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

Hall Ticket No:2303A51734

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

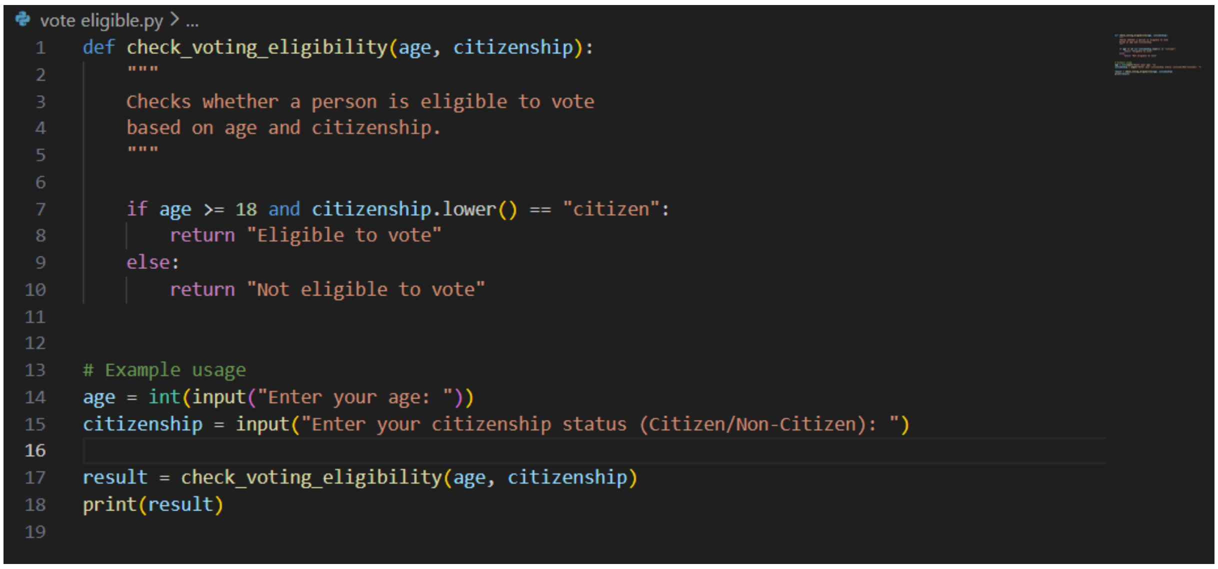
Task: Use an AI tool to generate eligibility logic.

Prompt:

#Generate a python code that to check voting eligibility based on age and citizenship status.

#include necessary comments to explain the code and correct eligibility criteria

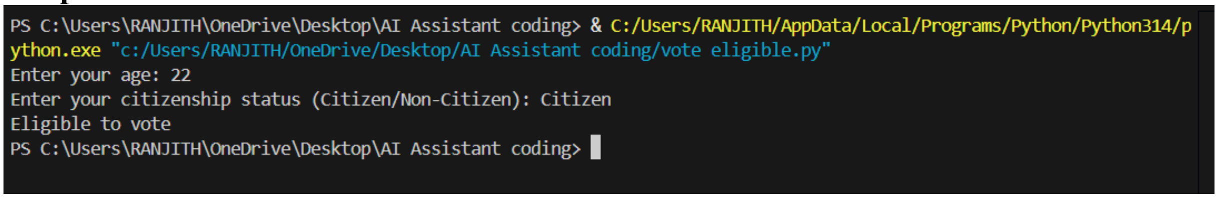
Code:



vote eligible.py > ...

```python
def check_voting_eligibility(age, citizenship):
    """
    Checks whether a person is eligible to vote
    based on age and citizenship.
    """

    if age >= 18 and citizenship.lower() == "citizen":
        return "Eligible to vote"
    else:
        return "Not eligible to vote"


# Example usage
age = int(input("Enter your age: "))
citizenship = input("Enter your citizenship status (Citizen/Non-Citizen): ")

result = check_voting_eligibility(age, citizenship)
print(result)
```

Output:



```
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding> & C:/Users/RANJITH/AppData/Local/Programs/Python/Python314/p
ython.exe "c:/Users/RANJITH/OneDrive/Desktop/AI Assistant coding/vote eligible.py"
Enter your age: 22
Enter your citizenship status (Citizen/Non-Citizen): Citizen
Eligible to vote
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding>
```

Observation:

1. The code correctly applies voting eligibility rules by checking if the age is 18 or above and verifying citizenship status. This ensures the decision follows standard legal voting requirements.

2. A separate function is used to check eligibility, which makes the code modular and reusable.This improves readability and allows easy testing or future updates.

3. The function includes a detailed docstring explaining its purpose, inputs, and output. This helps other developers understand and maintain the code easily.

