AI Assisted Coding

Assignment – 6.5

Name: J.Akshaya

Batch: 22

HtNo: 2303A51629

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.”

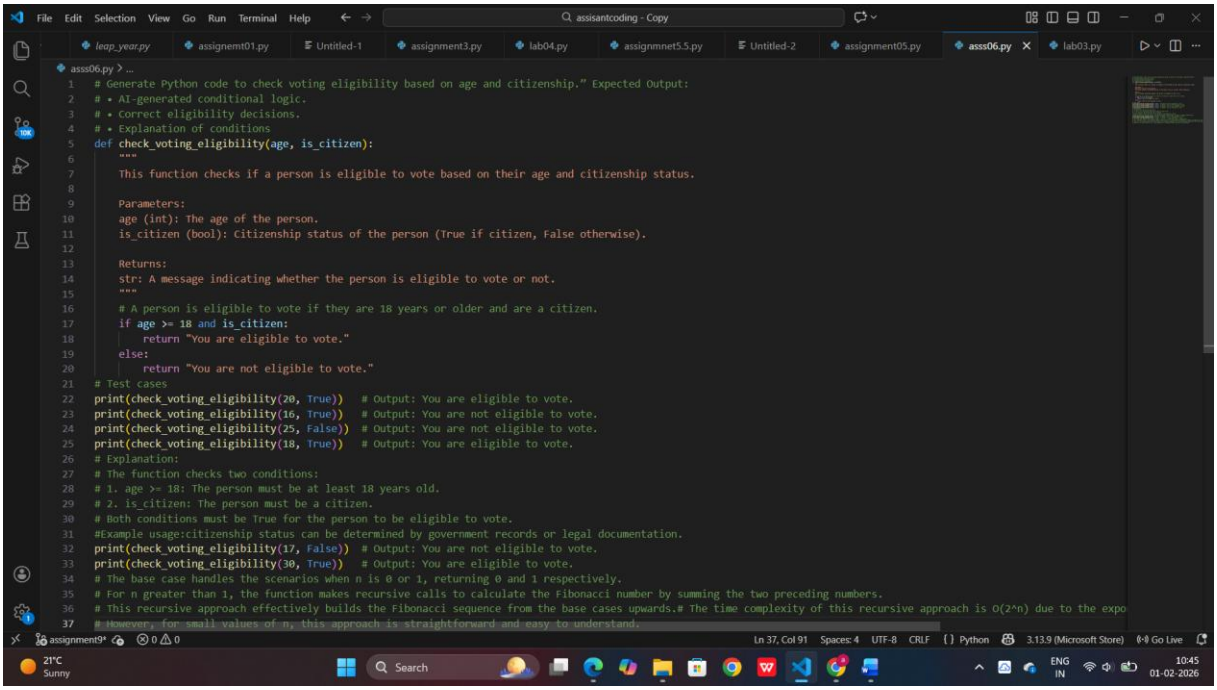
Expected Output:

• AI-generated conditional logic.

• Correct eligibility decisions.

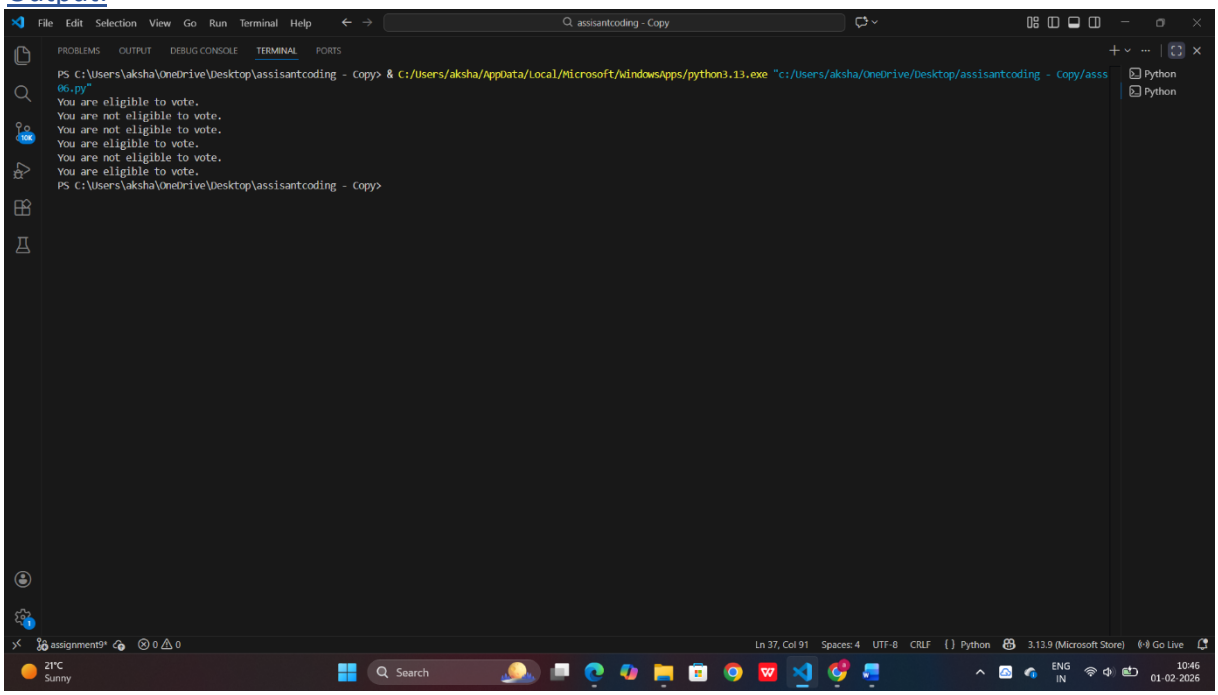
• Explanation of conditions.

