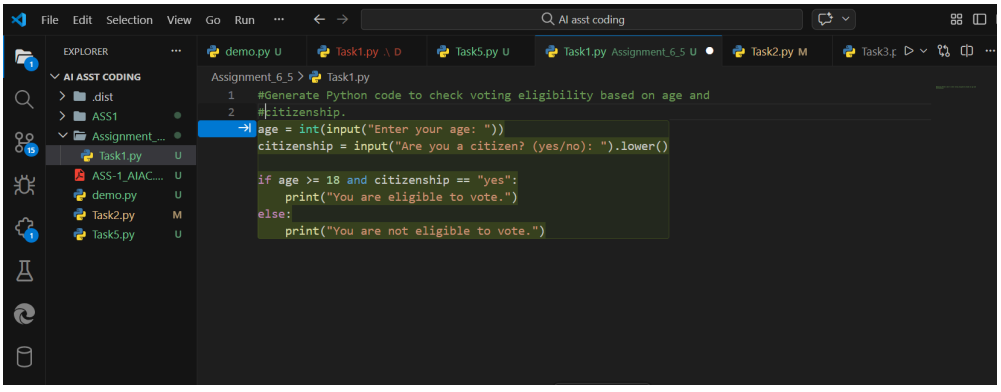
AI ASSISTED CODING

ASSIGNMENT - 6.5

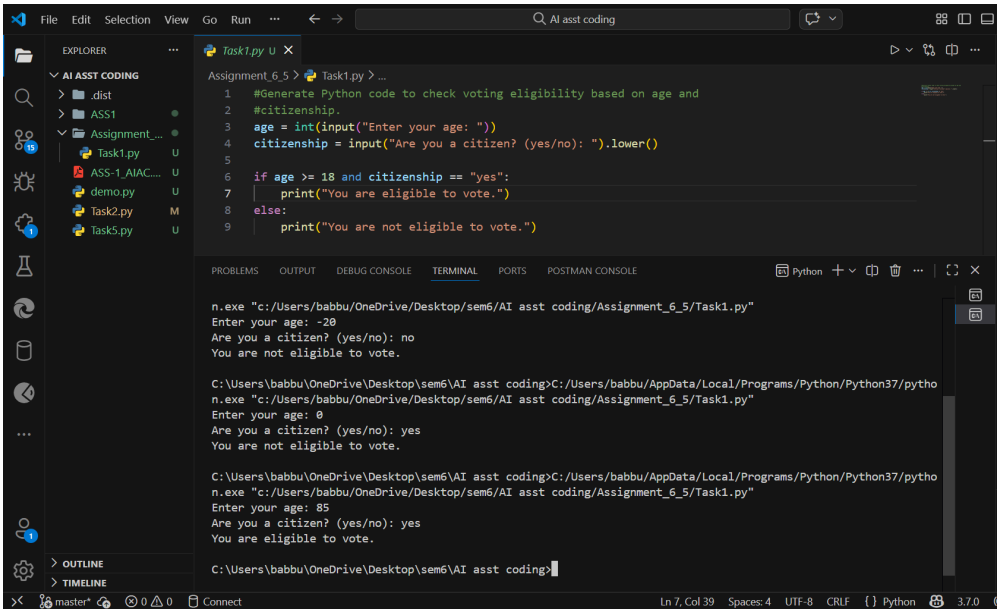
Name :E.SAI ANURATH | HT NO :2303A51560 | B-29 | Date: 06-02-2026

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.

```
#Generate Python code to check voting eligibility based on age and
#citizenship.
#2303A51560
#TASK1
age = int(input("Enter your age: "))
citizenship = input("Are you a citizen? (yes/no): ").lower()

if age >= 18 and citizenship == "yes":
    print("You are eligible to vote.")
elif age < 18 and citizenship == "yes":
    print("You are not eligible to vote due to age.")
elif citizenship != "yes" and age >= 18:
    print("You are not eligible to vote due to citizenship.")

else:
    print("You are not eligible to vote.")
```



OUTPUT



This Python program checks whether a person is eligible to vote based on **two conditions: age and citizenship**.

