

A.Abhiram

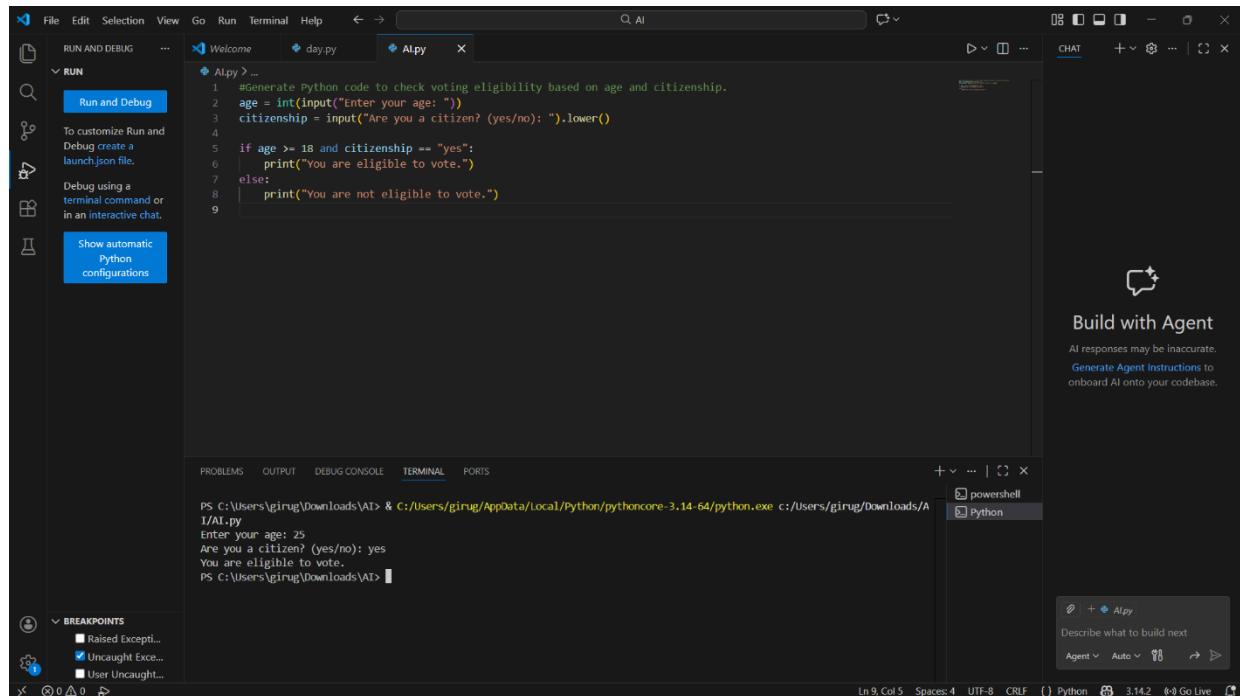
2403a51l13

Batch-51

## Lab 6: 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.



```
#Generate Python code to check voting eligibility based on age and citizenship.
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.")
else:
    print("You are not eligible to vote.")
```

### Justification:

It checks voting eligibility using conditional statements. It verifies whether the user's age is 18 or above and if the person is a citizen. Based on these conditions, it displays eligibility status.

`age >= 18` → Checks minimum voting age.

`citizen == "yes"` → Ensures citizenship.

`and` → Both conditions must be true.

`if-else` → Makes the eligibility decision.

## 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.

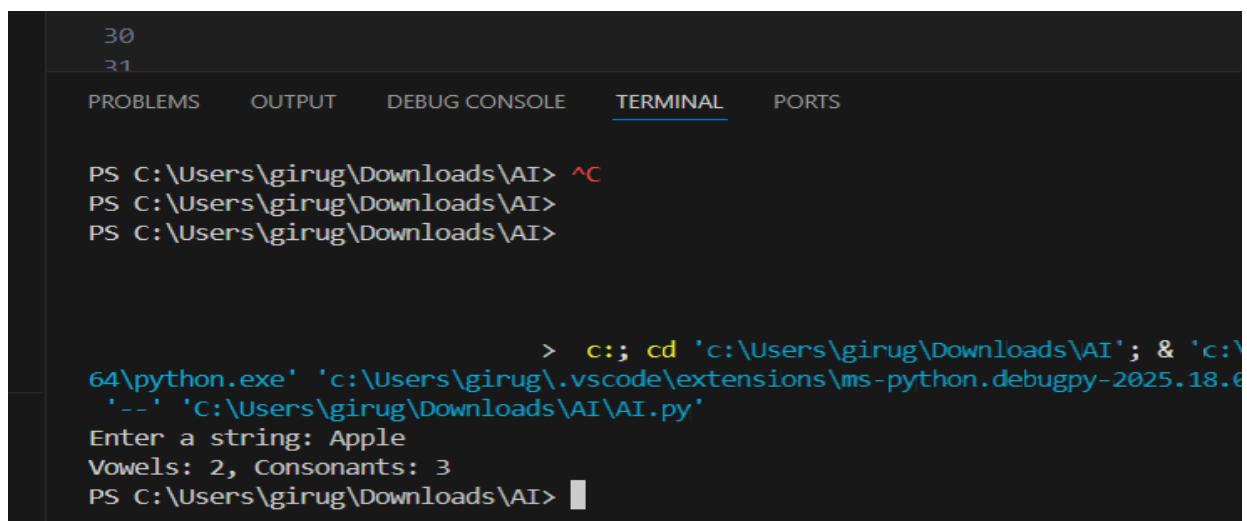
### AI-Generated Code:

```
#task2
#“Generate 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 a letter
            if char in vowels:
                vowel_count += 1 # Increment vowel count
            else:
                consonant_count += 1 # Increment consonant count

    return vowel_count, consonant_count
input_string = input("Enter a string: ")
vowels, consonants = count_vowels_consonants(input_string)
print(f"Vowels: {vowels}, Consonants: {consonants}")
```