Code:



```python

    # 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:



Explanation:

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.

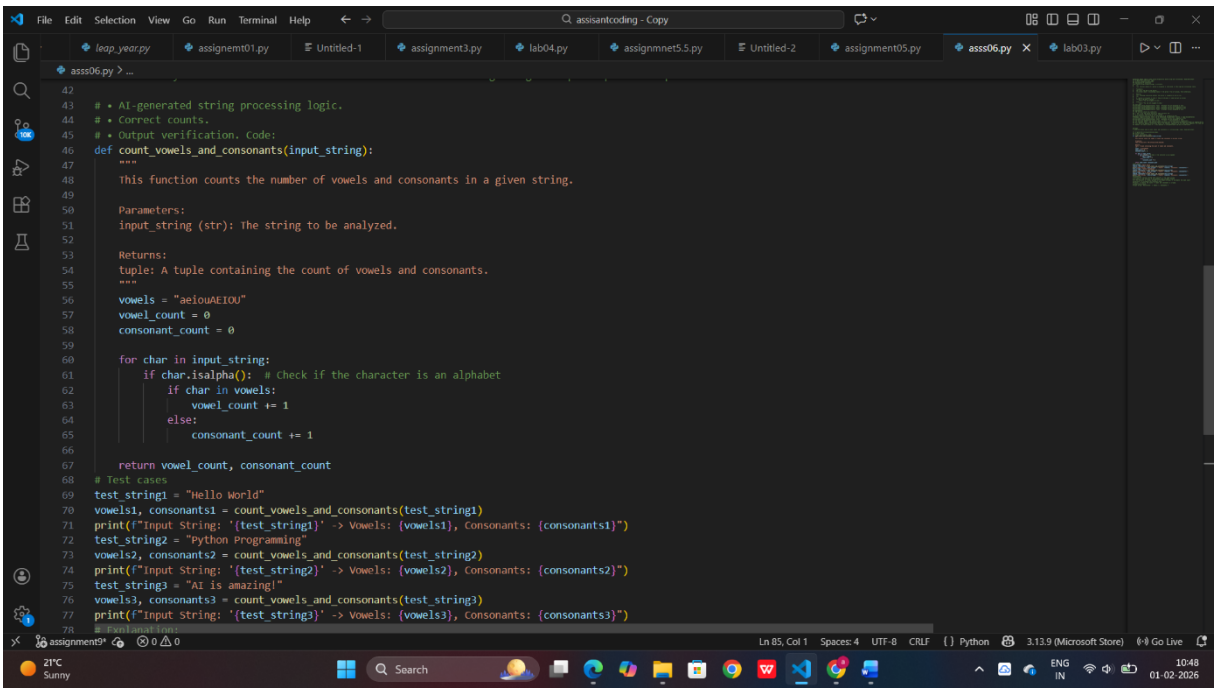
Prompt:

“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.

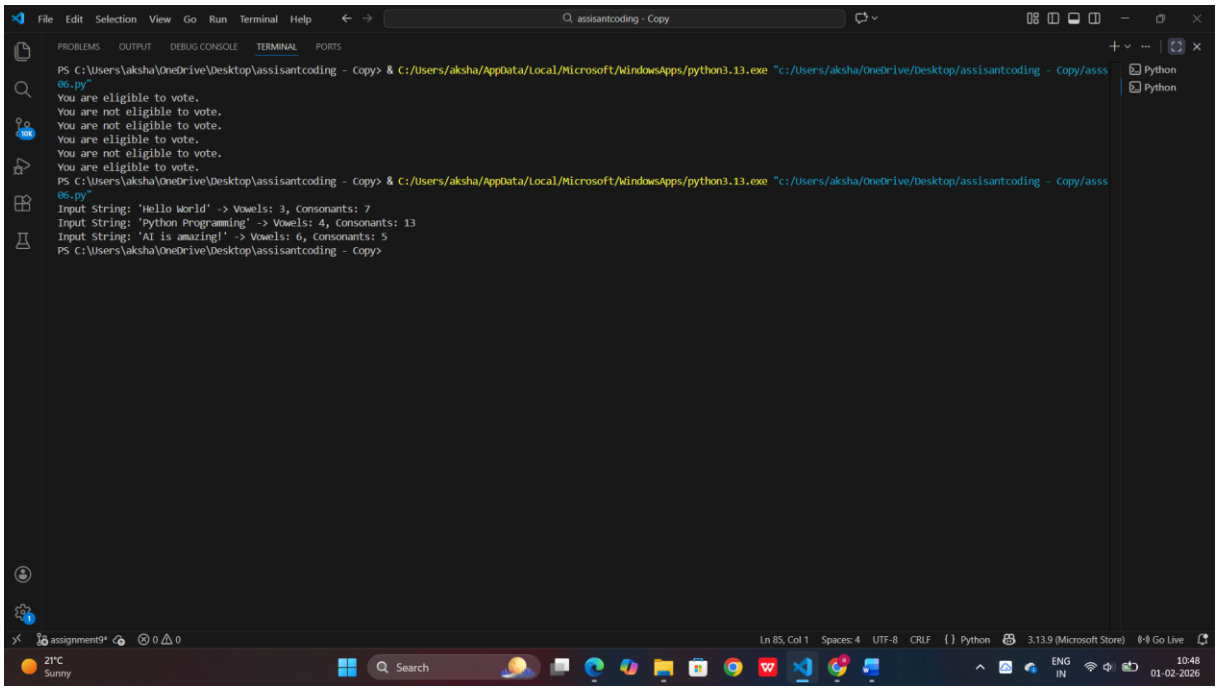
Code:



```
# • AI-generated string processing logic.
# • Correct counts.
# • Output verification. Code:
def count_vowels_and_consonants(input_string):
    """
    This function counts the number of vowels and consonants in a given string.

    Parameters:
    input_string (str): The string to be analyzed.

    Returns:
    tuple: A tuple containing the count of vowels and consonants.
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

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

    return vowel_count, consonant_count
# Test cases
test_string1 = "Hello World"
vowels1, consonants1 = count_vowels_and_consonants(test_string1)
print(f"Input String: '{test_string1}' -> Vowels: {vowels1}, Consonants: {consonants1}")
test_string2 = "Python Programming"
vowels2, consonants2 = count_vowels_and_consonants(test_string2)
print(f"Input String: '{test_string2}' -> Vowels: {vowels2}, Consonants: {consonants2}")
test_string3 = "AI is amazing!"
vowels3, consonants3 = count_vowels_and_consonants(test_string3)
print(f"Input String: '{test_string3}' -> Vowels: {vowels3}, Consonants: {consonants3}")
```

```
                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:


File Edit Selection View Go Run Terminal Help
assisantcoding - Copy
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/asss
06.py"
You are eligible to vote.
You are not eligible to vote.
You are not eligible to vote.
You are eligible to vote.
You are not eligible to vote.
You are eligible to vote.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/asss
06.py"
Input String: 'Hello World' -> Vowels: 3, Consonants: 7
Input String: 'Python Programming' -> Vowels: 4, Consonants: 13
Input String: 'AI is amazing!' -> Vowels: 6, Consonants: 5
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
Python
Python
assignment9*
Ln 85, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.9 (Microsoft Store) Go Live
21°C
Sunny
Search
ENG
IN
10:48
01-02-2026

Explanation :

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_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.”

