Vyshnavi Parisha

2403a51l34

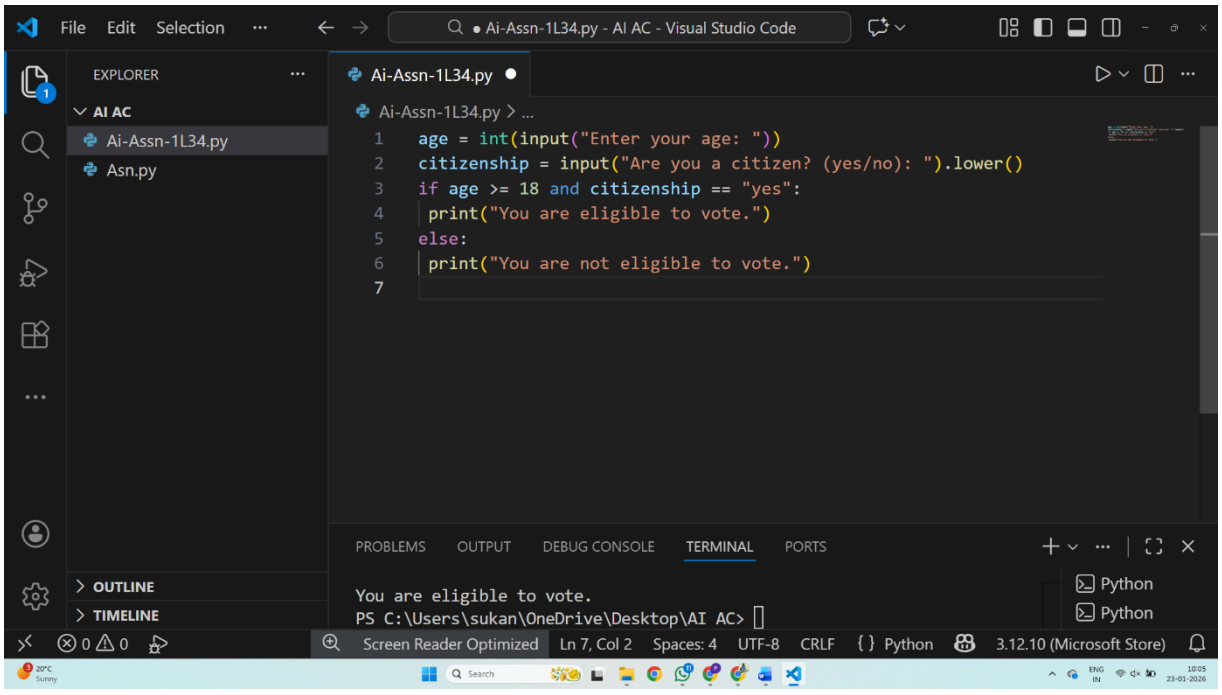
B-52

Lab 5: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals

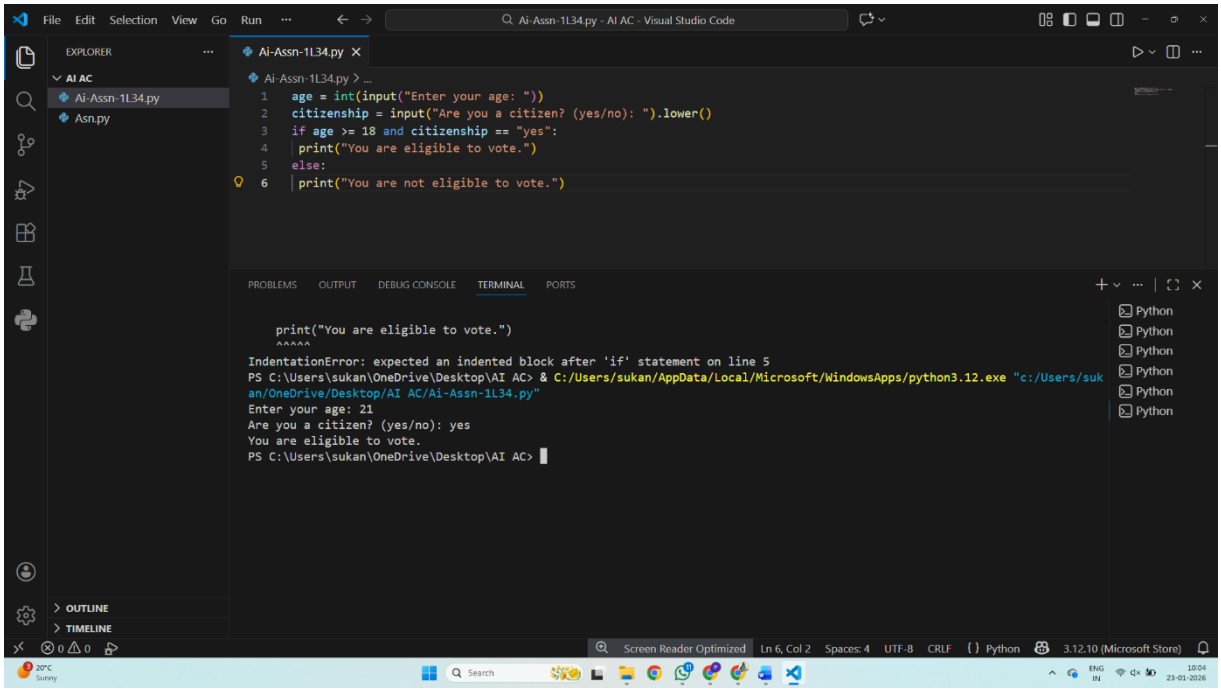
Task Description – 1: AI-Based Code Completion for Conditional Eligibility Check

Prompt: "Generate Python code to check voting eligibility based on age and citizenship."

Code



OUTPUT:



Explanation :