### OUTPUT:



```
PS C:\Users\girug\Downloads\AI> ^C
PS C:\Users\girug\Downloads\AI>
PS C:\Users\girug\Downloads\AI>

                                     > c;; cd 'c:\users\girug\downloads\AI'; & 'c:\v
64\python.exe' 'c:\users\girug\.vscode\extensions\ms-python.debugpy-2025.18.6
' '--' 'c:\users\girug\downloads\AI\AI.py'
Enter a string: Apple
Vowels: 2, Consonants: 3
PS C:\Users\girug\Downloads\AI>
```

### Justification:

This task uses AI-generated Python code to count vowels and consonants in a given string using a loop. The program reads a string input from the user and

converts it to lowercase to avoid case sensitivity. A for loop iterates through each character in the string. If it is a vowel, the vowel count is incremented; otherwise, the consonant count is increased. This task demonstrates the use of loops and nested conditionals for string processing and character analysis.

## 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.

The image shows two side-by-side screenshots of a code editor interface, likely Visual Studio Code, demonstrating AI-assisted code completion. Both screenshots show the same Python code for a library management system, with the bottom one showing the execution results in the terminal.

```

File Edit Selection View Go Run Terminal Help <- > AI
RUN AND DEBUG ... Welcome day.py Alpy
RUN
To customize Run and Debug create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations
RUN AND DEBUG ... Welcome day.py Alpy
RUN
To customize Run and Debug create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations

```

**Code (Top Screenshot):**

```

#task3
#Generate a Python program for a library management system using classes, loops, and conditional statements.
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_borrowed = False
class Library:
    def __init__(self):
        self.books = []
    def add_book(self, book):
        self.books.append(book)
    def display_books(self):
        for book in self.books:
            status = "Borrowed" if book.is_borrowed else "Available"
            print(f"{book.title} by {book.author} - {status}")
    def borrow_book(self, title):
        for book in self.books:
            if book.title == title and not book.is_borrowed:
                book.is_borrowed = True
                print(f"You have borrowed '{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 book.is_borrowed:
                book.is_borrowed = False
                print(f"You have returned '{book.title}'")
                return
        print(f"'{title}' was not borrowed.")
# Example usage
library = Library()
library.add_book(Book("1984", "George Orwell"))
library.add_book(Book("To Kill a Mockingbird", "Harper Lee"))
library.display_books()

```

**Code (Bottom Screenshot):**

```

class Library:
    def return_book(self, title):
        print(f"'{title}' was not borrowed.")
# Example usage
library = Library()
library.add_book(Book("1984", "George Orwell"))
library.add_book(Book("To Kill a Mockingbird", "Harper Lee"))
library.display_books()
library.borrow_book("1984")
library.display_books()
library.return_book("1984")
library.display_books()

```

**Terminal Output (Bottom Screenshot):**

```

... 'C:\Users\girug\Downloads\AI\AI.py'
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
You have borrowed '1984'.
1984 by George Orwell - Borrowed
To Kill a Mockingbird by Harper Lee - Available
You have returned '1984'.
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
You have returned '1984'.
1984 by George Orwell - Available
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
You have returned '1984'.
1984 by George Orwell - Available
You have returned '1984'.
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
PS C:\Users\girug\Downloads\AI> []

```

## OUTPUT:

The screenshot shows the Visual Studio Code interface with an AI-assisted development extension. The left sidebar has sections for RUN AND DEBUG, RUN, and Python configurations. The main editor shows a Python file named 'day.py' with the following code:

```
36     class Library:
37         def return_book(self, title):
38             print(f"'{title}' was not borrowed.")
39
40             # Example usage
41             library = Library()
42             library.add_book(book("1984", "George Orwell"))
43             library.display_books()
44             library.borrow_book("1984")
45             library.display_books()
46             library.return_book("1984")
47             library.display_books()
```

The terminal below shows the execution of the code, displaying the state of the library:

```
'--> C:\Users\girug\Downloads\AI\AI.py'
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
You have borrowed '1984'.
1984 by George Orwell - Borrowed
To Kill a Mockingbird by Harper Lee - Available
You have returned '1984'.
1984 by George Orwell - Available
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
You have borrowed '1984'.
1984 by George Orwell - Borrowed
To Kill a Mockingbird by Harper Lee - Available
You have returned '1984'.
1984 by George Orwell - Available
1984 by George Orwell - Borrowed
To Kill a Mockingbird by Harper Lee - Available
You have returned '1984'.
1984 by George Orwell - Available
You have returned '1984'.
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
PS C:\Users\girug\Downloads\AI> []
```

The status bar at the bottom indicates the code is in Python mode, with line 72, column 1, and other details.