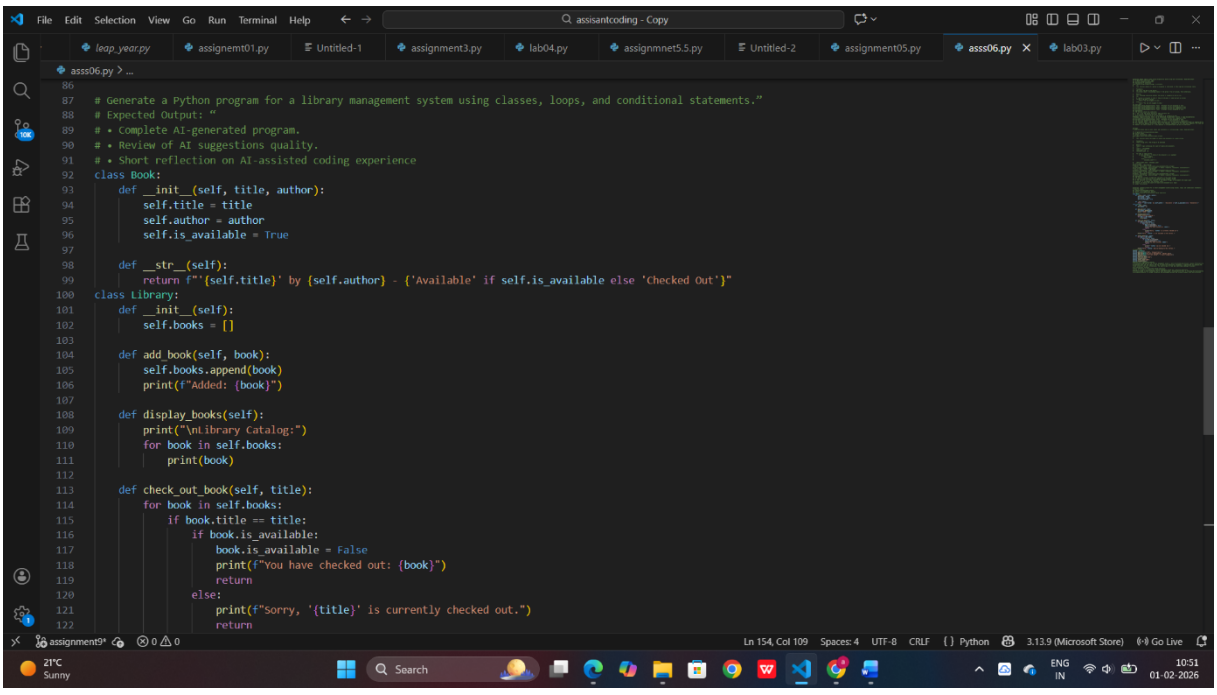
Expected Output: “

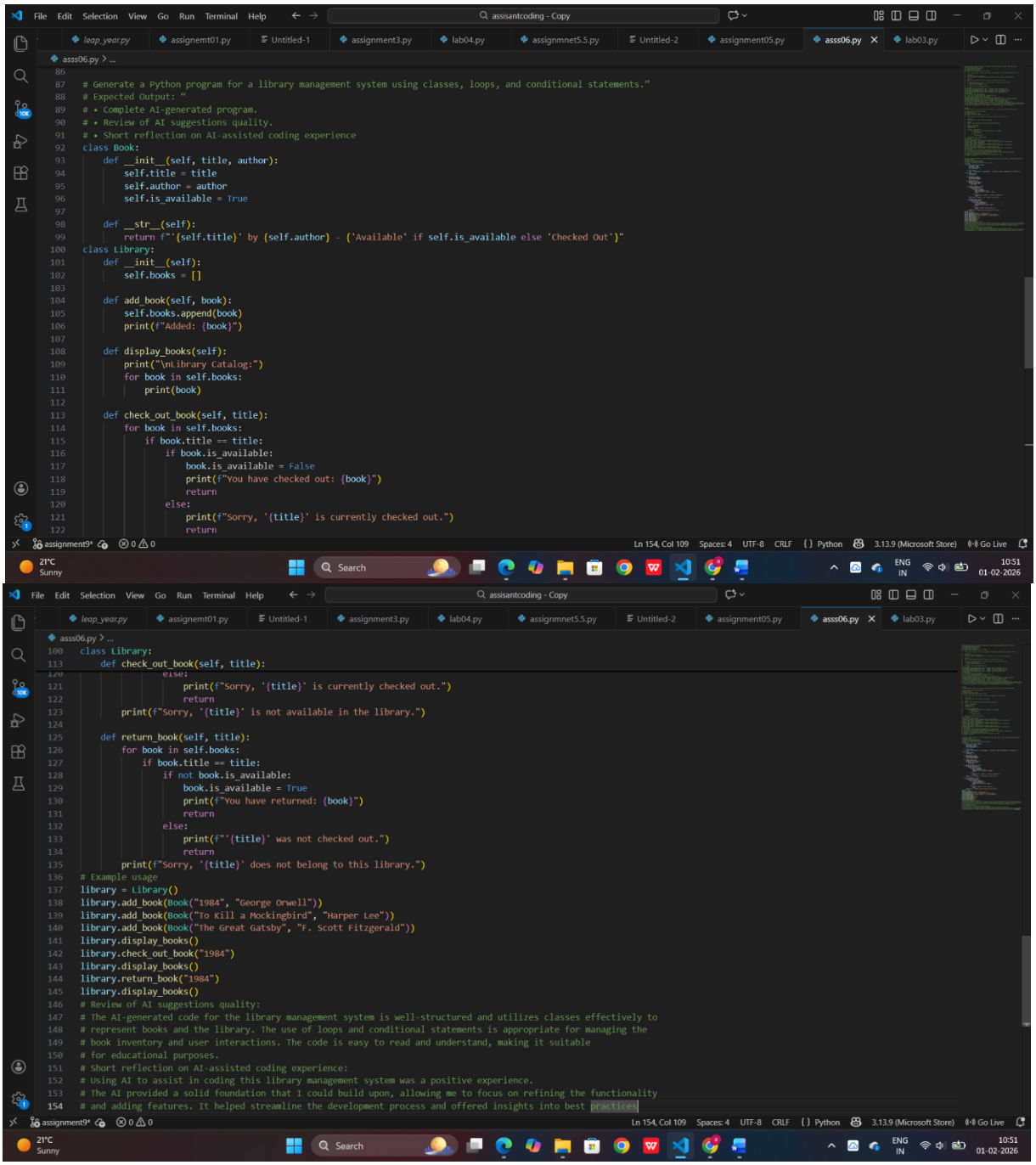
• Complete AI-generated program.

