AIAC-6.5

NAME:S.VYSHNAVI

BT-35

2303A52239

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

Eligibility Check)

Task: Use an AI tool to generate eligibility logic.

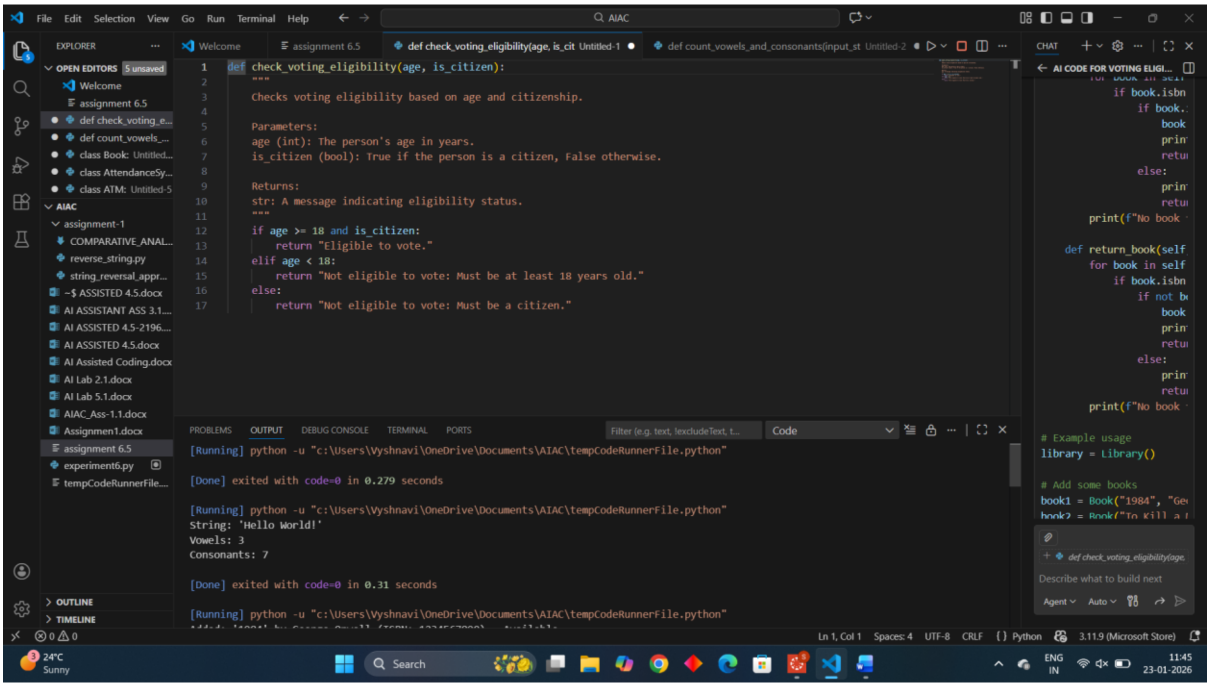
Prompt:

“Generate Python code to check voting eligibility based on age and

citizenship.”

Expected Output:

- AI-generated conditional logic.
- Correct eligibility decisions.
- Explanation of conditions.



def check_voting_eligibility(age, is_cit Untitled-1
def count_vowels_and_consonants(input_st Untitled-2
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
Python

```python
def check_voting_eligibility(age, is_citizen):
    """
    Checks voting eligibility based on age and citizenship.

    Parameters:
    age (int): The person's age in years.
    is_citizen (bool): True if the person is a citizen, False otherwise.

    Returns:
    str: A message indicating eligibility status.
    """
    if age >= 18 and is_citizen:
        return "Eligible to vote."
    elif age < 18:
        return "Not eligible to vote: Must be at least 18 years old."
    else:
        return "Not eligible to vote: Must be a citizen."
```

```
[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"

[Done] exited with code=0 in 0.279 seconds

[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"
String: 'Hello World!'
Vowels: 3
Consonants: 7

[Done] exited with code=0 in 0.31 seconds

[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"
```

Explanation of Conditions:

- **Age Check**: The person must be at least 18 years old to be eligible to vote. If age < 18, they are ineligible due to age.
- **Citizenship Check**: The person must be a citizen (is_citizen == True). If they are 18 or older but not a citizen, they are ineligible.
- **Combined Logic**: Both conditions must be met for eligibility. The function uses an if-elif-else structure to handle the different scenarios and provide clear feedback. This ensures correct eligibility decisions based on the inputs

