

ASSIGNMENT 6.5

Atla Sreeja

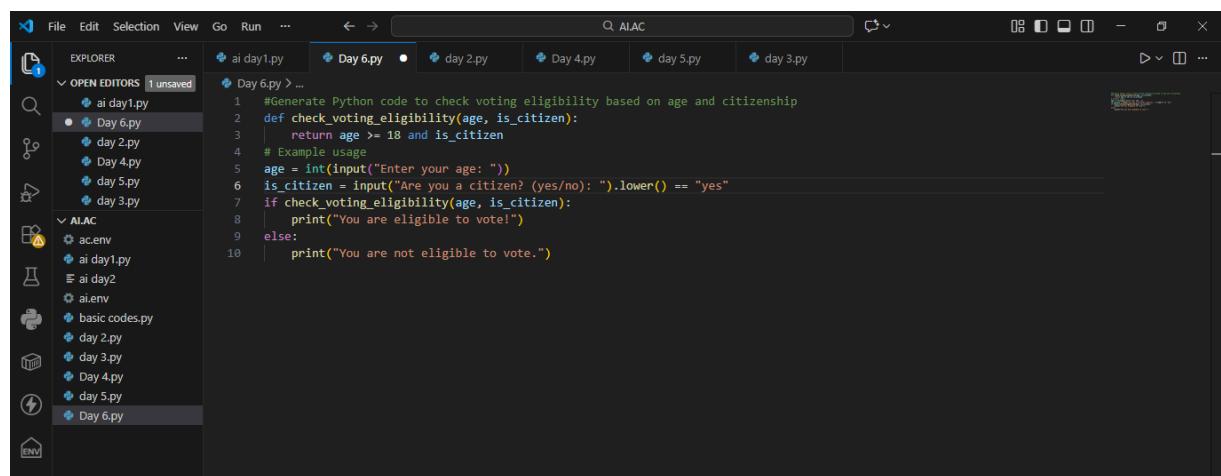
2403A51L02

B-51

AI-Based Code Completion: Working with suggestions for classes, loops, conditionals
Task Description-1: Zero-shot Prompting

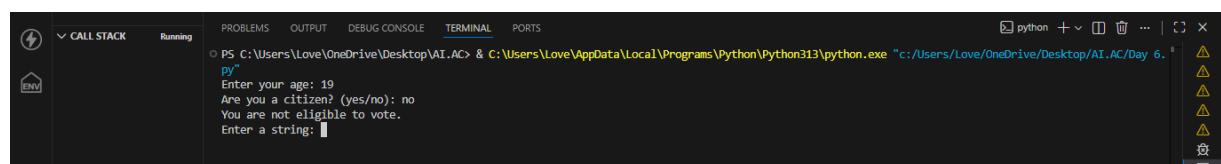
Task 1: Conditional Eligibility Check

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



```
#Generate Python code to check voting eligibility based on age and citizenship
def check_voting_eligibility(age, is_citizen):
    return age >= 18 and is_citizen
# Example usage
age = int(input("Enter your age: "))
is_citizen = input("Are you a citizen? (yes/no): ").lower() == "yes"
if check_voting_eligibility(age, is_citizen):
    print("You are eligible to vote!")
else:
    print("You are not eligible to vote.")
```