4. User input for citizenship is processed using strip() and lower() to avoid case-related errors. This makes the program more robust and user-friendly.

5. Boolean logic is used effectively to evaluate eligibility in a single conditional statement. This keeps the logic simple, clear, and efficient.

6. The program provides clear messages indicating whether the user is eligible to vote or not. This improves the overall user experience and clarity of results.

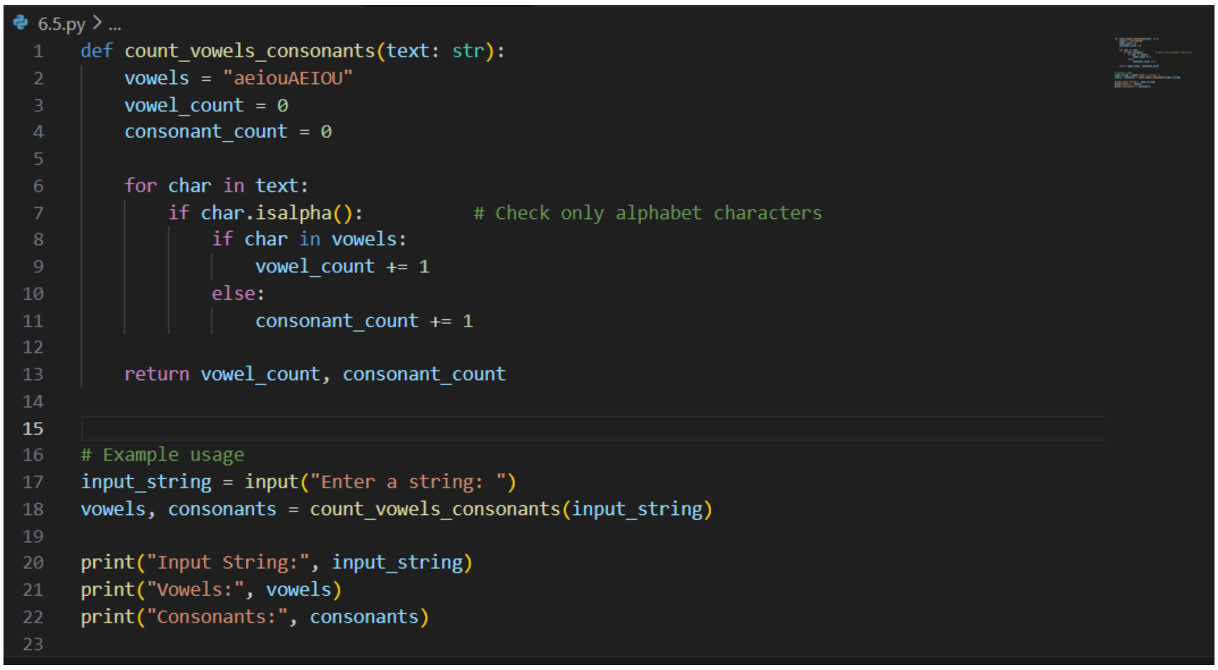
Task Description #2(AI-Based Code Completion for Loop-Based String Processing)

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

Prompt:

#Generate a python code to count vowles and consonants in a given string using loops.

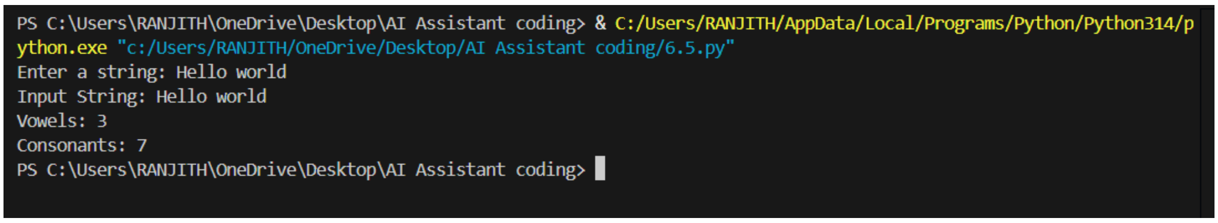
Code:



6.5.py > ...

```python
def count_vowels_consonants(text: str):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in text:
        if char.isalpha():          # Check only alphabet characters
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return vowel_count, consonant_count


# Example usage
input_string = input("Enter a string: ")
vowels, consonants = count_vowels_consonants(input_string)

print("Input String:", input_string)
print("Vowels:", vowels)
print("Consonants:", consonants)
```

Output:



```
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding> & C:/Users/RANJITH/AppData/Local/Programs/Python/Python314/p
ython.exe "c:/Users/RANJITH/OneDrive/Desktop/AI Assistant coding/6.5.py"
Enter a string: Hello world
Input String: Hello world
Vowels: 3
Consonants: 7
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding> 
```

Observation:

1. The program correctly identifies and counts vowels and consonants by iterating through each character using a loop. This ensures every character in the string is analyzed properly.