• Review of AI suggestions quality.

• Short reflection on AI-assisted coding experience

Code:



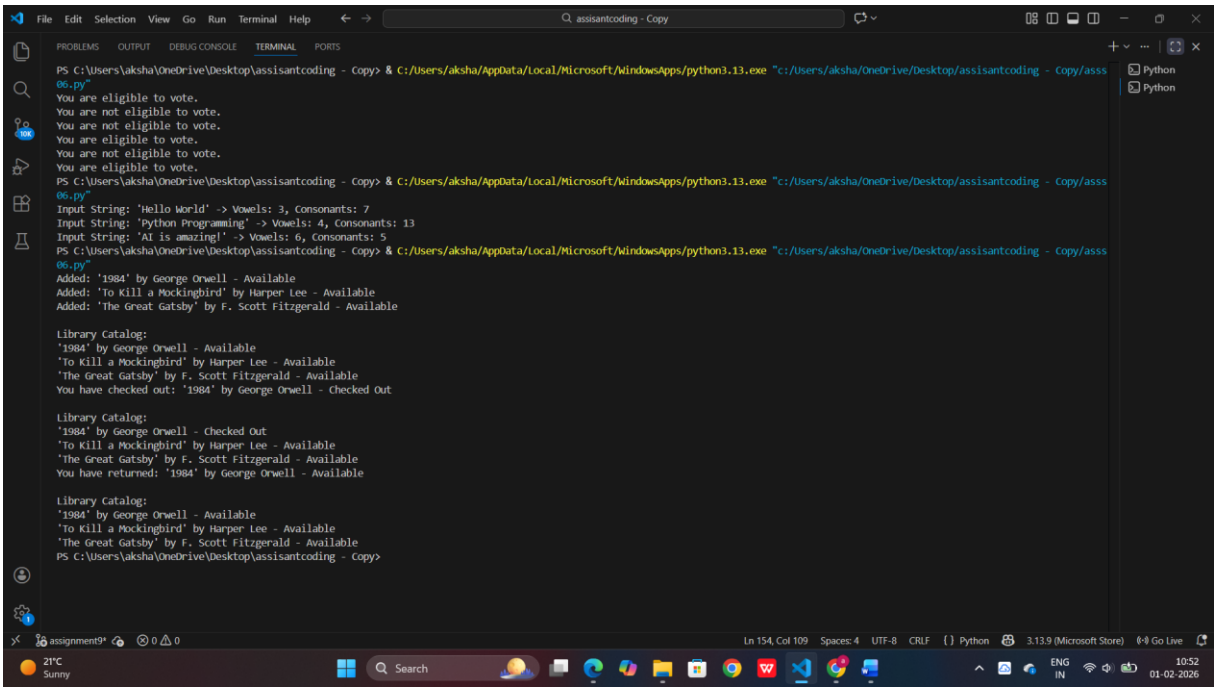


```python
# Generate a Python program for a library management system using classes, loops, and conditional statements."
# Expected Output: "
# • Complete AI-generated program.
# • Review of AI suggestions quality.
# • Short reflection on AI-assisted coding experience
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_available = True

    def __str__(self):
        return f"'{self.title}' by {self.author} - {'Available' if self.is_available else 'Checked Out'}"
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)
        print(f"Added: {book}")

    def display_books(self):
        print("\nLibrary Catalog:")
        for book in self.books:
            print(book)

    def check_out_book(self, title):
        for book in self.books:
            if book.title == title:
                if book.is_available:
                    book.is_available = False
                    print(f"You have checked out: {book}")
                    return
                else:
                    print(f"Sorry, '{title}' is currently checked out.")
                    return
```



```python
class Library:
    def check_out_book(self, title):
                else:
                    print(f"Sorry, '{title}' is currently checked out.")
                    return
        print(f"Sorry, '{title}' is not available in the library.")

    def return_book(self, title):
        for book in self.books:
            if book.title == title:
                if not book.is_available:
                    book.is_available = True
                    print(f"You have returned: {book}")
                    return
                else:
                    print(f"'{title}' was not checked out.")
                    return
        print(f"Sorry, '{title}' does not belong to this library.")
# Example usage
library = Library()
library.add_book(Book("1984", "George Orwell"))
library.add_book(Book("To Kill a Mockingbird", "Harper Lee"))
library.add_book(Book("The Great Gatsby", "F. Scott Fitzgerald"))
library.display_books()
library.check_out_book("1984")
library.display_books()
library.return_book("1984")
library.display_books()
# Review of AI suggestions quality:
# The AI-generated code for the library management system is well-structured and utilizes classes effectively to
# represent books and the library. The use of loops and conditional statements is appropriate for managing the
# book inventory and user interactions. The code is easy to read and understand, making it suitable
# for educational purposes.
# Short reflection on AI-assisted coding experience:
# Using AI to assist in coding this library management system was a positive experience.
# The AI provided a solid foundation that I could build upon, allowing me to focus on refining the functionality
# and adding features. It helped streamline the development process and offered insights into best practices
```