OUTPUT:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 6.py"
Enter your age: 19
Are you a citizen? (yes/no): no
You are not eligible to vote.
Enter a string: 
```

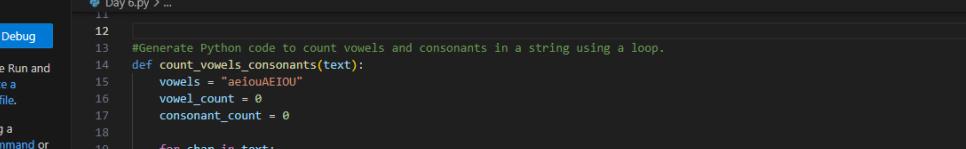
Explanation:

- `age = int(input())` → Takes age as integer input
- `citizen = input().lower()` → Converts input to lowercase for consistency
- `if age >= 18 and citizen == "yes":`

- Checks both age condition and citizenship
 - Prints eligibility result accordingly

Task2: Loop-Based String Processing

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



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure with files like "Day 1.py", "Day 6.py", "Day 2.py", "Day 4.py", "Day 5.py", and "day 3.py".
- Search Bar:** Contains the text "Q AIAC".
- Run and Debug View:** A sidebar titled "RUN" with a "Run and Debug" button highlighted in blue.
- Terminal:** Displays the command "Debug using a terminal command or in an interactive chat."
- Code Editor:** The main area contains Python code for counting vowels and consonants in a string. The code uses a loop to iterate through each character, checks if it's a vowel or consonant, and increments the respective count. It includes a comment explaining the purpose and an example usage at the bottom.
- Right Panel:** Shows the Python file "Day 6.py" with various analysis and navigation tools.

```
#Generate Python code to count vowels and consonants in a string using a loop.
def count_vowels_consonants(text):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in text:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return vowel_count, consonant_count

# Example usage
text = input("Enter a string: ")
vowels, consonants = count_vowels_consonants(text)
print(f"Vowels: {vowels}, Consonants: {consonants}")
```

OUTPUT:



The screenshot shows a terminal window with the following content:

```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 6.py"
Enter a string: keerthi
Vowels: 3, Consonants: 4
```

Explanation:

- Converts string to lowercase
 - Loops through each character
 - `isalpha()` ensures only letters are counted
 - Separates vowels and consonants logically

Task 3: AI-Assisted Code Completion Reflection Task

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

The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, and a search bar labeled 'AIAC'. Below the menu is a toolbar with icons for RUN AND DEBUG, RUN, and various file operations. A sidebar on the left contains sections for RUN (Run and Debug, To customize Run and Debug), BREAKPOINTS (Raised Exception, Uncaught Exceptions, User Uncaught), and ENV. The main code editor area displays the following Python code:

```
35 #Generate a Python program for a library management system using classes, loops, and conditional statements
36 class Book:
37     def __init__(self, title, author, isbn, available=True):
38         self.title = title
39         self.author = author
40         self.isbn = isbn
41         self.available = available
42
43 class Library:
44     def __init__(self):
45         self.books = []
46
47     def add_book(self, book):
48         self.books.append(book)
49         print(f"Book '{book.title}' added to library.")
50
51     def search_book(self, title):
52         for book in self.books:
53             if book.title.lower() == title.lower():
54                 return book
55         return None
56
57     def borrow_book(self, title):
58         book = self.search_book(title)
59         if book is None:
60             print("Book not found.")
61         elif not book.available:
62             print(f"'{book.title}' is not available.")
63         else:
64             book.available = False
65             print(f"You borrowed '{book.title}'")
66
67     def return_book(self, title):
68         book = self.search_book(title)
69         if book is None:
70             print("Book not found.")
71         elif book.available:
72             print(f"'{book.title}' was not borrowed.")
73         else:
74             book.available = True
75             print(f"You returned '{book.title}'")
76
77     def list_books(self):
78         if not self.books:
79             print("Library is empty.")
80         else:
81             for book in self.books:
82                 status = "Available" if book.available else "Borrowed"
83                 print(f"{book.title} by {book.author} - {status}")
84
85 # Example usage
86 library = Library()
87 library.add_book(Book("Python Basics", "John Doe", "123"))
88 library.add_book(Book("Web Development", "Jane Smith", "456"))
89 library.borrow_book("Python Basics")
90 library.list_books()
91 library.return_book("Python Basics")
```

The code defines a `Book` class with attributes `title`, `author`, `isbn`, and `available`. It also defines a `Library` class with methods for adding books, searching for books by title, borrowing books (which marks them as unavailable), returning books (which marks them as available again), and listing all books in the library. The code uses loops and conditional statements to implement these features. The AI-assisted code completion feature is shown at the top of the code editor, suggesting the next line of code.

This screenshot shows the same code editor interface after the AI-assisted code completion task has been completed. The code editor now displays the full Python program for the library management system. The code is identical to the one shown in the previous screenshot, but the AI-assisted completion feature is no longer visible at the top of the editor.

OUTPUT:

The screenshot shows a Python debugger interface with a terminal window displaying the following command-line session:

```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 6.py"
Book 'Python Basics' added to library.
Book 'Web Development' added to library.
You borrowed 'Python Basics'.
Python Basics by John Doe - Borrowed
Web Development by Jane Smith - Available
You returned 'Python Basics'.
```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected. The status bar at the bottom indicates "Indexing completed".

Explanation:

AI-assisted coding helped speed up development and provided a structured solution. However, human review was necessary to improve input validation, error handling, and real-world usability. Responsible AI use requires understanding and modifying generated code rather than copying blindly.

Task 4: Class-Based Attendance System

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

The screenshot shows a Python debugger interface with a code editor containing the following Python code:

```
#Generate a Python class to mark and display student attendance using loops.
class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id
        self.attendance = []

    def mark_attendance(self, present):
        self.attendance.append(present)

    def get_attendance_percentage(self):
        if not self.attendance:
            return 0
        present_count = sum(1 for day in self.attendance if day)
        return (present_count / len(self.attendance)) * 100

class AttendanceSystem:
    def __init__(self):
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def mark_attendance(self, student_id, attendance_list):
        for student in self.students:
            if student.student_id == student_id:
                for present in attendance_list:
                    student.mark_attendance(present)
                return
        print("Student not found.")

    def display_attendance(self):
```

```
108
109     class AttendanceSystem:
110         def __init__(self):
111             self.students = []
112
113         def add_student(self, student):
114             self.students.append(student)
115
116         def mark_attendance(self, student_id, attendance_list):
117             for student in self.students:
118                 if student.student_id == student_id:
119                     for present in attendance_list:
120                         student.mark_attendance(present)
121
122             return
123
124         def display_attendance(self):
125             if not self.students:
126                 print("No students in the system.")
127             else:
128                 for student in self.students:
129                     percentage = student.get_attendance_percentage()
