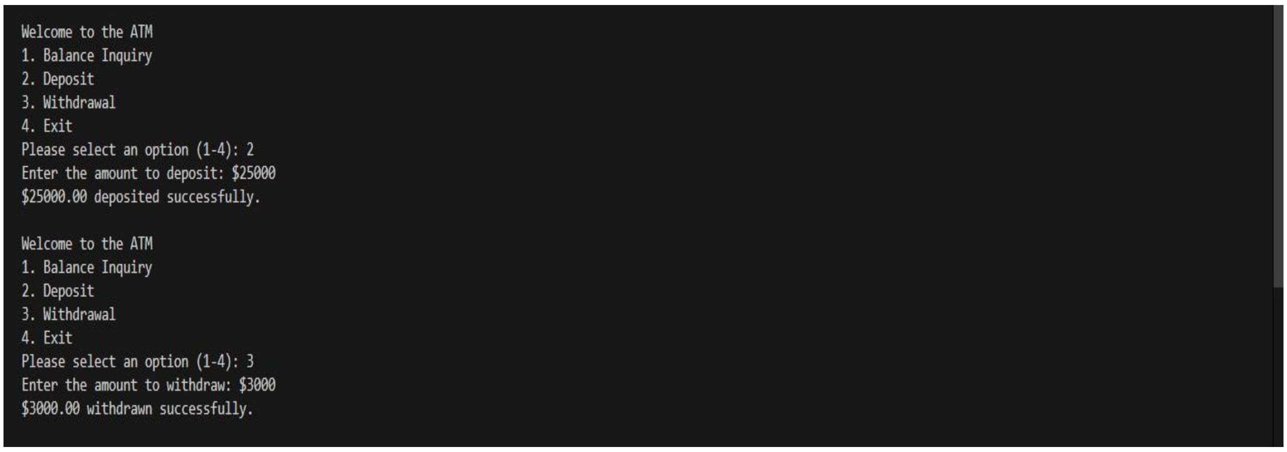

TempRun.py

```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():    # Initialize the user's balance
    balance = 0.0

    while True:
        # Display the ATM menu
        print("\nWelcome to the ATM")
        print("1. Balance Inquiry")
        print("2. Deposit")
        print("3. Withdrawal")
        print("4. Exit")

        # Get the user's choice
        choice = input("Please select an option (1-4): ")

        # Handle the user's choice
        if choice == '1':
            # Balance Inquiry
            print(f"Your current balance is: ${balance:.2f}")

        elif choice == '2':
            # Deposit
            try:
                amount = float(input("Enter the amount to deposit: $"))
                if amount <= 0:
                    print("Invalid amount. Please enter a positive number.")
```



```python
                    print("Invalid amount. Please enter a positive number.")
                else:
                    balance += amount
                    print(f"${amount:.2f} deposited successfully.")
            except ValueError:
                print("Invalid input. Please enter a numeric value.")

        elif choice == '3':
            # Withdrawal
            try:
                amount = float(input("Enter the amount to withdraw: $"))
                if amount <= 0:
                    print("Invalid amount. Please enter a positive number.")
                elif amount > balance:
                    print("Insufficient funds. Please enter a smaller amount.")
                else:
                    balance -= amount
                    print(f"${amount:.2f} withdrawn successfully.")
            except ValueError:
                print("Invalid input. Please enter a numeric value.")

        elif choice == '4':
            # Exit
            print("Thank you for using the ATM. Goodbye!")
            break

        else:
            # Invalid menu choice
            print("Invalid choice. Please select a valid option (1-4).")
# Run the ATM menu function
if __name__ == "__main__":
    atm_menu()
```

Output:



```
Welcome to the ATM
1. Balance Inquiry
2. Deposit
3. Withdrawal
4. Exit
Please select an option (1-4): 2
Enter the amount to deposit: $25000
$25000.00 deposited successfully.

Welcome to the ATM
1. Balance Inquiry
2. Deposit
3. Withdrawal
4. Exit
Please select an option (1-4): 3
Enter the amount to withdraw: $3000
$3000.00 withdrawn successfully.
```

Explanation:

The AI-generated Python program implemented an attendance management system using a class-based architecture, with loops and conditional logic to drive menu-based operations. Methods were designed to add students, mark attendance, and maintain records using dictionaries for efficient storage and retrieval. Robust input validation was added to handle edge cases such as empty names, invalid status values, and non-existent students. The code was refactored to enhance readability and maintainability through descriptive method names, structured control flow, and concise documentation. The generated logic was independently verified and corrected where necessary to ensure accuracy and dependable behavior