2. It checks only alphabetic characters using isalpha(), ignoring numbers and special symbols. This prevents incorrect counting and improves accuracy.

3. Vowels are clearly defined in a separate string containing both uppercase and lowercase letters. This allows the program to handle case-insensitive vowel detection.

4. The use of separate counters for vowels and consonants makes the logic easy to understand. This also helps in maintaining and debugging the code.

5. The function is modular and reusable, returning results as a tuple. This allows the function to be used in other programs without modification.

6. Clear input and output statements make the program user-friendly.
The final results are displayed in an understandable format for the user.

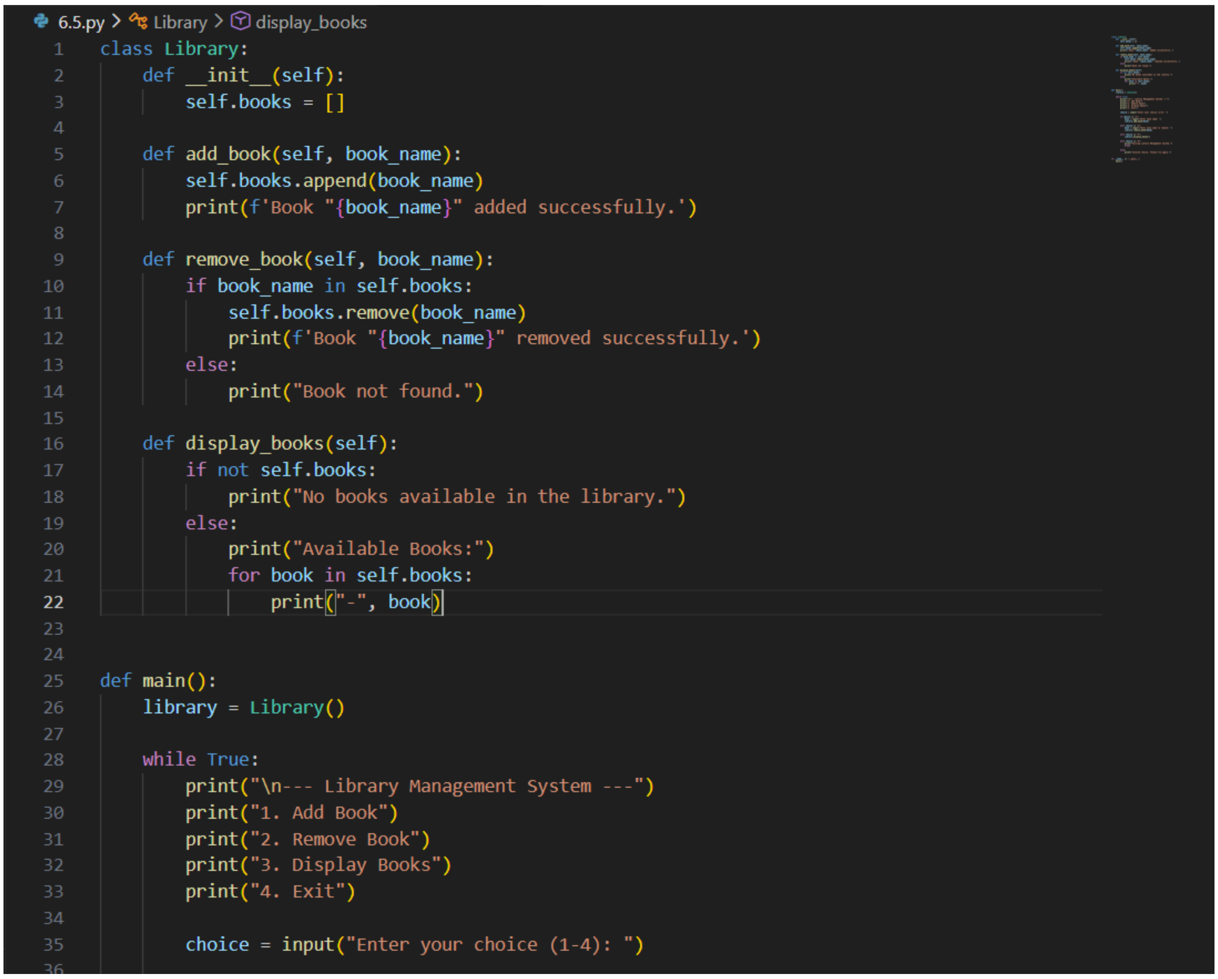
Task Description #3 (AI-Assisted Code Completion Reflection Task)