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

String Processing)

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

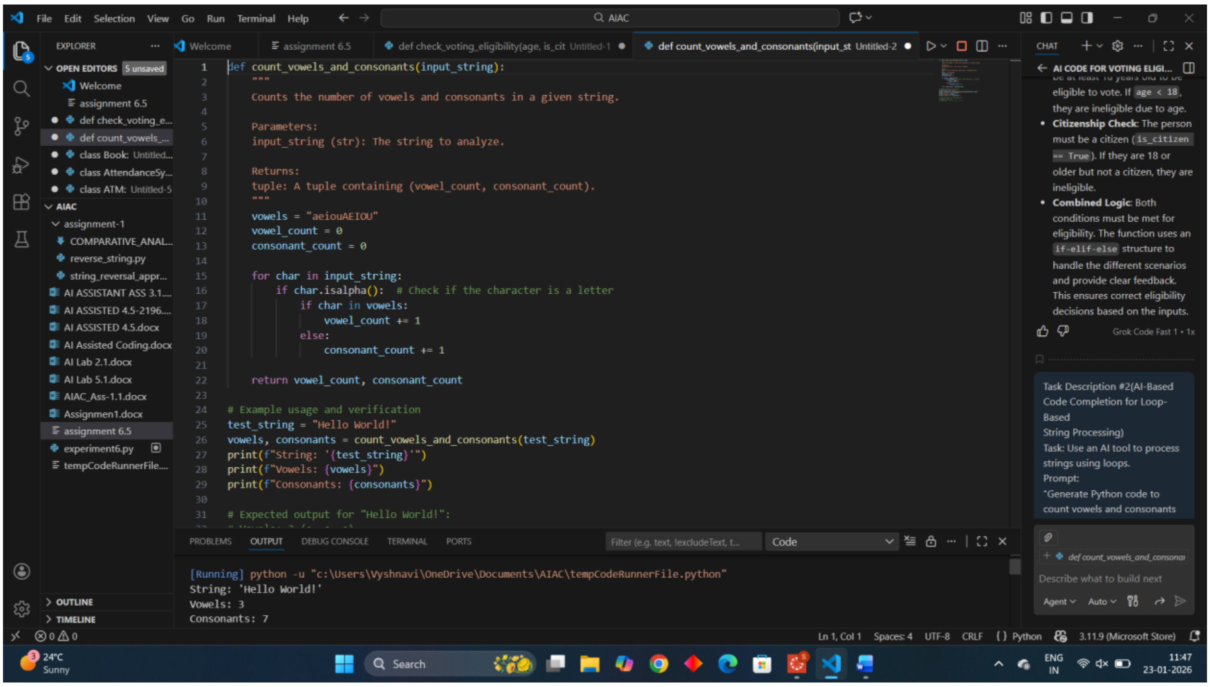
Prompt:

“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.



```python
def count_vowels_and_consonants(input_string):
    """
    Counts the number of vowels and consonants in a given string.

    Parameters:
    input_string (str): The string to analyze.

    Returns:
    tuple: A tuple containing (vowel_count, consonant_count).
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

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

    return vowel_count, consonant_count

# Example usage and verification
test_string = "Hello World!"
vowels, consonants = count_vowels_and_consonants(test_string)
print(f"String: '{test_string}'")
print(f"Vowels: {vowels}")
print(f"Consonants: {consonants}")

# Expected output for "Hello World!":
```

```
[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"
String: 'Hello World!'
Vowels: 3
Consonants: 7
```

eligible to vote. If `age < 18`, they are ineligible due to age.

- **Citizenship Check**: The person must be a citizen (`is_citizen == True`). If they are 18 or older but not a citizen, they are ineligible.
- **Combined Logic**: Both conditions must be met for eligibility. The function uses an `if-elif-else` structure to handle the different scenarios and provide clear feedback. This ensures correct eligibility decisions based on the 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

Explanation:

- **Loop Logic**: The function iterates through each character in the input string using a for loop.
- **Vowel Check**: Vowels are defined as 'a', 'e', 'i', 'o', 'u' (case-insensitive). If a character is alphabetic and in the vowels string, it's counted as a vowel.

- **Consonant Check**: Any alphabetic character not in vowels is counted as a consonant.
- **Non-Letter Handling**: Non-alphabetic characters (like spaces, punctuation) are ignored.
- **Output Verification**: The example with "Hello World!" correctly counts 3 vowels and 7 consonants, demonstrating correct functionality. You can test with other strings to verify.