- The program accepts the user's age and citizenship status.
- The conditional statement checks if age is 18 or above and citizenship is "yes".
- If both conditions are satisfied, eligibility is confirmed; otherwise, the user is not eligible.

Verification:

Age = 20, Citizenship = yes → Eligible

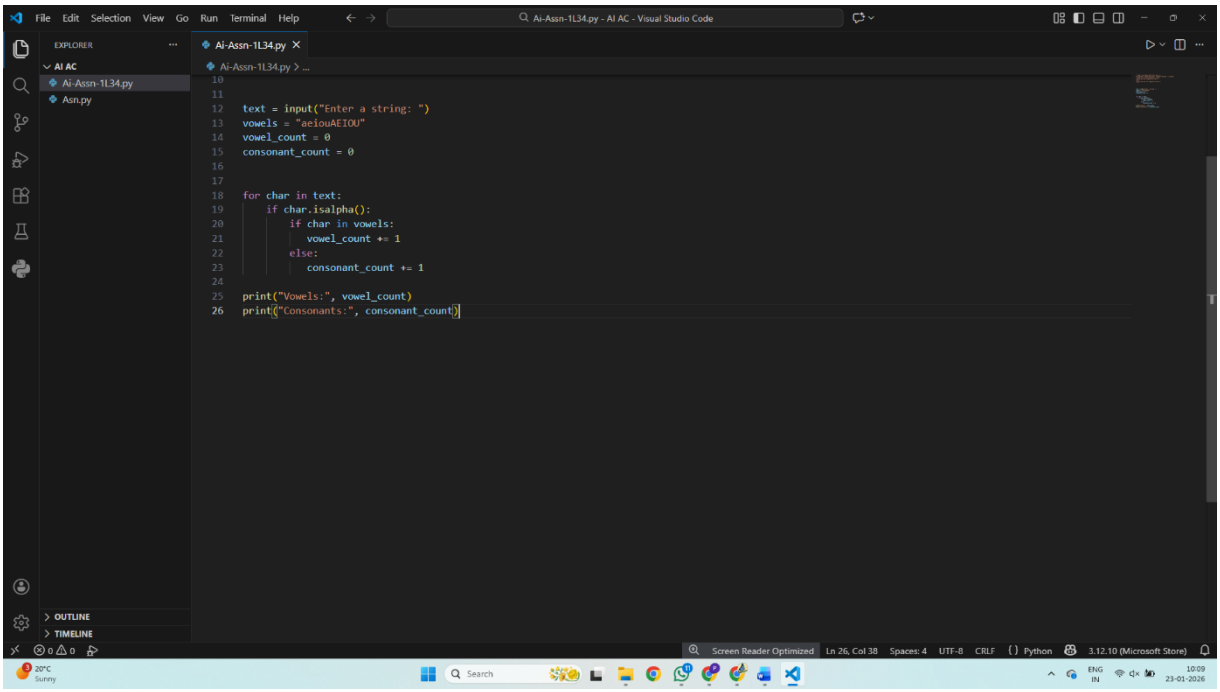
Age = 16, Citizenship = yes → Not eligible