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

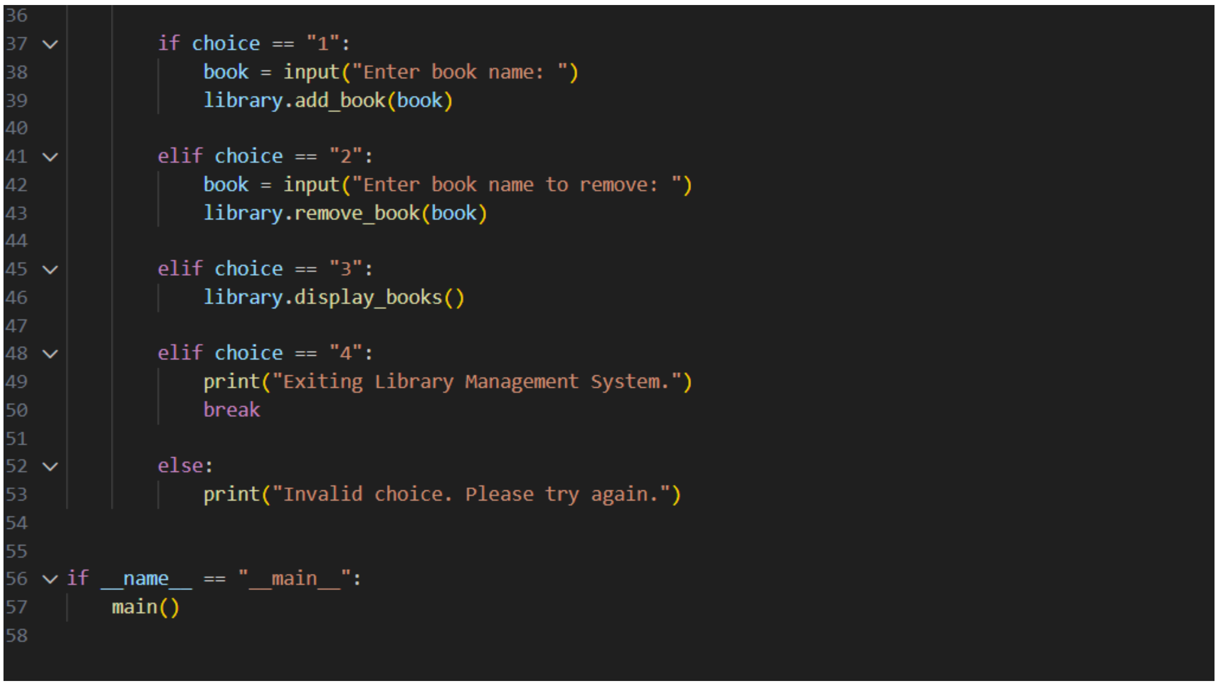
Prompt:

#Generate a python code to manage a library system using classes,loops and conditionlal statements.

Code:

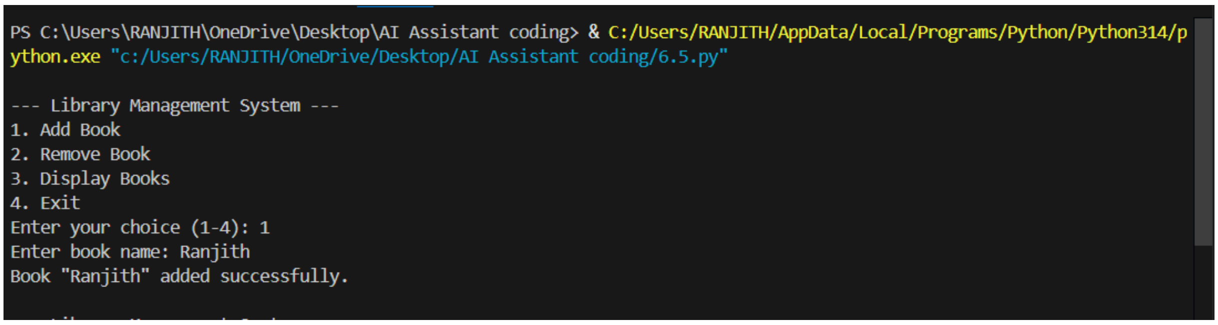

6.5.py > Library > display_books

```python
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book_name):
        self.books.append(book_name)
        print(f'Book "{book_name}" added successfully.')

    def remove_book(self, book_name):
        if book_name in self.books:
            self.books.remove(book_name)
            print(f'Book "{book_name}" removed successfully.')
        else:
            print("Book not found.")

    def display_books(self):
        if not self.books:
            print("No books available in the library.")
        else:
            print("Available Books:")
            for book in self.books:
                print("-", book)


def main():
    library = Library()

    while True:
        print("\n--- Library Management System ---")
        print("1. Add Book")
        print("2. Remove Book")
        print("3. Display Books")
        print("4. Exit")

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