Task Description #3 (AI-Assisted Code Completion Reflection

Task)

Task: Use an AI tool to generate a complete program using classes,

loops, and conditionals.

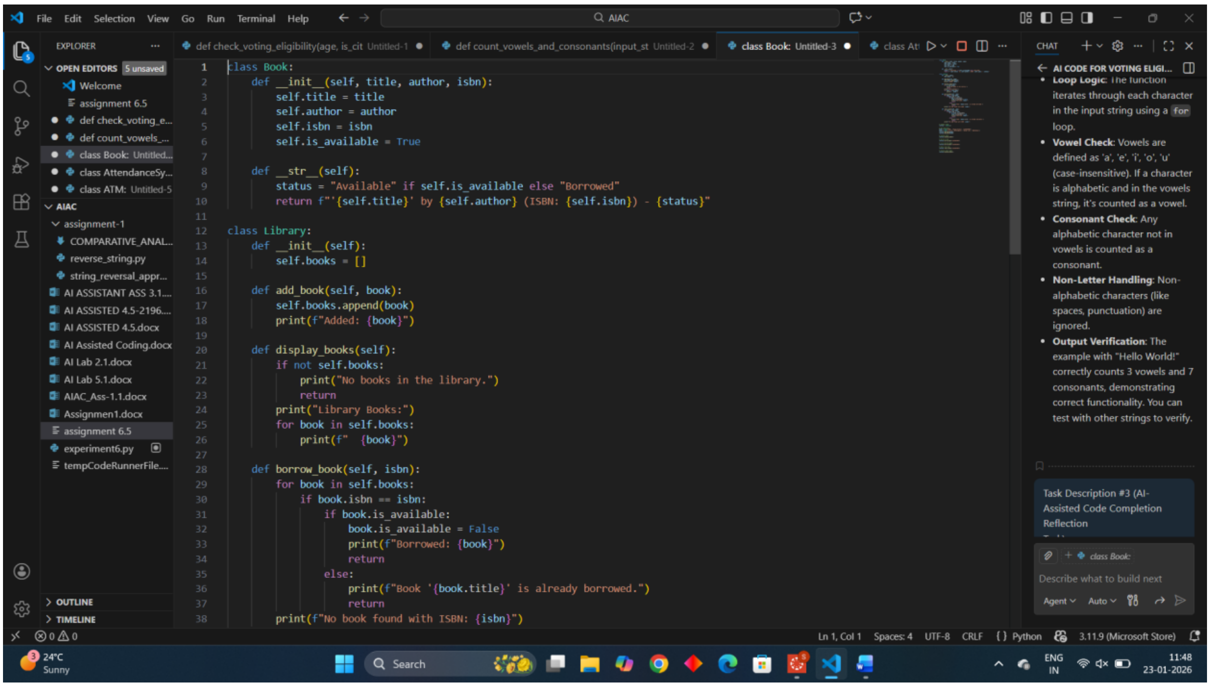
Prompt:

“Generate a Python program for a library management system

using classes, loops, and conditional statements.”

Expected Output:

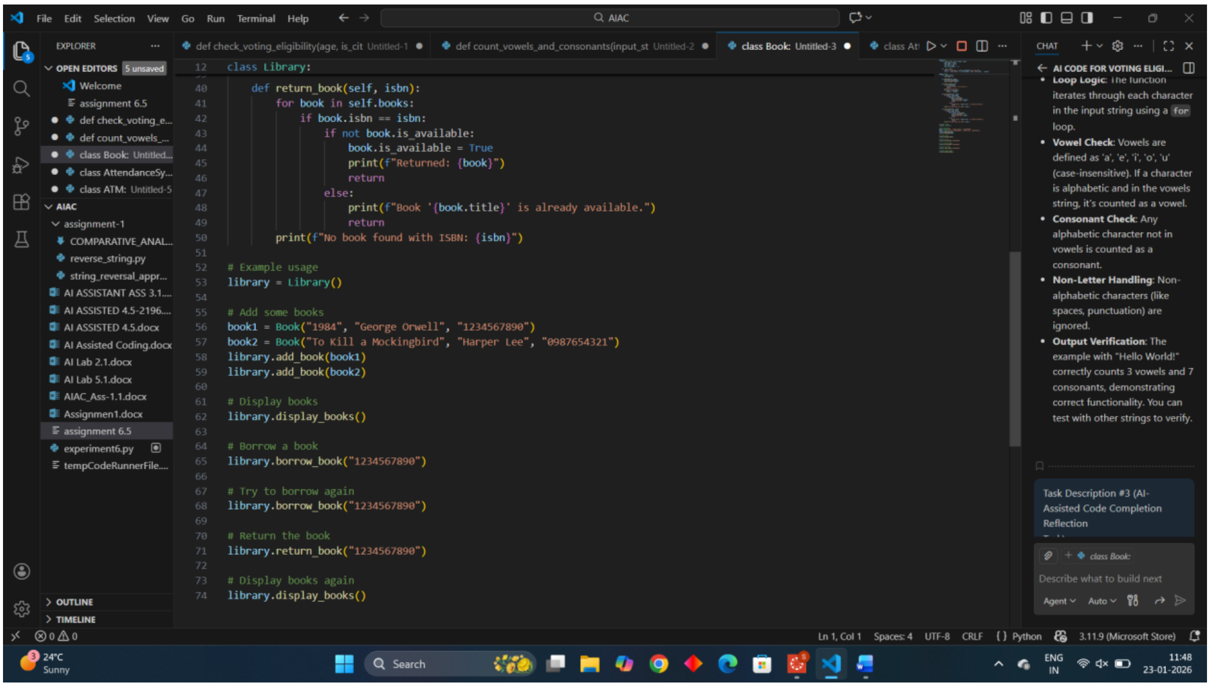
- Complete AI-generated program.
- Review of AI suggestions quality.
- Short reflection on AI-assisted coding experience.



```python
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_available = True

    def __str__(self):
        status = "Available" if self.is_available else "Borrowed"
        return f"'{self.title}' by {self.author} (ISBN: {self.isbn}) - {status}"

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

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

    def display_books(self):
        if not self.books:
            print("No books in the library.")
            return
        print("Library Books:")
        for book in self.books:
            print(f"  {book}")

    def borrow_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                if book.is_available:
                    book.is_available = False
                    print(f"Borrowed: {book}")
                    return
                else:
                    print(f"Book '{book.title}' is already borrowed.")
                    return
        print(f"No book found with ISBN: {isbn}")
```

- **Loop Logic**: The function iterates through each character in the input string using a `for` loop.
- **Vowel Check**: Vowels are defined as 'a', 'e', 'i', 'o', 'u' (case-insensitive). If a character is alphabetic and in the vowels string, it's counted as a vowel.
- **Consonant Check**: Any alphabetic character not in vowels is counted as a consonant.
- **Non-Letter Handling**: Non-alphabetic characters (like spaces, punctuation) are ignored.
- **Output Verification**: The example with "Hello World!" correctly counts 3 vowels and 7 consonants, demonstrating correct functionality. You can test with other strings to verify.

Task Description #3 (AI-Assisted Code Completion Reflection



```python
class Library:
    def return_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                if not book.is_available:
                    book.is_available = True
                    print(f"Returned: {book}")
                    return
                else:
                    print(f"Book '{book.title}' is already available.")
                    return
        print(f"No book found with ISBN: {isbn}")

# Example usage
library = Library()

# Add some books
book1 = Book("1984", "George Orwell", "1234567890")
book2 = Book("To Kill a Mockingbird", "Harper Lee", "0987654321")
library.add_book(book1)
library.add_book(book2)

# Display books
library.display_books()

# Borrow a book
library.borrow_book("1234567890")

# Try to borrow again
library.borrow_book("1234567890")

# Return the book
library.return_book("1234567890")

# Display books again
library.display_books()
```

- **Loop Logic**: The function iterates through each character in the input string using a `for` loop.
- **Vowel Check**: Vowels are defined as 'a', 'e', 'i', 'o', 'u' (case-insensitive). If a character is alphabetic and in the vowels string, it's counted as a vowel.
- **Consonant Check**: Any alphabetic character not in vowels is counted as a consonant.
- **Non-Letter Handling**: Non-alphabetic characters (like spaces, punctuation) are ignored.
- **Output Verification**: The example with "Hello World!" correctly counts 3 vowels and 7 consonants, demonstrating correct functionality. You can test with other strings to verify.

Task Description #3 (AI-Assisted Code Completion Reflection

The generated code is complete, functional, and well-structured. It uses classes effectively (Book and Library), incorporates loops for iterating through books, and employs conditionals for checking availability and handling operations. The code is readable, includes docstrings and comments, and handles edge cases like non-existent books or already borrowed/returned books. It demonstrates good object-oriented principles and provides clear output for verification.