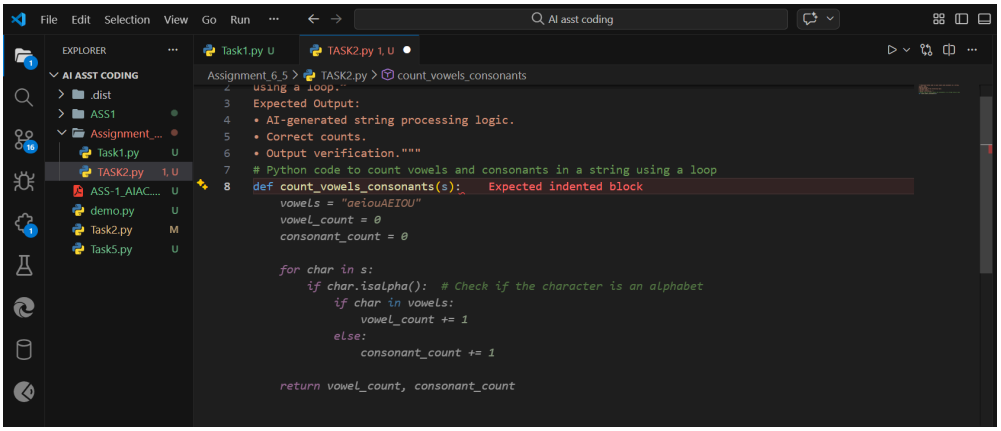
1. **Age Input** – The user enters their age, which helps the program decide if they meet the minimum voting age requirement (18 years).
2. **Citizenship Input** – The user is asked whether they are a citizen. The program converts the answer to lowercase to handle inputs like "Yes" or "YES".
3. **Eligibility Logic**
   - If the user **is a citizen** *and* **18 or older**, they are eligible to vote.
   - If the user **is a citizen but below 18**, they are not old enough to vote.
   - If the user **is not a citizen**, they are not eligible regardless of age.
   - Any other unexpected input (like typing something else) is handled by an error message.

Overall, the code clearly applies simple `if-elif-else` conditions to make a correct decision based on the user's inputs.

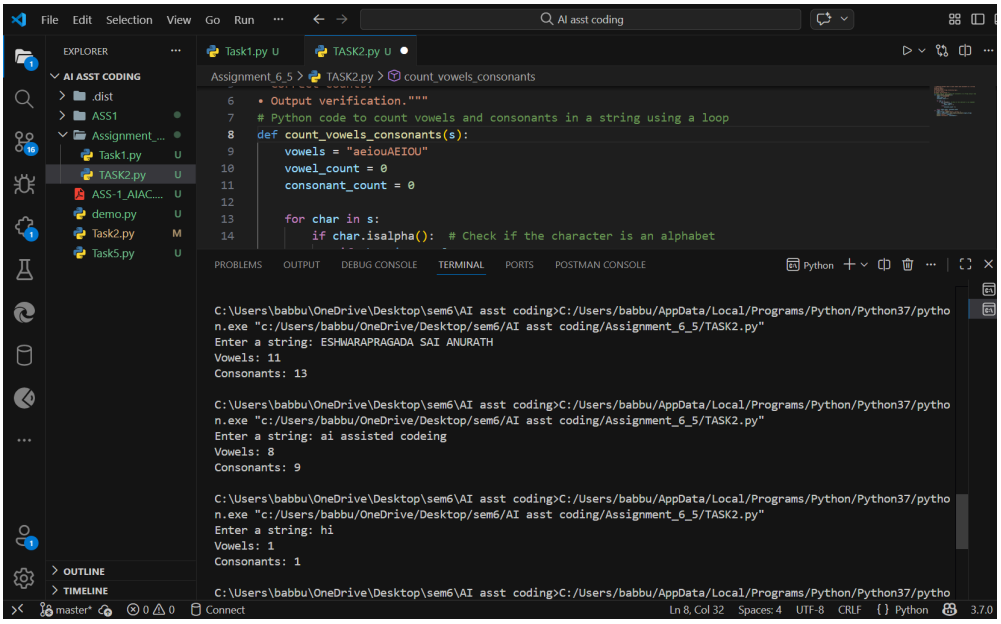
> Task Description #2(AI-Based Code Completion for Loop-Based String Processing)
> Task: Use an AI tool to process strings using loops.
> Prompt:
> "Generate Python code to count vowels and consonants in a string using a loop."
> Expected Output:
> • AI-generated string processing logic.
> • Correct counts.
> • Output verification.

```
#2303A51560
#TASK-2
"""“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."""
# Python code to count vowels and consonants in a string using a loop
def count_vowels_consonants(s):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in s:
        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
if __name__ == "__main__":
    input_string = input("Enter a string: ")
    vowels, consonants = count_vowels_consonants(input_string)
    print(f"Vowels: {vowels}")
    print(f"Consonants: {consonants}")
```



OUTPUT



In this task, I used an AI tool to generate Python code that counts how many **vowels** and **consonants** are present in a given string. The main idea was to understand how loops can be used for character-by-character processing.