```python

        if choice == "1":
            book = input("Enter book name: ")
            library.add_book(book)

        elif choice == "2":
            book = input("Enter book name to remove: ")
            library.remove_book(book)

        elif choice == "3":
            library.display_books()

        elif choice == "4":
            print("Exiting Library Management System.")
            break

        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    main()

```

Output:



```
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding> & C:/Users/RANJITH/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/RANJITH/OneDrive/Desktop/AI Assistant coding/6.5.py"

--- Library Management System ---
1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter your choice (1-4): 1
Enter book name: Ranjith
Book "Ranjith" added successfully.
```

Observation:

1. The program uses Object-Oriented Programming with Book and Library classes. This makes the code structured and easy to maintain.

2. Book availability is managed using a boolean variable.It ensures a book cannot be borrowed twice at the same time.

3. Borrow and return operations are implemented using separate methods. This improves clarity and reusability of the code.

4. The library searches books using their title. It allows users to easily borrow or return a specific book.

5. Proper messages are displayed for success and failure cases. This helps users understand the system's response clearly.

6. The program includes example usage for demonstration. It makes the working of the system easy to understand for beginners**.**

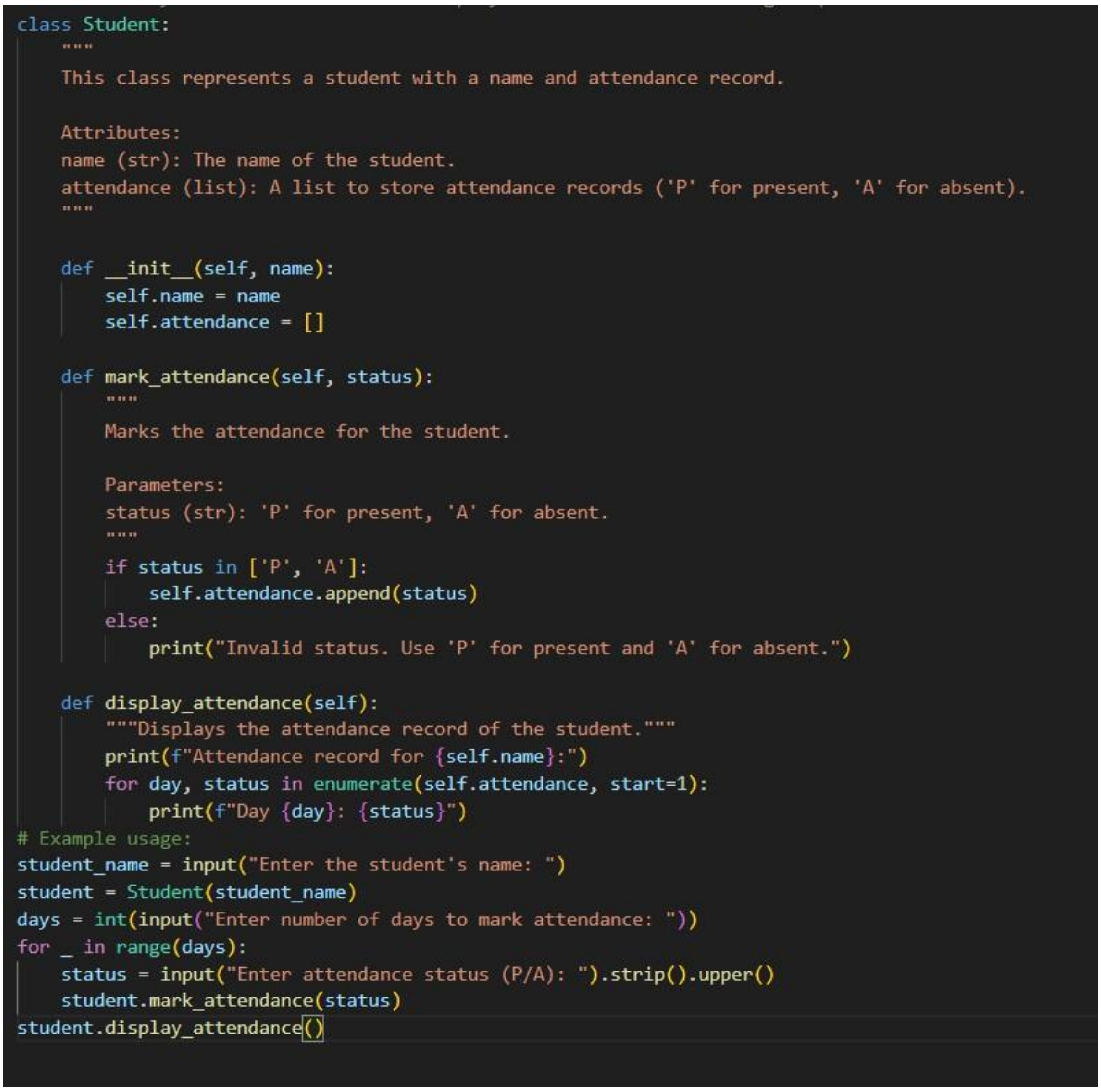
Task Description #4 (AI-Assisted Code Completion for Class-Based Attendance System)

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

Prompt:

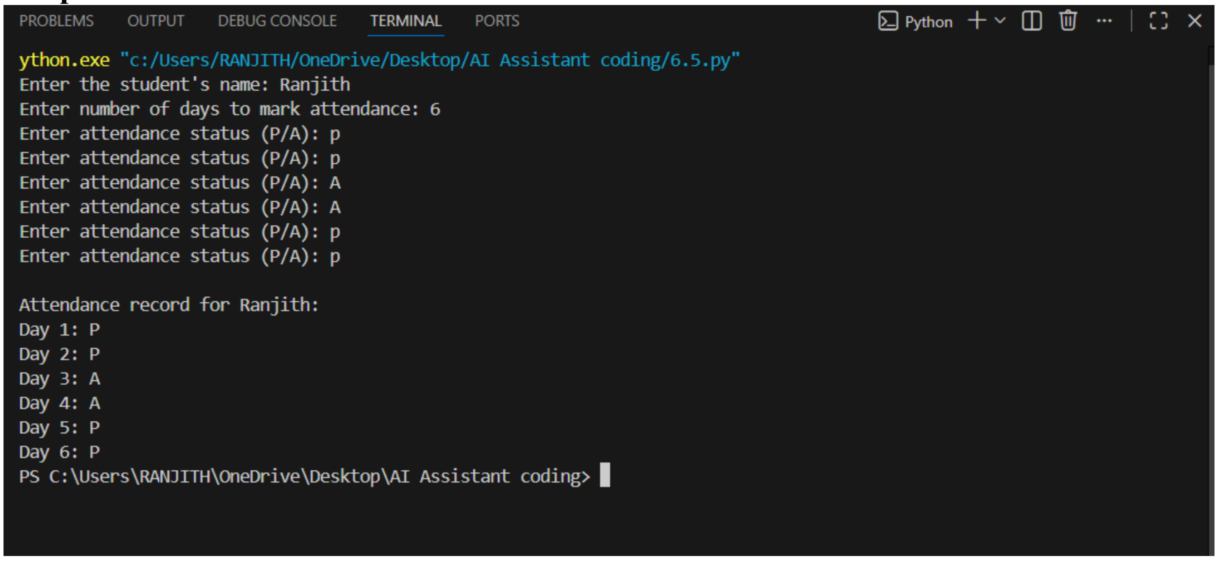
#Generate a Python class to mark and display student attendance using loops.

Code:



```python
class Student:
    """
    This class represents a student with a name and attendance record.

    Attributes:
    name (str): The name of the student.
    attendance (list): A list to store attendance records ('P' for present, 'A' for absent).
    """

    def __init__(self, name):
        self.name = name
        self.attendance = []

    def mark_attendance(self, status):
        """
        Marks the attendance for the student.

        Parameters:
        status (str): 'P' for present, 'A' for absent.
        """
        if status in ['P', 'A']:
            self.attendance.append(status)
        else:
            print("Invalid status. Use 'P' for present and 'A' for absent.")

    def display_attendance(self):
        """Displays the attendance record of the student."""
        print(f"Attendance record for {self.name}:")
        for day, status in enumerate(self.attendance, start=1):
            print(f"Day {day}: {status}")
# Example usage:
student_name = input("Enter the student's name: ")
student = Student(student_name)
days = int(input("Enter number of days to mark attendance: "))
for _ in range(days):
    status = input("Enter attendance status (P/A): ").strip().upper()
    student.mark_attendance(status)
student.display_attendance()
```

Output:



```
ython.exe "c:/Users/RANJITH/OneDrive/Desktop/AI Assistant coding/6.5.py"
Enter the student's name: Ranjith
Enter number of days to mark attendance: 6
Enter attendance status (P/A): p
Enter attendance status (P/A): p
Enter attendance status (P/A): A
Enter attendance status (P/A): A
Enter attendance status (P/A): p
Enter attendance status (P/A): p