Short Reflection on AI-Assisted Coding Experience:

AI-assisted coding streamlines the development process by quickly generating boilerplate code and suggesting logical structures, allowing focus on higher-level design. However, it's essential to review and test the output to ensure correctness and alignment with specific requirements, as AI can sometimes miss nuanced edge cases or optimizations. Overall, it enhances productivity while requiring human oversight for quality assurance.

Task Description #4 (AI-Assisted Code Completion for Class-

Based Attendance System)

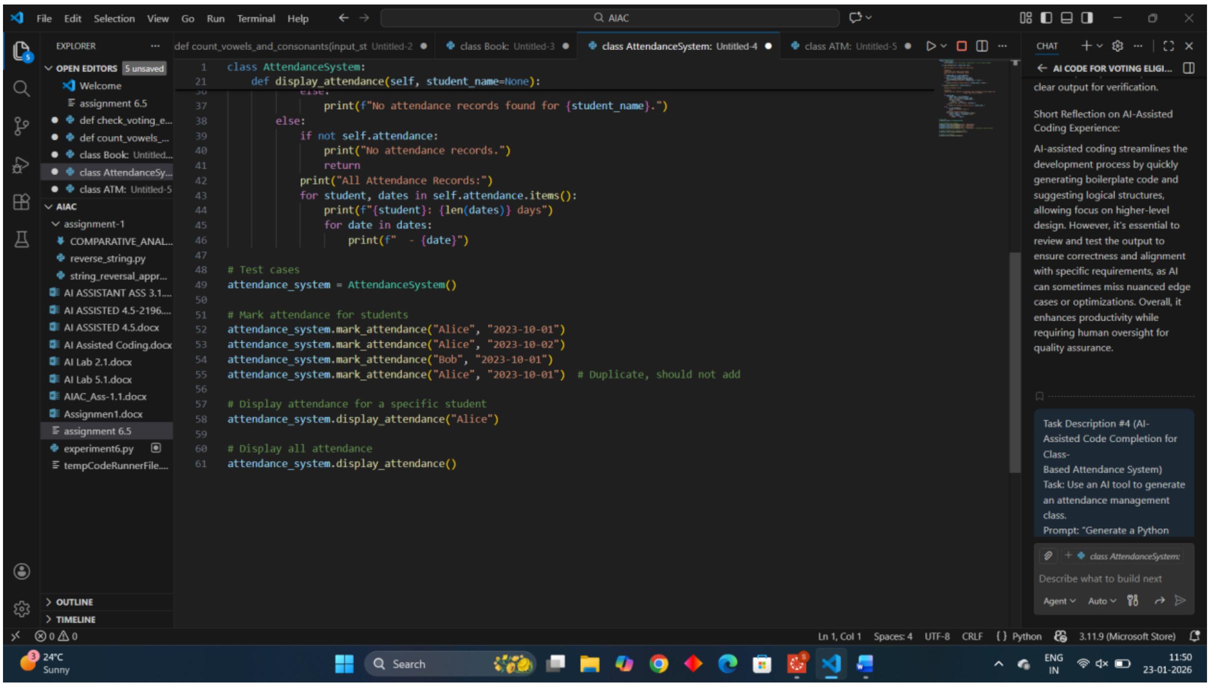
Task: Use an AI tool to generate an attendance management class.

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.