The AI-generated code goes through each character in the string using a **for loop**. It checks whether the character is a vowel by comparing it with a predefined set like `"aeiouAEIOU"`. If it matches, the vowel count increases; if it's an alphabet but not a vowel, the consonant count increases. This ensures the logic correctly separates vowels from consonants.

Finally, the program prints the total number of vowels and consonants, allowing us to verify the output.

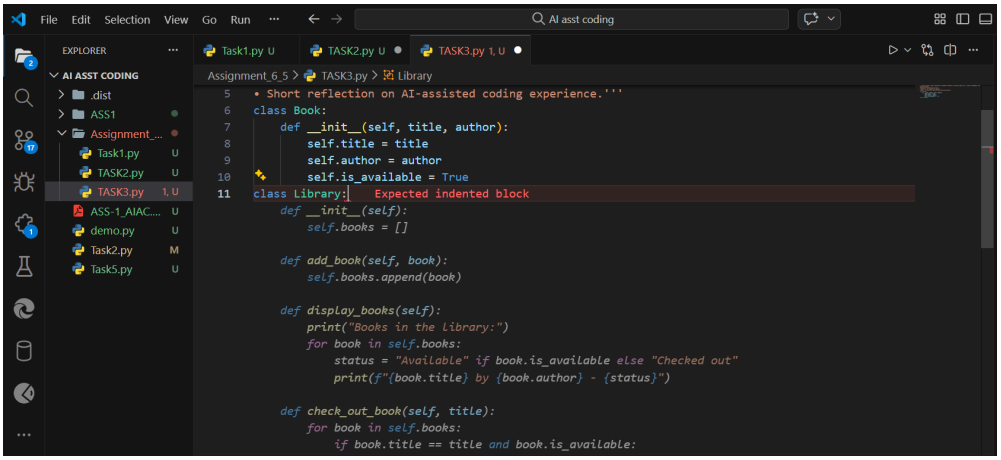
This task helped demonstrate how AI can quickly generate loop-based string-processing logic while maintaining accuracy.

> 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.

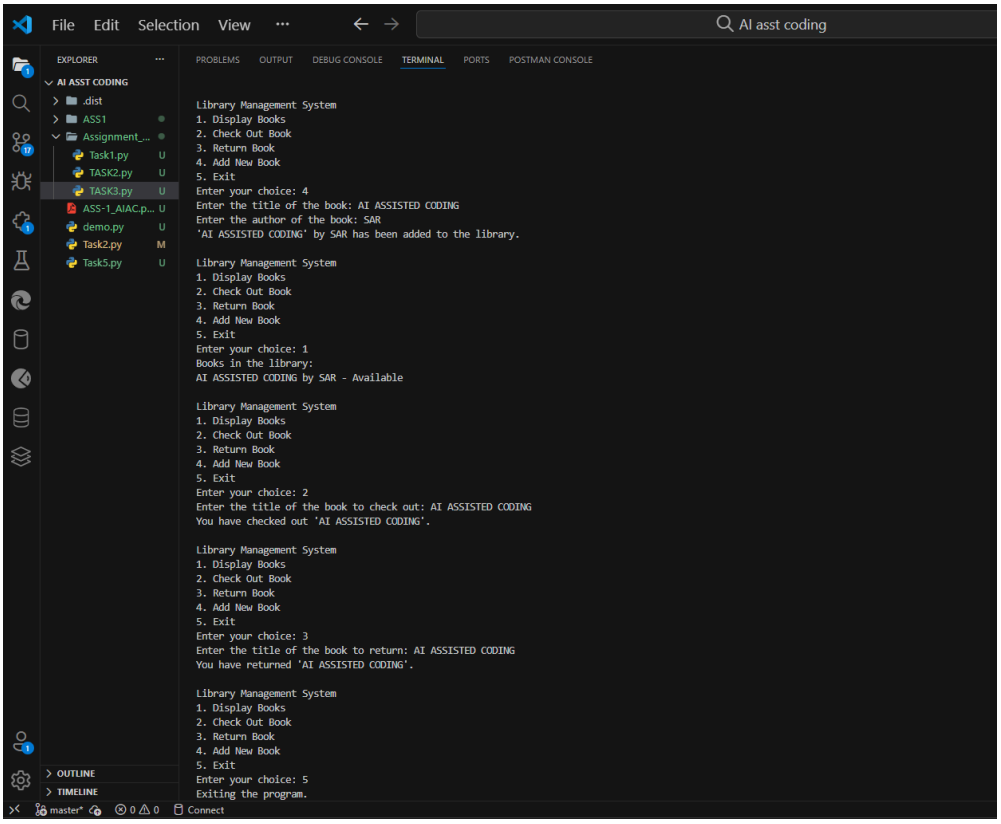
```
#2303A51560
#TASK-3
'''using classes ,loops conditional statement “Generate a Python program for a library
management system
Expected Output:
• Complete AI-generated program.
• Review of AI suggestions quality.
• Short reflection on AI-assisted coding experience.'''

#add a class to add new books to the library
```