Output:



Explanation:

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-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:

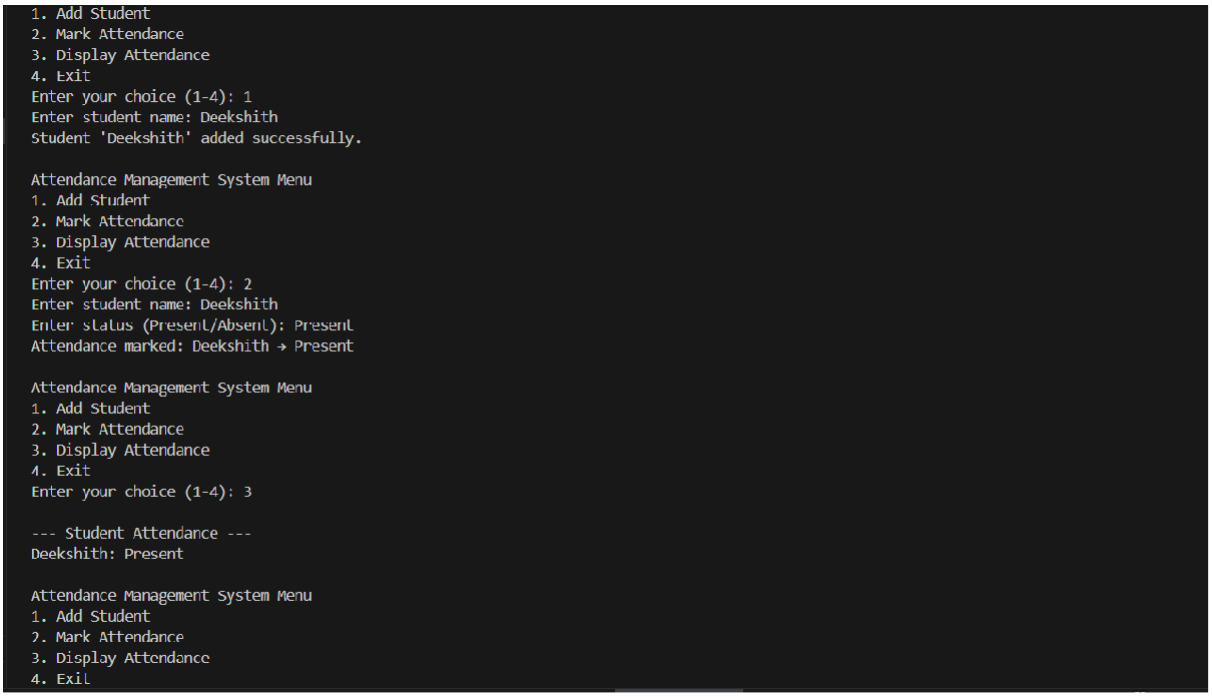
TempRun.py > ...

```
'''
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:
            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
```

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":
            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:



Explanation:

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."