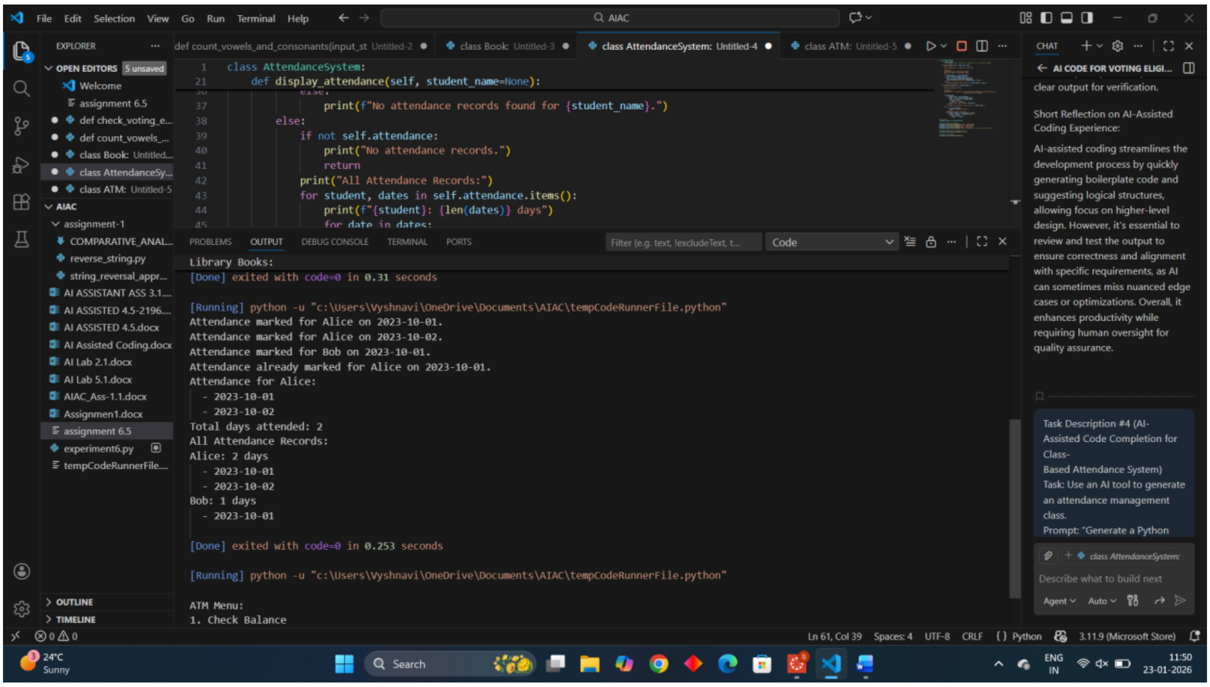
```python
class AttendanceSystem:
    def display_attendance(self, student_name=None):
                print(f"No attendance records found for {student_name}.")
        else:
            if not self.attendance:
                print("No attendance records.")
                return
            print("All Attendance Records:")
            for student, dates in self.attendance.items():
                print(f"{student}: {len(dates)} days")
                for date in dates:
                    print(f"  - {date}")

# Test cases
attendance_system = AttendanceSystem()

# Mark attendance for students
attendance_system.mark_attendance("Alice", "2023-10-01")
attendance_system.mark_attendance("Alice", "2023-10-02")
attendance_system.mark_attendance("Bob", "2023-10-01")
attendance_system.mark_attendance("Alice", "2023-10-01")  # Duplicate, should not add

# Display attendance for a specific student
attendance_system.display_attendance("Alice")

# Display all attendance
attendance_system.display_attendance()
```

clear output for verification.

Short Reflection on AI-Assisted Coding Experience:

AI-assisted coding streamlines the development process by quickly generating boilerplate code and suggesting logical structures, allowing focus on higher-level design. However, it's essential to review and test the output to ensure correctness and alignment with specific requirements, as AI can sometimes miss nuanced edge cases or optimizations. Overall, it enhances productivity while requiring human oversight for quality assurance.

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



```python
class AttendanceSystem:
    def display_attendance(self, student_name=None):
                print(f"No attendance records found for {student_name}.")
        else:
            if not self.attendance:
                print("No attendance records.")
                return
            print("All Attendance Records:")
            for student, dates in self.attendance.items():
                print(f"{student}: {len(dates)} days")
```

```
Library Books:
[Done] exited with code=0 in 0.31 seconds

[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"
Attendance marked for Alice on 2023-10-01.
Attendance marked for Alice on 2023-10-02.
Attendance marked for Bob on 2023-10-01.
Attendance already marked for Alice on 2023-10-01.
Attendance for Alice:
  - 2023-10-01
  - 2023-10-02
Total days attended: 2
All Attendance Records:
Alice: 2 days
  - 2023-10-01
  - 2023-10-02
Bob: 1 days
  - 2023-10-01

[Done] exited with code=0 in 0.253 seconds

[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"

ATM Menu:
1. Check Balance
```

clear output for verification.

Short Reflection on AI-Assisted Coding Experience:

AI-assisted coding streamlines the development process by quickly generating boilerplate code and suggesting logical structures, allowing focus on higher-level design. However, it's essential to review and test the output to ensure correctness and alignment with specific requirements, as AI can sometimes miss nuanced edge cases or optimizations. Overall, it enhances productivity while requiring human oversight for quality assurance.

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

Explanation:

- **Mark Attendance**: Uses a loop implicitly in list operations, but the main logic checks for duplicates.
- **Display Attendance**: Uses loops to iterate through dates for a student or through all students and their dates.