Observations: The initial API code is insecure because it uses a hardcoded API key and does not protect user data. The corrected version improves security by validating inputs, hashing passwords, and using token-based authentication for safer access control.

Task Description – 2: AI-Based Code Completion for Loop-Based String Processing)

Prompt: "Generate Python code to count vowels and consonants in a string using a loop."

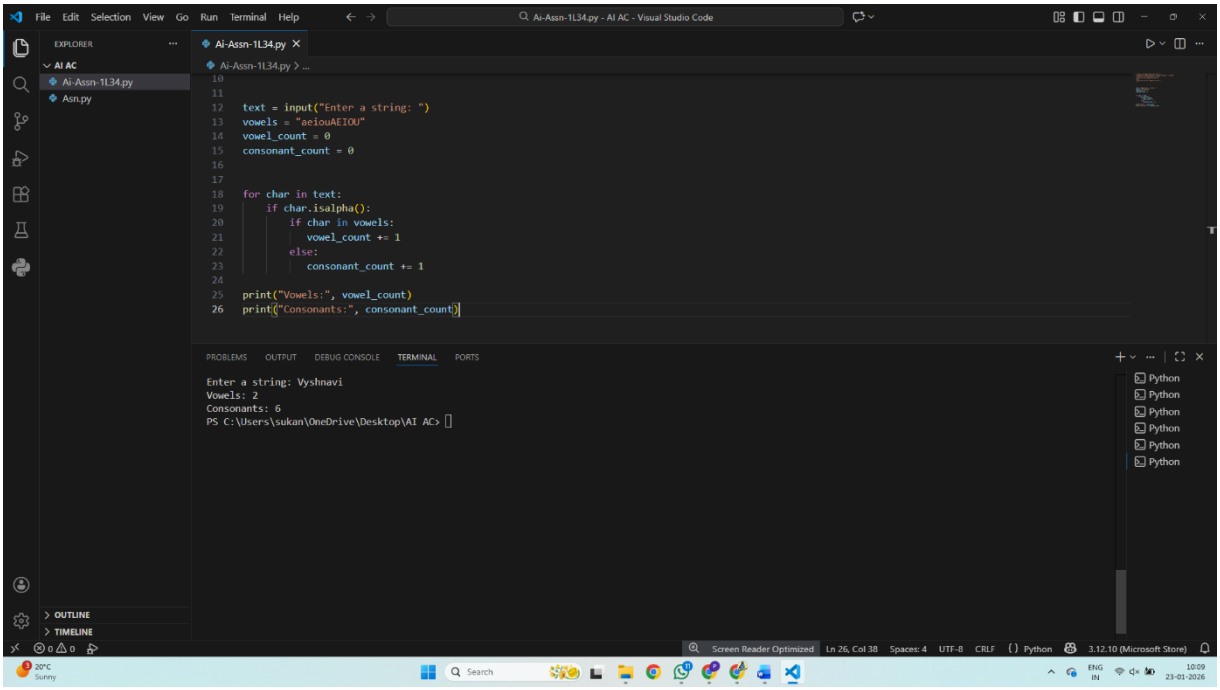
AI-Generated Code:



Observations:

1. The logic unfairly favors urban students
2. Rural or semi-urban students are excluded
3. No flexibility or weighted scoring approach **Improved Version:**

OUTPUT:



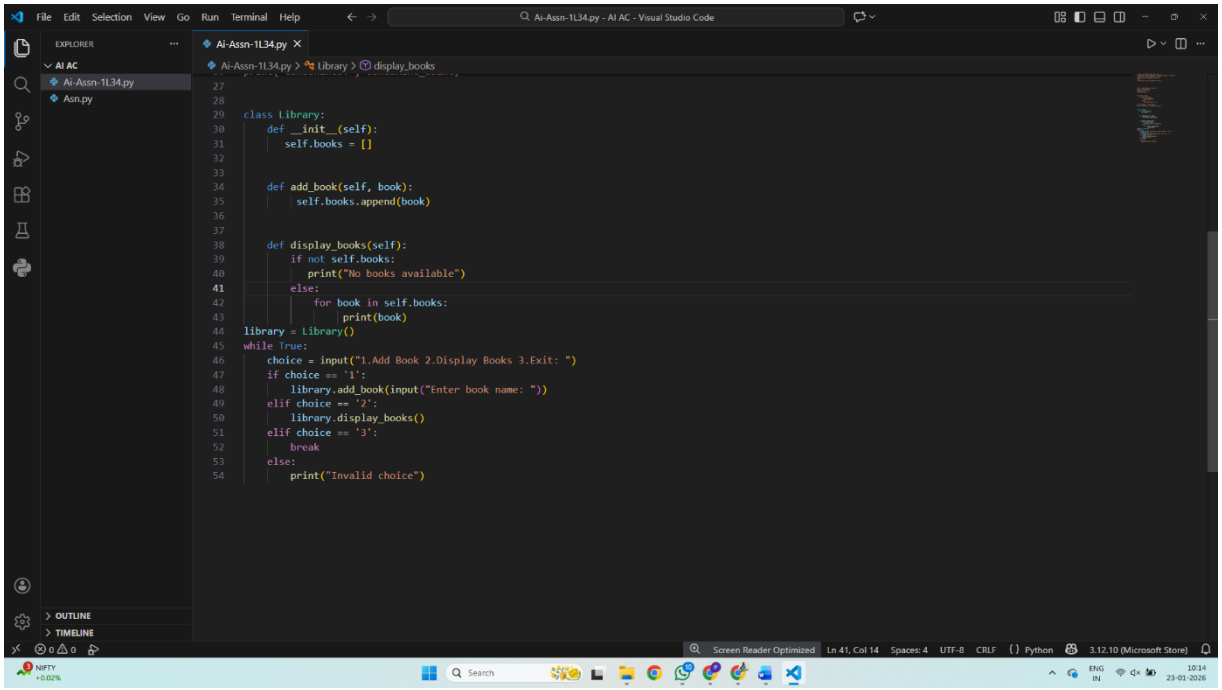
Explanation:

- A loop iterates through each character in the string.
- Alphabetic characters are checked.
- Vowels and consonants are counted separately

Task Description – 3: AI-Assisted Code Completion Reflection Task

Prompt: "Generate a Python program for a library management system using classes, loops, and conditional statements."

Code:

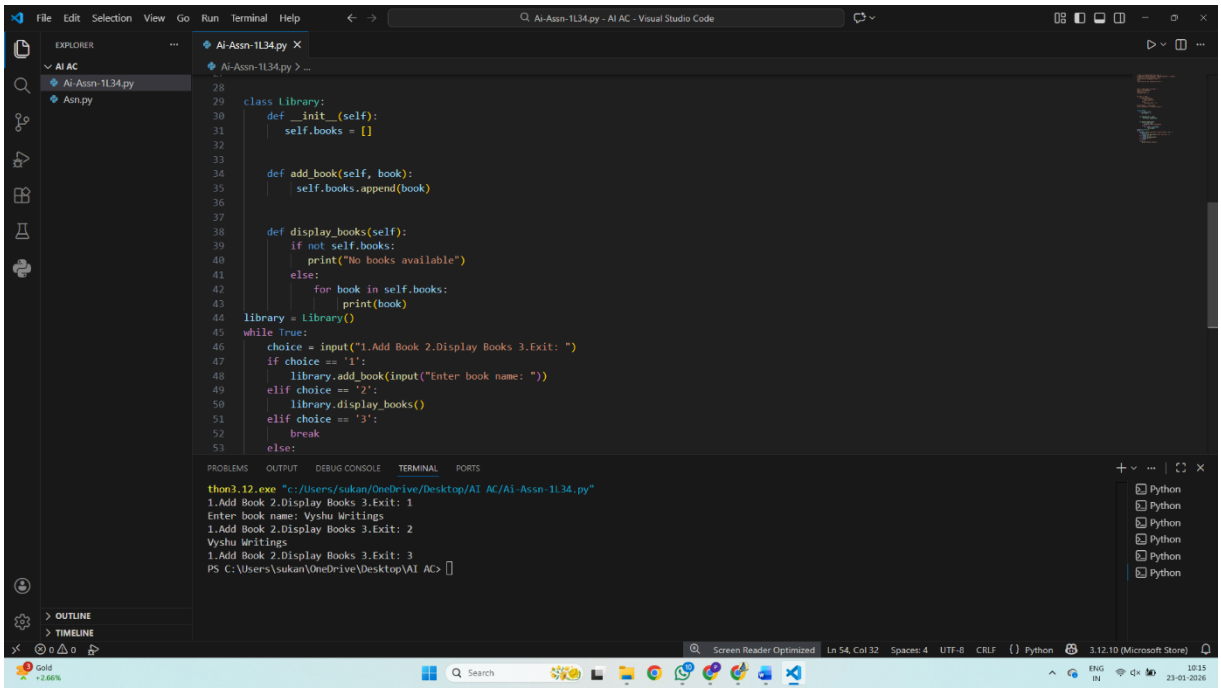


```
class Library:
    def __init__(self):
        self.books = []


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


    def display_books(self):
        if not self.books:
            print("No books available")
        else:
            for book in self.books:
                print(book)
library = Library()
while True:
    choice = input("1.Add Book 2.Display Books 3.Exit: ")
    if choice == '1':
        library.add_book(input("Enter book name: "))
    elif choice == '2':
        library.display_books()
    elif choice == '3':
        break
    else:
        print("Invalid choice")
```

OUTPUT:



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


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


    def display_books(self):
        if not self.books:
            print("No books available")
        else:
            for book in self.books:
                print(book)
library = Library()
while True:
    choice = input("1.Add Book 2.Display Books 3.Exit: ")
    if choice == '1':
        library.add_book(input("Enter book name: "))
    elif choice == '2':
        library.display_books()
    elif choice == '3':
        break
    else:
```