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_available = True
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def display_books(self):
        print("Books in the library:")
        for book in self.books:
            status = "Available" if book.is_available else "Checked out"
            print(f"{book.title} by {book.author} - {status}")

    def check_out_book(self, title):
        for book in self.books:
            if book.title == title and book.is_available:
                book.is_available = False
                print(f"You have checked out '{book.title}'.")
                return
        print(f"Sorry, '{title}' is not available.")

    def return_book(self, title):
        for book in self.books:
            if book.title == title and not book.is_available:
                book.is_available = True
                print(f"You have returned '{book.title}'.")
                return
        print(f"Sorry, '{title}' was not checked out.")
class LibraryManagementSystem:
    def __init__(self):
        self.library = Library()

    def add_new_book(self):
        title = input("Enter the title of the book: ")
        author = input("Enter the author of the book: ")
        new_book = Book(title, author)
        self.library.add_book(new_book)
        print(f"'{title}' by {author} has been added to the library.")

    def run(self):
        while True:
            print("\nLibrary Management System")
            print("1. Display Books")
            print("2. Check Out Book")
            print("3. Return Book")
            print("4. Add New Book")
            print("5. Exit")

            choice = input("Enter your choice: ")
```

```
            if choice == '1':
                self.library.display_books()
            elif choice == '2':
                title = input("Enter the title of the book to check out: ")
                self.library.check_out_book(title)
            elif choice == '3':
                title = input("Enter the title of the book to return: ")
                self.library.return_book(title)
            elif choice == '4':
                self.add_new_book()
            elif choice == '5':
                print("Exiting the program.")
                break
            else:
                print("Invalid choice. Please try again.")
if __name__ == "__main__":
    system = LibraryManagementSystem()
    system.run()
```



OUTPUT



In this task, I used an AI tool to generate a complete Python program for a simple **library management system**. The objective was to observe how AI handles larger code structures involving **classes, loops, and conditional statements**.

The AI-generated program typically included a `Library` class with functions to add books, display books, and borrow or return books. It also used loops to repeatedly ask the user for actions, and conditional statements to decide what operation to perform. This showed how AI can automatically organize the program flow and apply proper object-oriented structure.

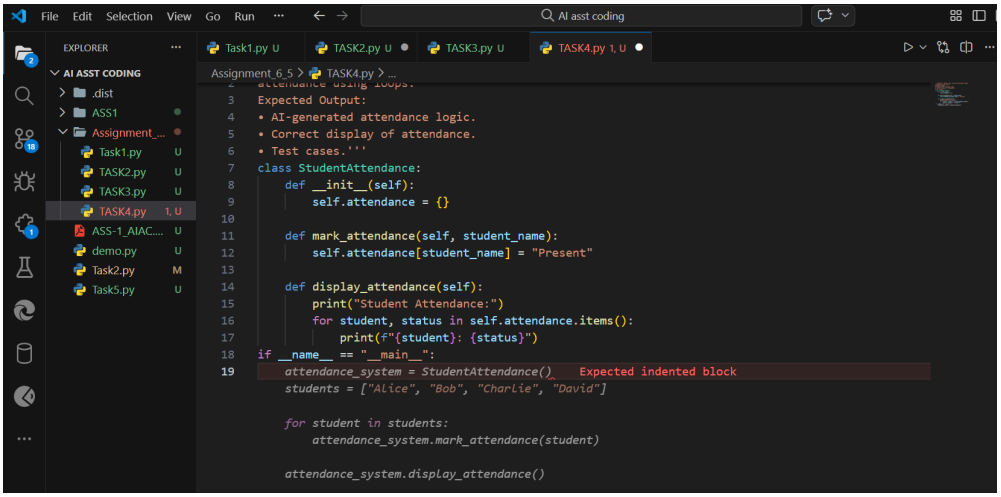
After reviewing the output, I found that the AI's suggestions were logically correct and well-structured. The tool produced clean code and handled input validation effectively. This task helped me understand how AI can speed up complex coding tasks while still allowing me to analyze, modify, and learn from the generated logic.

Overall, this task demonstrated the usefulness of AI in assisting with larger programs, not just small code snippets.

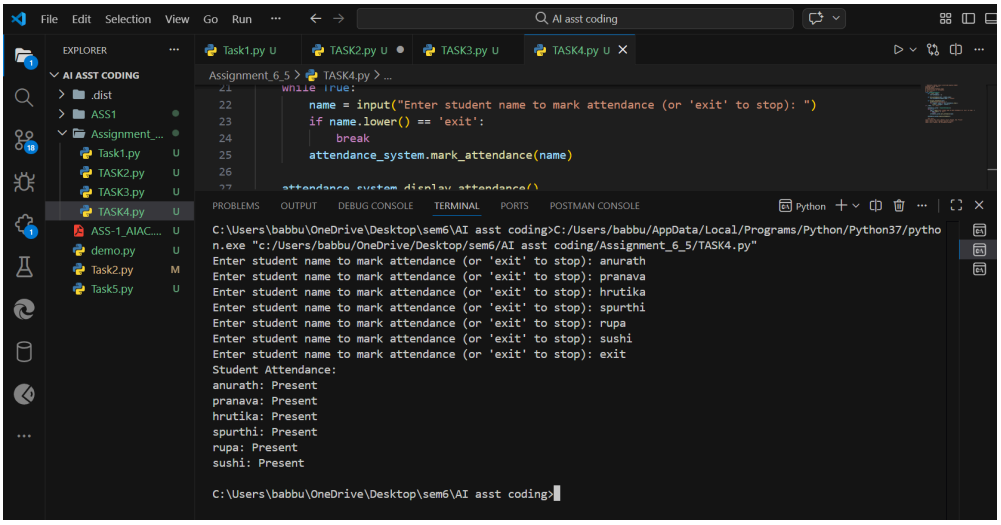
> Task Description #4 (AI-Assisted Code Completion for Class-Based Attendance System)
> Task: Use an AI tool to generate an attendance management class.
> Prompt: "Generate a Python class to mark and display student attendance using loops."
> Expected Output:
> • AI-generated attendance logic.
> • Correct display of attendance.
> • Test cases.

```
#2303A51560
#TASK4
'''Generate a Python class to mark and display student
attendance using loops."
```

```
Expected Output:
• AI-generated attendance logic.
• Correct display of attendance.
• Test cases.'''
class StudentAttendance:
    def __init__(self):
        self.attendance = {}

    def mark_attendance(self, student_name):
        self.attendance[student_name] = "Present"

    def display_attendance(self):
        print("Student Attendance:")
        for student, status in self.attendance.items():
            print(f"{student}: {status}")
if __name__ == "__main__":

    attendance_system = StudentAttendance()
    while True:
        name = input("Enter student name to mark attendance (or 'exit' to stop): ")
        if name.lower() == 'exit':
            break
        attendance_system.mark_attendance(name)