- **Test Cases**: The example code demonstrates marking attendance, handling duplicates, and displaying records, verifying correct functionality. For "Alice", it shows 2 days attended; for "Bob", 1 day. The duplicate mark is handled gracefully.

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

Menu Navigation)

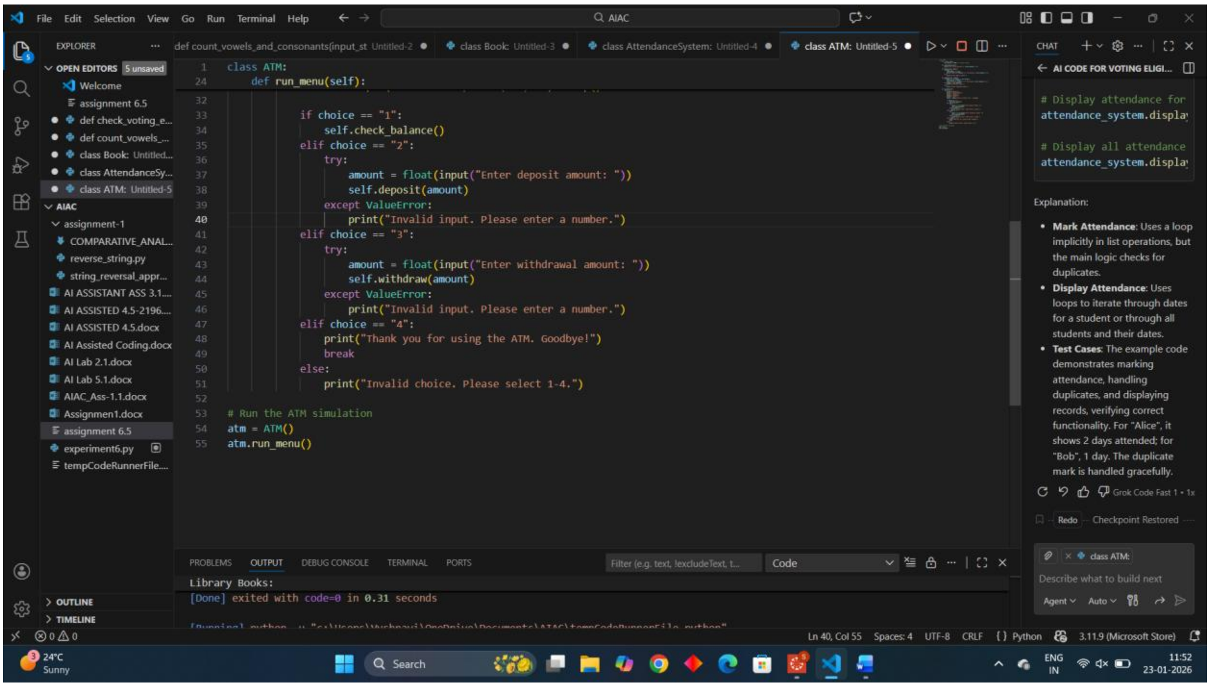
Task: Use an AI tool to complete a navigation menu.

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.