Attendance record for Ranjith:
Day 1: P
Day 2: P
Day 3: A
Day 4: A
Day 5: P
Day 6: P
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding>
```

Observation:

1. The program uses a Student class to store student details and attendance. This follows Object-Oriented Programming principles.

2. Attendance is stored in a list using P for present and A for absent. This makes attendance

tracking simple and efficient.

3. Input validation is implemented for attendance status. It prevents invalid entries other than P and A.

4. Attendance is marked day by day using a loop. This allows flexible entry for any number of days.

5. The display_attendance() method shows attendance clearly with day numbers.It improves readability and understanding of records.

6. The example usage demonstrates real-time user input and output. This helps beginners easily understand how the program works.

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

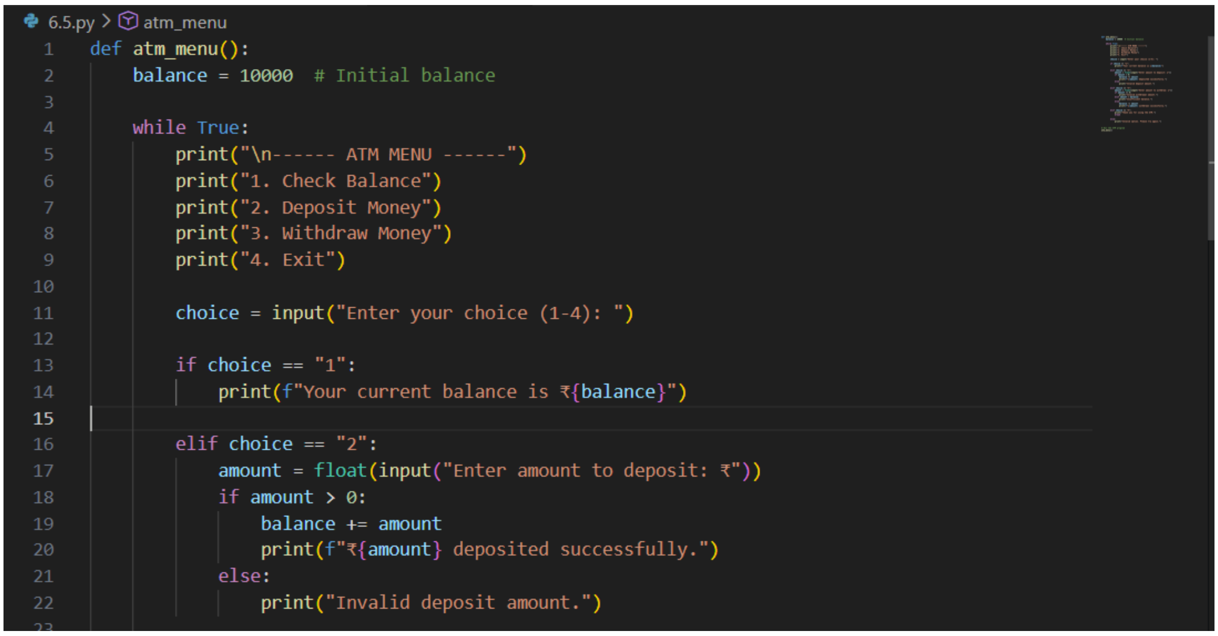
Task: Use an AI tool to complete a navigation menu.

Prompt:

#Generate a Python program using loops and conditionals to simulate an ATM menu.

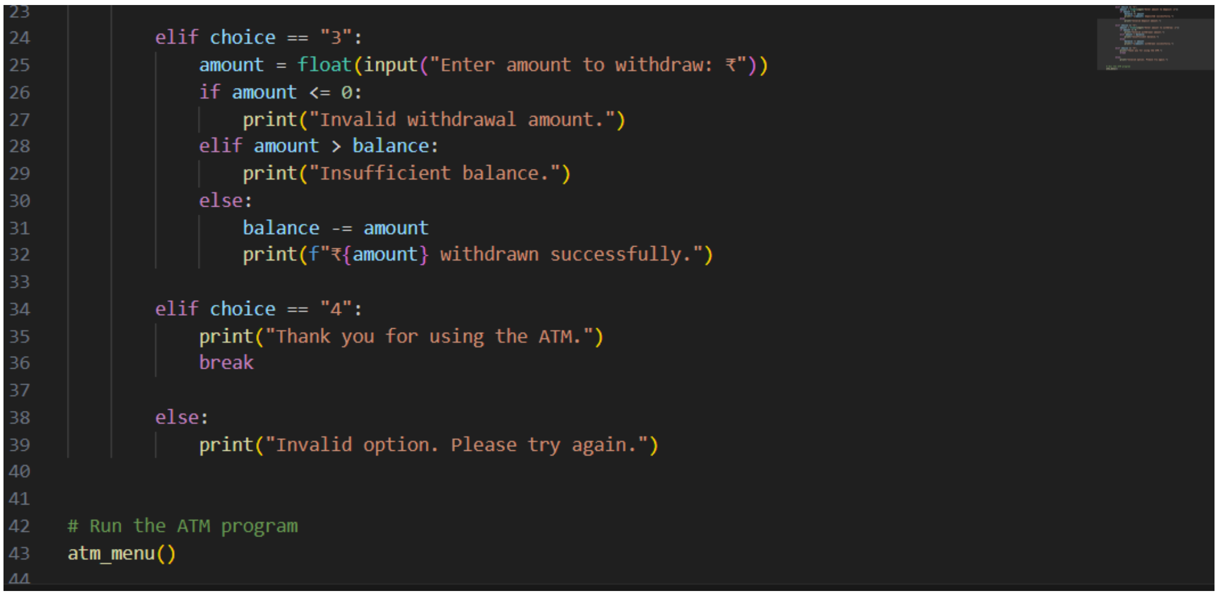
#Initialize ATM with a starting balance of $100

Code:



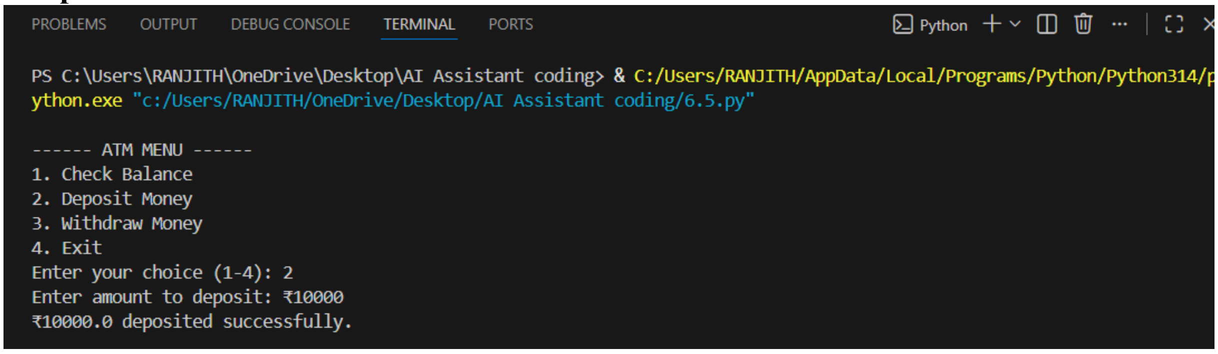
6.5.py > atm_menu