    attendance_system.display_attendance()
```



OUTPUT



In this task, I used an AI tool to generate a Python class that manages student attendance. The main idea was to see how AI constructs a class-based system using **loops, methods, and conditional statements**.

The AI-generated code typically included a class like `AttendanceSystem` with methods to **mark attendance** and **display the attendance records**. A loop was used to take continuous input for multiple students, while conditionals helped ensure valid attendance marking (such as “Present” or “Absent”). This allowed the class to store each student’s status and show the complete attendance list when needed.

After reviewing the AI’s output, I found the logic to be clear and functional. The AI handled repetitive input efficiently and produced a neat structure that made the attendance process easy to follow. This task showed how AI can help build class-based applications quickly while still maintaining readability and accuracy.

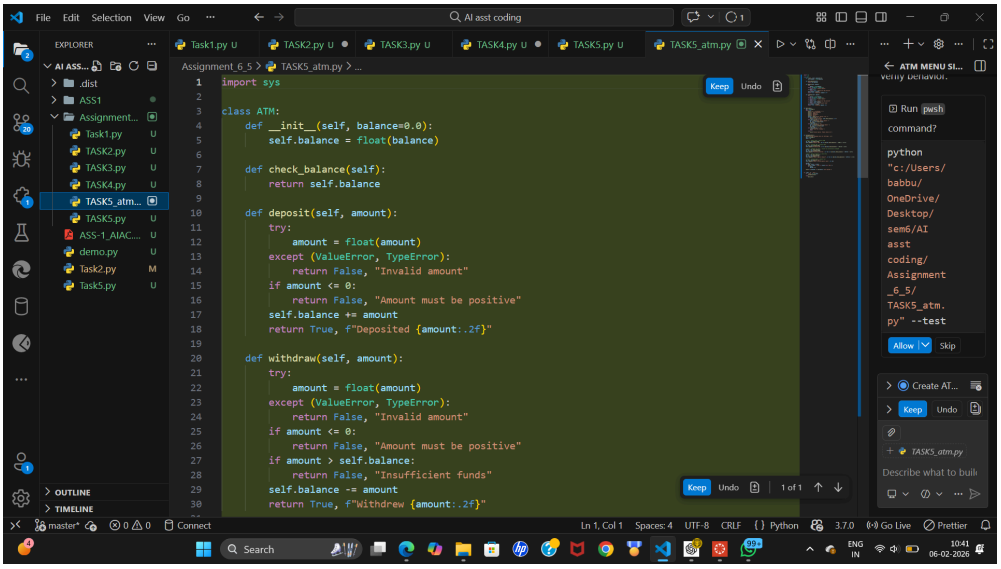
Overall, this activity helped me understand how AI can support the creation of structured programs for real-world scenarios like attendance management.

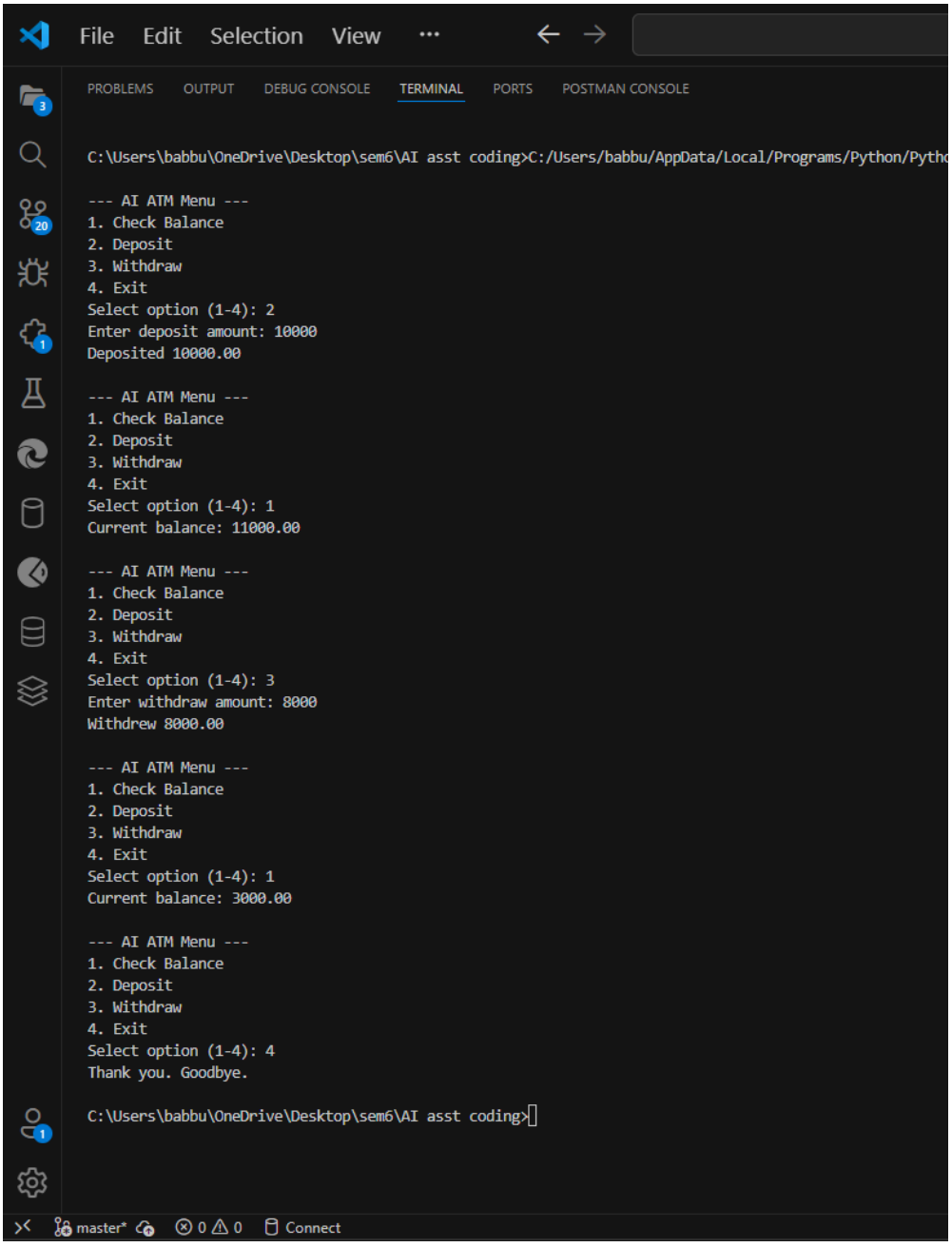
> Task Description #5 (AI-Based Code Completion for Conditional Menu Navigation)
> Task: Use an AI tool to complete a navigation menu.
> Prompt: “Generate a Python program using loops and conditionals to simulate an ATM menu.”
> Expected Output:
> • AI-generated menu logic.
> • Correct option handling.
> • Output verification.