**Justification:** 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 Description – 4: AI-Assisted Code Completion for Class-

Based Attendance System

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

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

- File Explorer:** Shows a folder structure with files like `Alpy`, `day.py`, and `Alpy >...`.
- Code Editor:** Displays Python code for a `Student` class with methods for marking attendance and displaying results.
- Terminal:** Shows the output of running the code, which prints the attendance status for a student named Alice over three days.
- Breakpoints:** A sidebar shows breakpoints for the current file.
- Output:** Shows standard output from the terminal.
- Terminal:** Shows the command to run the script and its output.
- Python Environment:** Shows the Python environment configuration.
- Build with Agent:** A sidebar for AI integration.

```

73 #task4
74 #Generate a Python class to mark and display student attendance using loops.
75 class Student:
76     def __init__(self, name):
77         self.name = name
78         self.attendance = []
79
80     def mark_attendance(self, status):
81         self.attendance.append(status)
82
83     def display_attendance(self):
84         print(f"Attendance for {self.name}:")
85         for i, status in enumerate(self.attendance):
86             print(f"Day {i+1}: {status}")
87
88 # Example usage
89 student = Student("Alice")
90 student.mark_attendance("Present")
91 student.mark_attendance("Absent")
92 student.mark_attendance("Present")
93 student.display_attendance()
94

```

## OUTPUT:

This screenshot is identical to the one above, showing the same code execution and output in the terminal.

## Observations:

- Dictionary stores student name and attendance

- Loop iterates through records
- Simple and efficient design

## 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.

The screenshot shows a code editor window with the following details:

- File:** day.py
- Content:**

```

95     #task5
96     #generate a Python program using loops and conditionals to simulate an ATM menu
97     def atm_menu():
98         balance = 1000 # Initial balance
99         while True:
100             print("\nATM Menu:")
101             print("1. Check Balance")
102             print("2. Deposit Money")
103             print("3. Withdraw Money")
104             print("4. Exit")
105
106             choice = input("Choose an option (1-4): ")
107
108             if choice == '1':
109                 print(f"Your current balance is: ${balance}")
110             elif choice == '2':
111                 amount = float(input("Enter amount to deposit: "))
112                 if amount > 0:
113                     balance += amount
114                     print(f"${amount} deposited successfully.")
115                 else:
116                     print("Invalid amount. Please enter a positive number.")
117             elif choice == '3':
118                 amount = float(input("Enter amount to withdraw: "))
119                 if 0 < amount <= balance:
120                     balance -= amount
121                     print(f"${amount} withdrawn successfully.")
122                 else:
123                     print("Invalid amount or insufficient funds.")
124             elif choice == '4':
125                 print("Thank you for using the ATM. Goodbye!")
126                 break
127             else:
128                 print("Invalid choice. Please select a valid option.")
129
atm_menu()

```
- Toolbars and Panels:**
  - VARIABLES panel on the left.
  - WATCH panel below VARIABLES.
  - CALL STACK panel below WATCH, showing "Running".
  - BREAKPOINTS panel at the bottom left.
  - PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS tabs at the bottom center.
  - AI panel on the right with a "Build with Agent" button and instructions.
  - Terminal tab at the bottom right showing "powershell" and "Python" options.
- Status Bar:** Shows "Ln 130, Col 1" and various file and terminal settings.

## OUTPUT:

```
ATM Menu:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
choose an option (1-4): 2
Enter amount to deposit: 20
$20.0 deposited successfully.

ATM Menu:
1. Check Balance
2. Deposit Money
choose an option (1-4): 2
Enter amount to deposit: 20
$20.0 deposited successfully.

ATM Menu:
1. Check Balance
2. Deposit Money
ATM Menu:
1. Check Balance
2. Deposit Money
1. Check Balance
2. Deposit Money
2. Deposit Money
3. Withdraw Money
4. Exit
Choose an option (1-4): 4
Thank you for using the ATM. Goodbye!
PS C:\Users\girug\Downloads\ATI>
```

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

- File Explorer:** Shows a folder structure with files like `Alpy`, `__init__.py`, and `day.py`.
- Run View:** Displays the `Alpy` process with its PID.
- Terminal:** Shows the execution of the `Alpy` script, which runs an ATM menu. It performs a deposit of \$20.00 twice and then exits the program.
- Breakpoints:** A sidebar shows breakpoints for the `Alpy` file, with one breakpoint for "Uncaught Exceptions" marked as active.
- AI Assistant:** A sidebar titled "Build with Agent" includes a message about AI responses being inaccurate and a button to "Generate Agent Instructions".
- Status Bar:** Shows the current line (Ln 130), column (Col 1), and file (Python), along with other settings like "Auto" and "CRLF".

## Justification:

- Correct balance update
- Prevents overdraft
- Loop exits safely