```
thon3.12.exe "c:/Users/sukan/OneDrive/Desktop/AI AC/Ai-Assn-1L34.py"
1.Add Book 2.Display Books 3.Exit: 1
Enter book name: Vyshu Writings
1.Add Book 2.Display Books 3.Exit: 2
Vyshu Writings
1.Add Book 2.Display Books 3.Exit: 3
PS C:\Users\sukan\OneDrive\Desktop\AI AC>
```

Explanation:

- The AI generated a logically correct and readable program.
- It effectively used classes, loops, and conditionals.

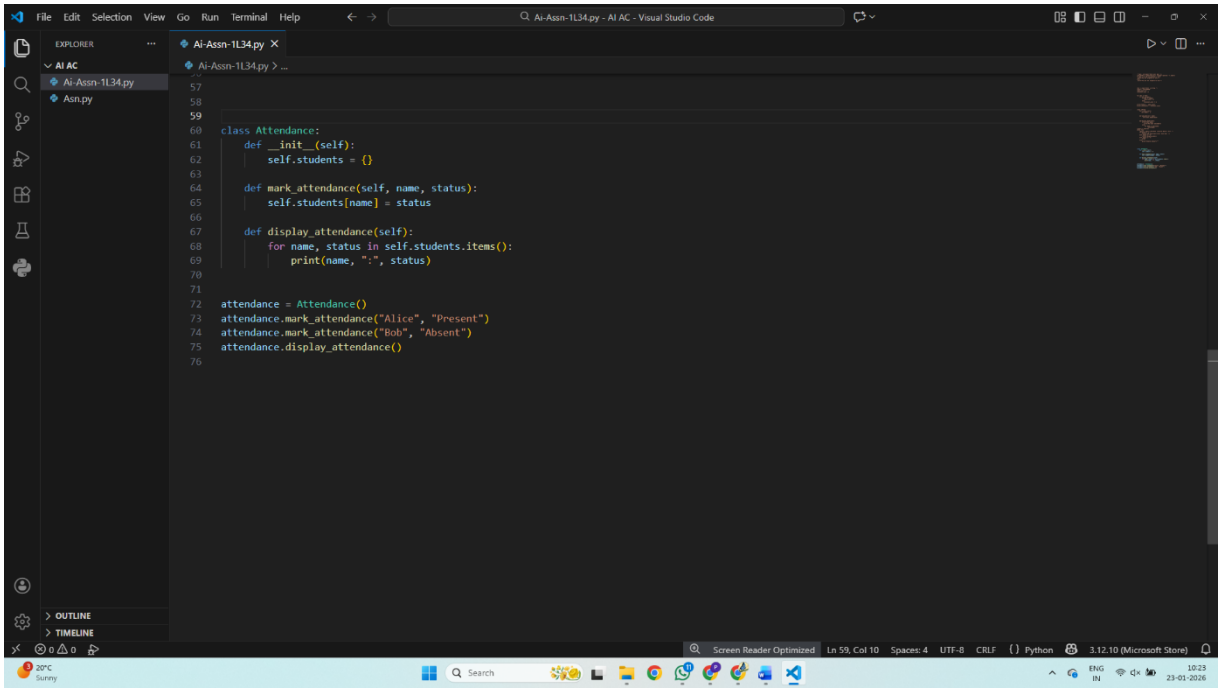
Observations:

AI-assisted coding helps speed up development and provides structured solutions. However, human review is essential to ensure efficiency, security, and correctness.

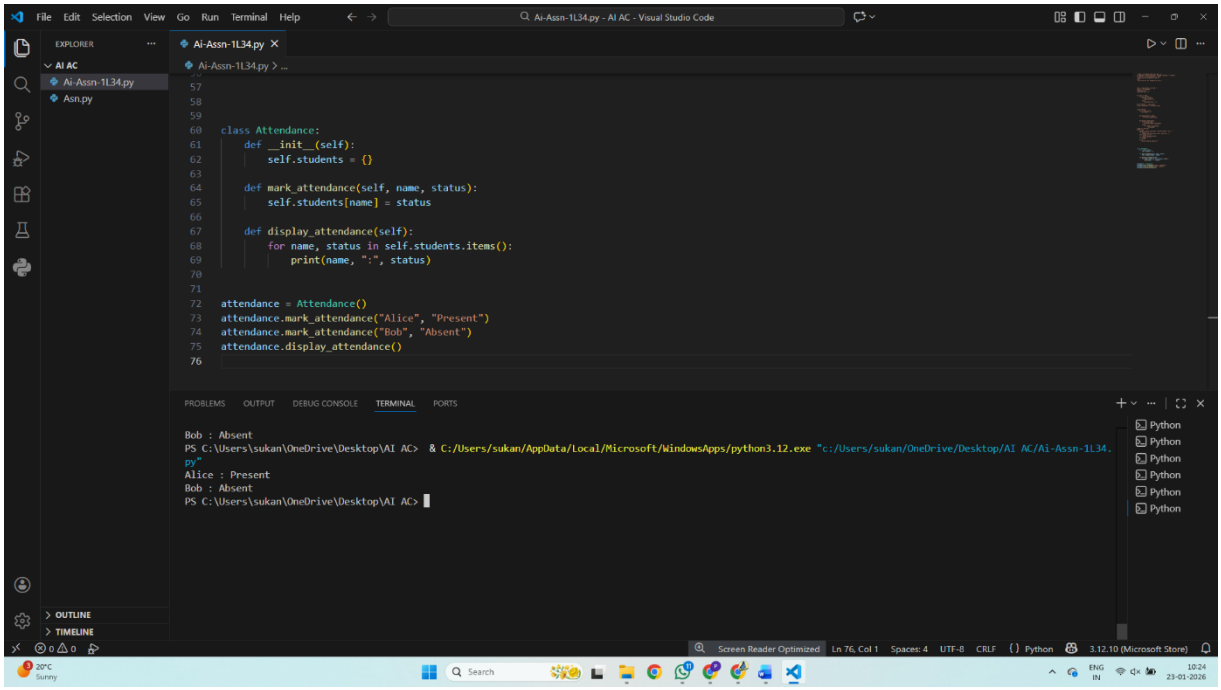