130                     print(f"{student.name} ({student.student_id}): {percentage:.2f}% attendance")
131
132     # Example usage
133     system = AttendanceSystem()
134     system.add_student(Student("Alice", "S001"))
135     system.add_student(Student("Bob", "S002"))
136     system.mark_attendance("S001", [True, True, False, True, True])
137     system.mark_attendance("S002", [True, False, True, False, True])
138     system.display_attendance()
```

OUTPUT:

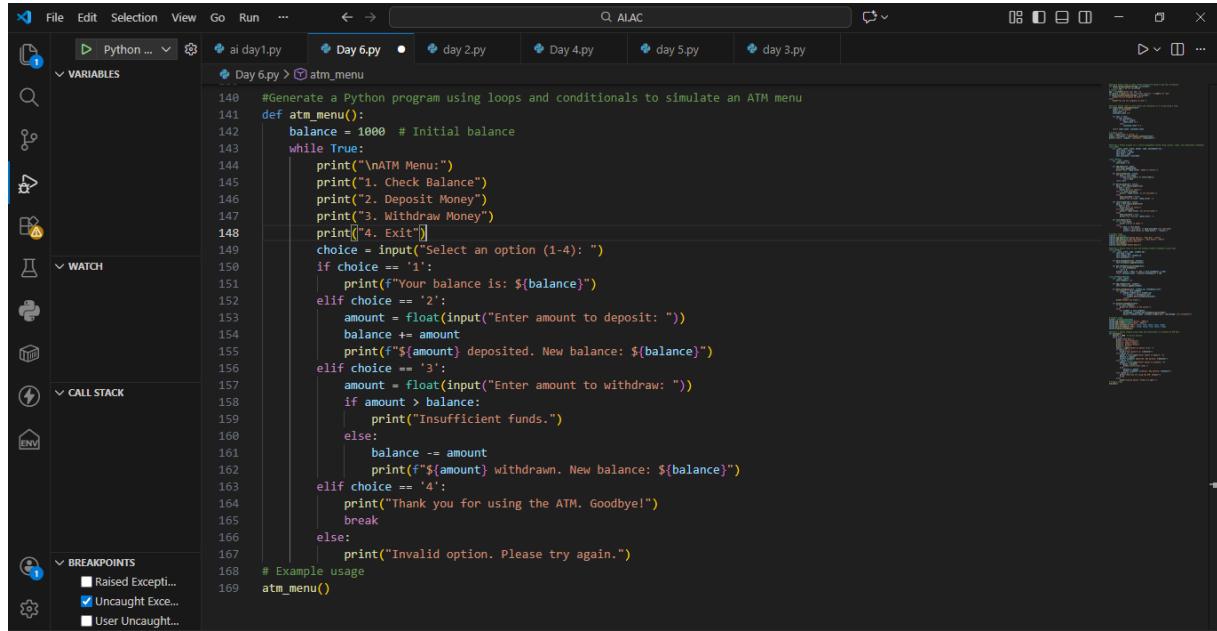
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 6.py"
Alice (S001): 80.00% attendance
Bob (S002): 60.00% attendance
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation:

- Dictionary stores student name and attendance
- Loop iterates through records
- Simple and efficient design

Task 5: Conditional Menu Navigation (ATM Menu)

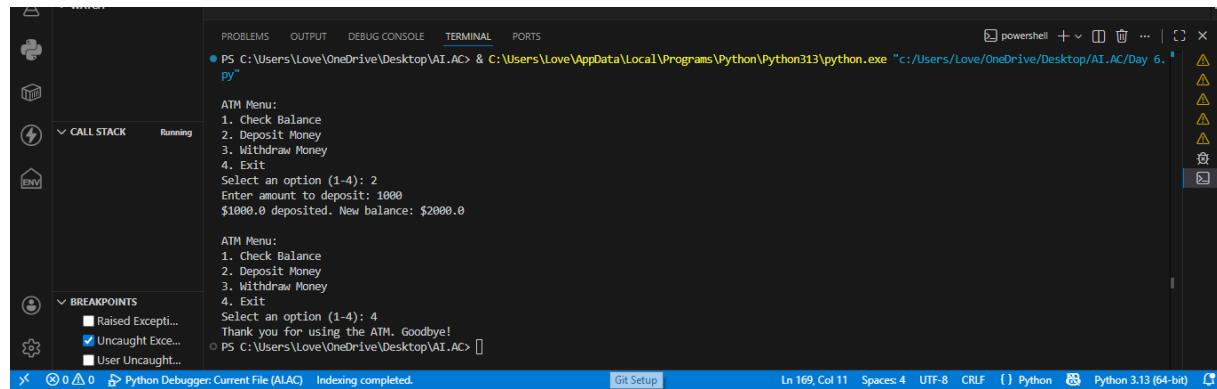
Prompt: Generate a Python program using loops and conditionals to simulate an ATM menu.



```
#Generate a Python program using loops and conditionals to simulate an ATM menu
def atm_menu():
    balance = 1000 # Initial balance
    while True:
        print("\nATM Menu:")
        print("1. Check Balance")
        print("2. Deposit Money")
        print("3. Withdraw Money")
        print("4. Exit")
        choice = input("Select an option (1-4): ")
        if choice == '1':
            print("Your balance is: ${balance}")
        elif choice == '2':
            amount = float(input("Enter amount to deposit: "))
            balance += amount
            print("${amount} deposited. New balance: ${balance}")
        elif choice == '3':
            amount = float(input("Enter amount to withdraw: "))
            if amount > balance:
                print("Insufficient funds.")
            else:
                balance -= amount
                print("${amount} withdrawn. New balance: ${balance}")
        elif choice == '4':
            print("Thank you for using the ATM. Goodbye!")
            break
        else:
            print("Invalid option. Please try again.")

# Example usage
atm_menu()
```

OUTPUT:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 6.py"
ATM Menu:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Select an option (1-4): 2
Enter amount to deposit: 1000
$1000.0 deposited. New balance: $2000.0

ATM Menu:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Select an option (1-4): 4
Thank you for using the ATM. Goodbye!
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation:

- Correct balance update
- Prevents overdraft
- Loop exits safely