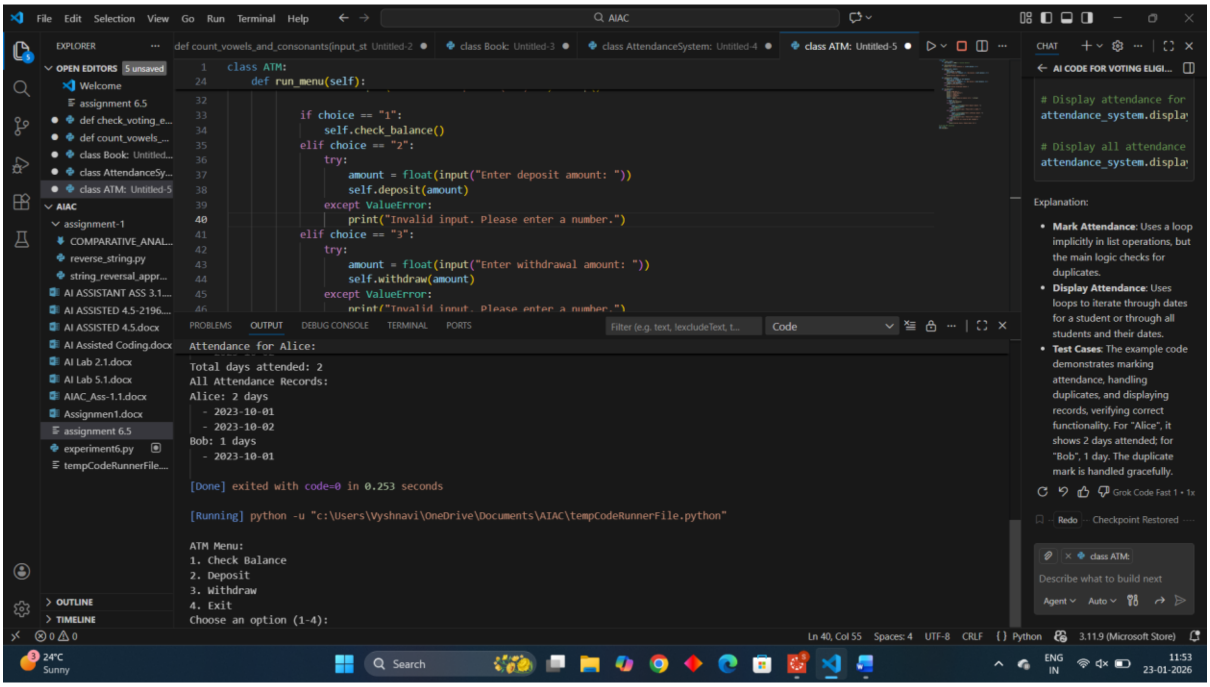
```python
class ATM:
    def run_menu(self):

            if choice == "1":
                self.check_balance()
            elif choice == "2":
                try:
                    amount = float(input("Enter deposit amount: "))
                    self.deposit(amount)
                except ValueError:
                    print("Invalid input. Please enter a number.")
            elif choice == "3":
                try:
                    amount = float(input("Enter withdrawal amount: "))
                    self.withdraw(amount)
                except ValueError:
                    print("Invalid input. Please enter a number.")
            elif choice == "4":
                print("Thank you for using the ATM. Goodbye!")
                break
            else:
                print("Invalid choice. Please select 1-4.")

# Run the ATM simulation
atm = ATM()
atm.run_menu()
```

```
Library Books:
[Done] exited with code=0 in 0.31 seconds
```

```python
# Display attendance for
attendance_system.displa

# Display all attendance
attendance_system.displa
```

Explanation:

- **Mark Attendance**: Uses a loop implicitly in list operations, but the main logic checks for duplicates.
- **Display Attendance**: Uses loops to iterate through dates for a student or through all students and their dates.
- **Test Cases**: The example code demonstrates marking attendance, handling duplicates, and displaying records, verifying correct functionality. For "Alice", it shows 2 days attended; for "Bob", 1 day. The duplicate mark is handled gracefully.



```python
class ATM:
    def run_menu(self):

            if choice == "1":
                self.check_balance()
            elif choice == "2":
                try:
                    amount = float(input("Enter deposit amount: "))
                    self.deposit(amount)
                except ValueError:
                    print("Invalid input. Please enter a number.")
            elif choice == "3":
                try:
                    amount = float(input("Enter withdrawal amount: "))
                    self.withdraw(amount)
                except ValueError:
                    print("Invalid input. Please enter a number.")
```

```
Attendance for Alice:
Total days attended: 2
All Attendance Records:
Alice: 2 days
  - 2023-10-01
  - 2023-10-02
Bob: 1 days
  - 2023-10-01

[Done] exited with code=0 in 0.253 seconds

[Running] python -u "c:\Users\Vyshnavi\OneDrive\Documents\AIAC\tempCodeRunnerFile.python"

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose an option (1-4):
```

```python
# Display attendance for
attendance_system.displa

# Display all attendance
attendance_system.displa
```

Explanation:

- **Mark Attendance**: Uses a loop implicitly in list operations, but the main logic checks for duplicates.
- **Display Attendance**: Uses loops to iterate through dates for a student or through all students and their dates.
- **Test Cases**: The example code demonstrates marking attendance, handling duplicates, and displaying records, verifying correct functionality. For "Alice", it shows 2 days attended; for "Bob", 1 day. The duplicate mark is handled gracefully.

- **Loop**: The run_menu method uses a while True loop to continuously display the menu until the user chooses to exit.
- **Conditionals**: if-elif-else statements handle different menu choices (1-4), with input validation for amounts using try-except to catch invalid numbers.

- **Option Handling**: Each option calls the appropriate method (check_balance, deposit, withdraw) or exits the loop. Invalid choices are handled gracefully.