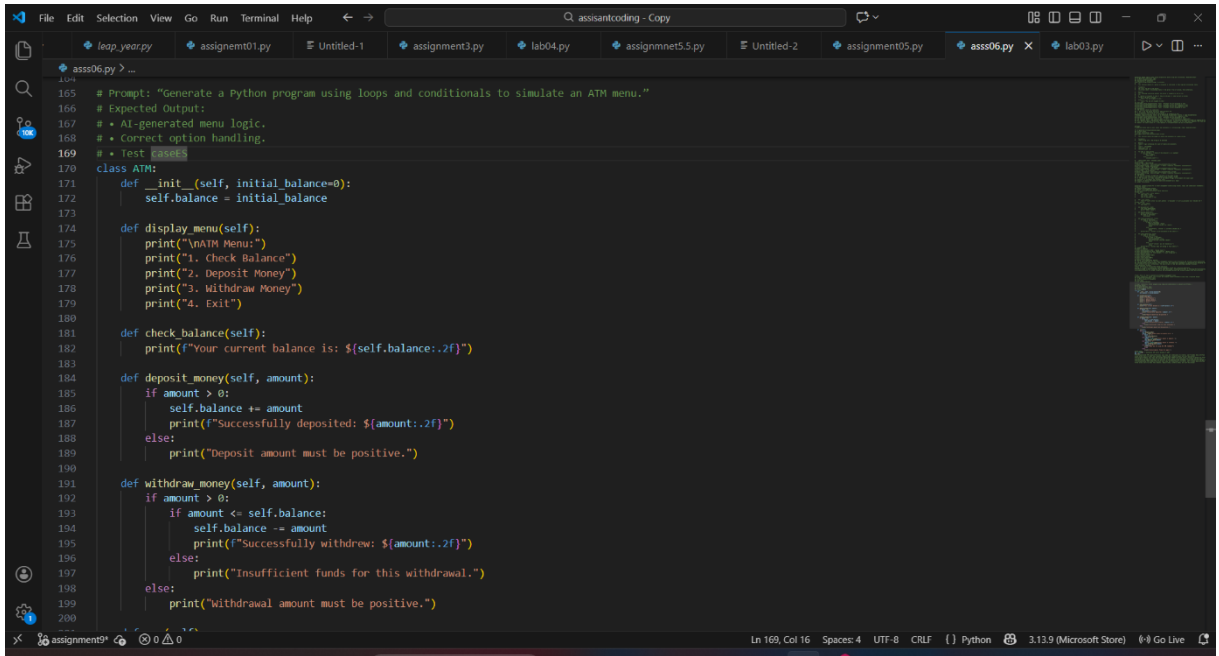
Expected Output:

• AI-generated menu logic.

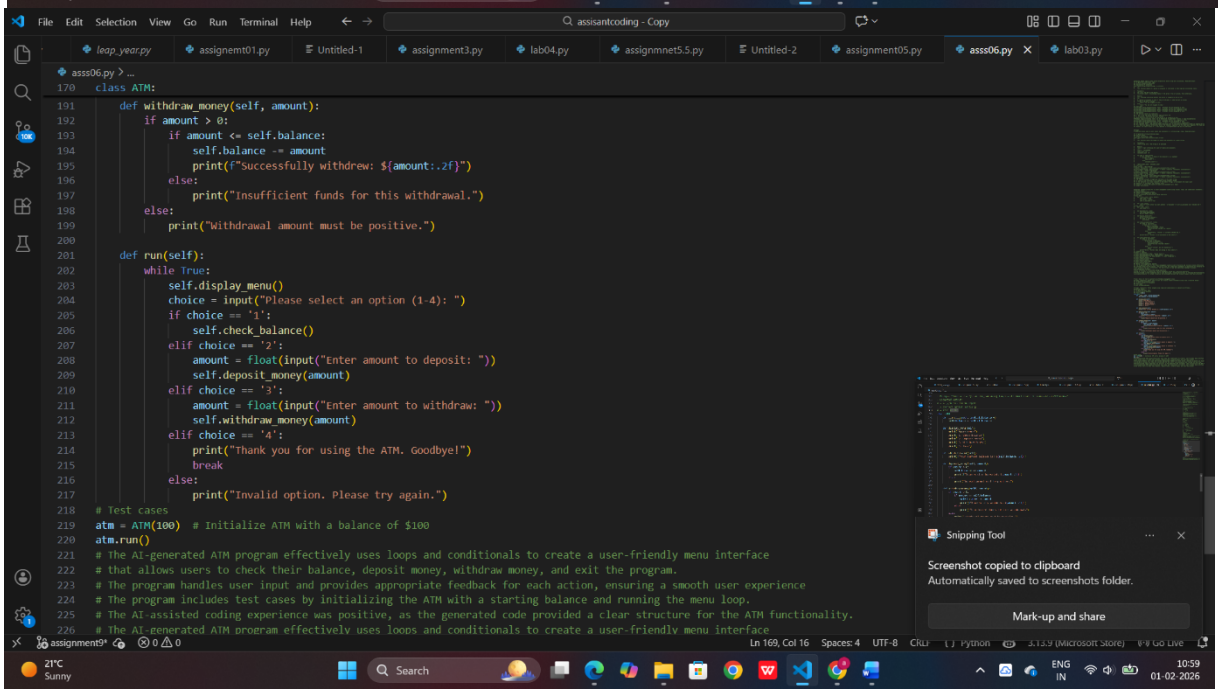
• Correct option handling.

- Output verification.