Task Description – 4: AI-Assisted Code Completion for Class-Based Attendance System

Prompt: "Generate a Python class to mark and display student attendance using loops."

Code:



OUTPUT:



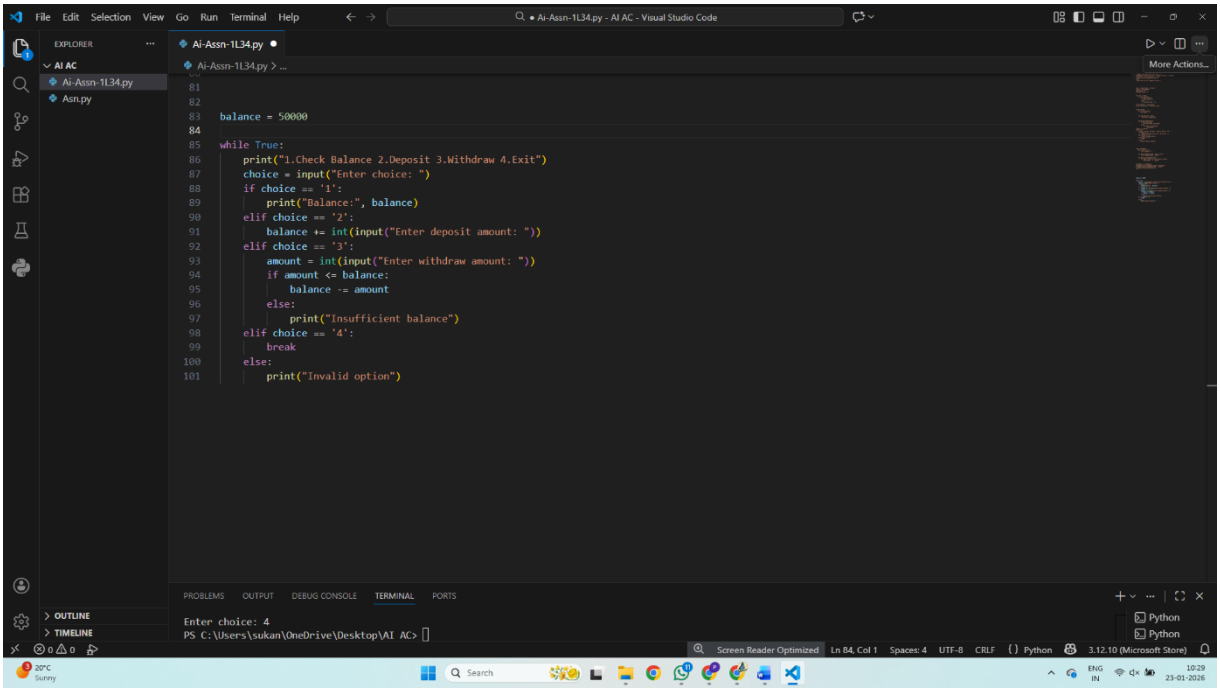
Observations:

- The class-based design improves code organization and readability.
- Dictionary usage ensures efficient storage and retrieval of attendance data.
- The program produces accurate output with minimal code.

Task Description – 5: AI-Based Code Completion for Conditional Menu Navigation

Prompt: "Generate a Python program using loops and conditionals to simulate an ATM menu."

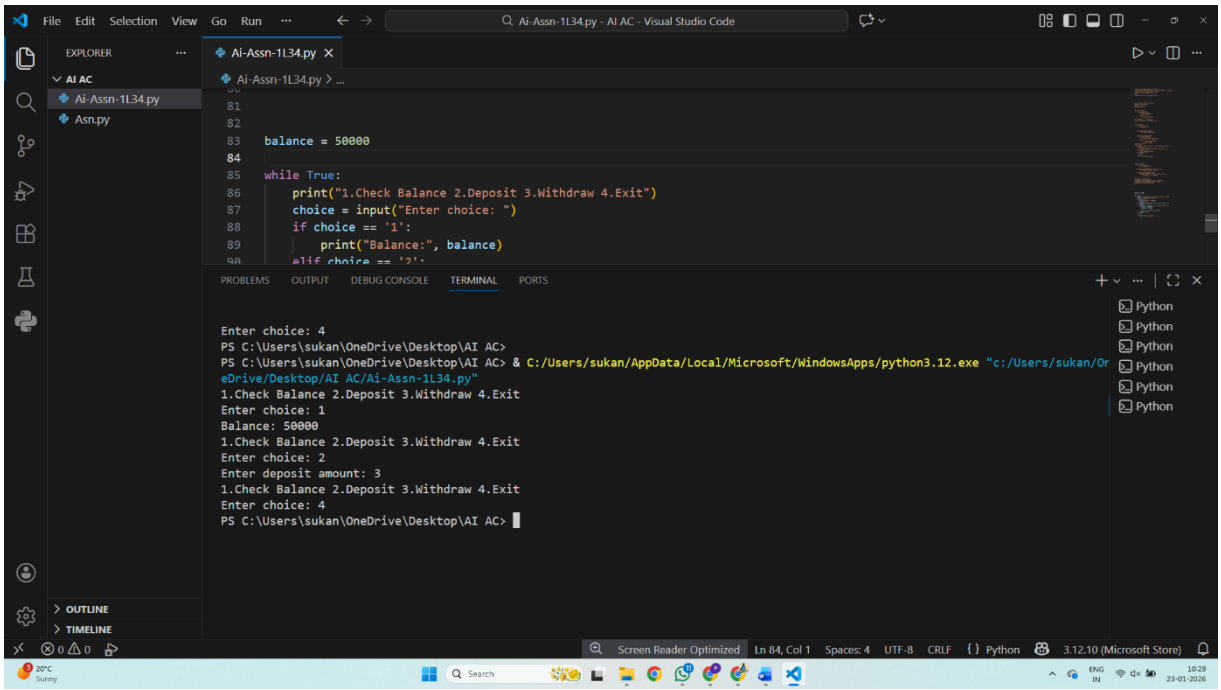
Code:



```
balance = 50000

while True:
    print("1.Check Balance 2.Deposit 3.Withdraw 4.Exit")
    choice = input("Enter choice: ")
    if choice == '1':
        print("Balance:", balance)
    elif choice == '2':
        balance += int(input("Enter deposit amount: "))
    elif choice == '3':
        amount = int(input("Enter withdraw amount: "))
        if amount <= balance:
            balance -= amount
        else:
            print("Insufficient balance")
    elif choice == '4':
        break
    else:
        print("Invalid option")
```

OUTPUT:



```
Enter choice: 4
PS C:\Users\sukan\OneDrive\Desktop\AI AC>
PS C:\Users\sukan\OneDrive\Desktop\AI AC> & C:/Users/sukan/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/sukan/OneDrive/Desktop/AI AC/Ai-Assn-1L34.py"
1.Check Balance 2.Deposit 3.Withdraw 4.Exit
Enter choice: 1
Balance: 50000
1.Check Balance 2.Deposit 3.Withdraw 4.Exit
Enter choice: 2
Enter deposit amount: 3
1.Check Balance 2.Deposit 3.Withdraw 4.Exit
Enter choice: 4
PS C:\Users\sukan\OneDrive\Desktop\AI AC>
```

Observations:

- The program effectively simulates real-world ATM operations.
- Conditional statements and loops are used correctly.
- The system handles invalid inputs and balance conditions properly.