```python
def atm_menu():
    balance = 10000  # Initial balance

    while True:
        print("\n------ ATM MENU ------")
        print("1. Check Balance")
        print("2. Deposit Money")
        print("3. Withdraw Money")
        print("4. Exit")

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

        if choice == "1":
            print(f"Your current balance is ₹{balance}")

        elif choice == "2":
            amount = float(input("Enter amount to deposit: ₹"))
            if amount > 0:
                balance += amount
                print(f"₹{amount} deposited successfully.")
            else:
                print("Invalid deposit amount.")
```



```python
        elif choice == "3":
            amount = float(input("Enter amount to withdraw: ₹"))
            if amount <= 0:
                print("Invalid withdrawal amount.")
            elif amount > balance:
                print("Insufficient balance.")
            else:
                balance -= amount
                print(f"₹{amount} withdrawn successfully.")

        elif choice == "4":
            print("Thank you for using the ATM.")
            break

        else:
            print("Invalid option. Please try again.")


# Run the ATM program
atm_menu()
```

Output:



```
PS C:\Users\RANJITH\OneDrive\Desktop\AI Assistant coding> & C:/Users/RANJITH/AppData/Local/Programs/Python/Python314/p
ython.exe "c:/Users/RANJITH/OneDrive/Desktop/AI Assistant coding/6.5.py"

------ ATM MENU ------
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1-4): 2
Enter amount to deposit: ₹10000
₹10000.0 deposited successfully.
```

Observation:

1. The program uses an ATM class to simulate basic ATM operations. This follows Object-Oriented Programming concepts.

2. The balance is stored as a class attribute and updated dynamically. It reflects deposits and withdrawals accurately.

3. A menu-driven approach is implemented using a loop. This allows the user to perform multiple operations continuously.

4. Input validation is done for deposit and withdrawal amounts. It prevents negative values and invalid transactions.

5. The program checks for sufficient balance before withdrawal. This avoids overdraft situations.

6. Clear and user-friendly messages are displayed for every operation. This improves usability and user understanding.