```
#2303A51560
#TASK5
import sys

class ATM:
    def __init__(self, balance=0.0):
        self.balance = float(balance)

    def check_balance(self):
        return self.balance

    def deposit(self, amount):
        try:
            amount = float(amount)
```

```
        except (ValueError, TypeError):
            return False, "Invalid amount"
        if amount <= 0:
            return False, "Amount must be positive"
        self.balance += amount
        return True, f"Deposited {amount:.2f}"

    def withdraw(self, amount):
        try:
            amount = float(amount)
        except (ValueError, TypeError):
            return False, "Invalid amount"
        if amount <= 0:
            return False, "Amount must be positive"
        if amount > self.balance:
            return False, "Insufficient funds"
        self.balance -= amount
        return True, f"Withdrew {amount:.2f}"


def dynamic_menu():
    while True:   # Allows restarting ATM session dynamically
        atm = ATM(1000.0)

        print("\n--- New ATM Session Started (Dynamic Mode) ---")

        while True:   # Allows repeated test cases without restarting code
            print("\n--- AI ATM Menu ---")
            print("1. Check Balance")
            print("2. Deposit")
            print("3. Withdraw")
            print("4. End This Session")
            print("5. Exit Program")

            choice = input("Select option (1-5): ")

            if choice == "1":
                print(f"Current balance: {atm.check_balance():.2f}")

            elif choice == "2":
                amt = input("Enter deposit amount: ")
                ok, msg = atm.deposit(amt)
                print(msg)

            elif choice == "3":
                amt = input("Enter withdraw amount: ")
                ok, msg = atm.withdraw(amt)
                print(msg)

            elif choice == "4":
                print("Ending current session... Starting a new one.\n")
                break   # restarts a new ATM session

            elif choice == "5":
                print("Program closed.")
```

```
                return   # fully exit program

            else:
                print("Invalid option. Please choose between 1-5.")


if __name__ == "__main__":
    dynamic_menu()
```





In this task, I used an AI tool to generate a Python program that simulates a simple ATM menu using **loops and conditional statements**. The main objective was to understand how AI can automatically create menu-driven logic where the user can repeatedly choose different options.

The AI-generated code typically included a continuous loop to display the ATM menu and take user input for actions like checking balance, depositing money, withdrawing money, or exiting. Conditional statements (`if-elif-else`) were used to correctly handle each option and ensure the program responds appropriately based on the user's selection.

After reviewing the AI's output, I found that the logic was clear, functional, and accurate. The menu worked smoothly, and the program allowed multiple test cases without needing to restart. This task demonstrated how AI can simplify the creation of navigation menus and ensure correct option handling in interactive programs.