Code:



```python
# Prompt: "Generate a Python program using loops and conditionals to simulate an ATM menu."
# Expected Output:
# • AI-generated menu logic.
# • Correct option handling.
# • Test cases
class ATM:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def display_menu(self):
        print("\nATM Menu:")
        print("1. Check Balance")
        print("2. Deposit Money")
        print("3. Withdraw Money")
        print("4. Exit")

    def check_balance(self):
        print(f"Your current balance is: ${self.balance:.2f}")

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

    def withdraw_money(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Successfully withdrew: ${amount:.2f}")
            else:
                print("Insufficient funds for this withdrawal.")
        else:
            print("Withdrawal amount must be positive.")
```



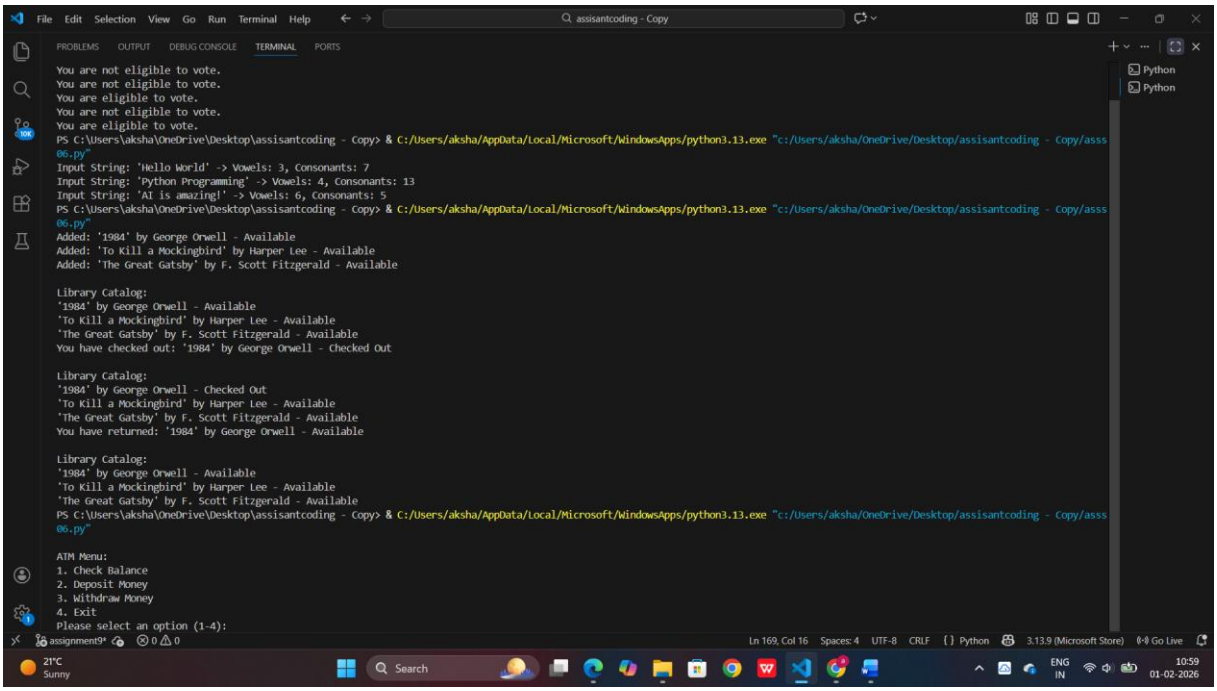
```python
class ATM:
    def withdraw_money(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Successfully withdrew: ${amount:.2f}")
            else:
                print("Insufficient funds for this withdrawal.")
        else:
            print("Withdrawal amount must be positive.")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Please select an option (1-4): ")
            if choice == '1':
                self.check_balance()
            elif choice == '2':
                amount = float(input("Enter amount to deposit: "))
                self.deposit_money(amount)
            elif choice == '3':
                amount = float(input("Enter amount to withdraw: "))
                self.withdraw_money(amount)
            elif choice == '4':
                print("Thank you for using the ATM. Goodbye!")
                break
            else:
                print("Invalid option. Please try again.")
# Test cases
atm = ATM(100)  # Initialize ATM with a balance of $100
atm.run()
# The AI-generated ATM program effectively uses loops and conditionals to create a user-friendly menu interface
# that allows users to check their balance, deposit money, withdraw money, and exit the program.
# The program handles user input and provides appropriate feedback for each action, ensuring a smooth user experience
# The program includes test cases by initializing the ATM with a starting balance and running the menu loop.
# The AI-assisted coding experience was positive, as the generated code provided a clear structure for the ATM functionality.
# The AI-generated ATM program effectively uses loops and conditionals to create a user-friendly menu interface
```

Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Mark-up and share

Output:



Explanation:

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.