

# lab AI assitant assignment 1.4

name:balaji

roll no:2403A51102

batch:06

task:1

Lab 1: Environment Setup – GitHub Copilot and VS Code Integration

## Lab Objectives:

- To install and configure GitHub Copilot in Visual Studio Code.
- To explore AI-assisted code generation using GitHub Copilot.
- To analyze the accuracy and effectiveness of Copilot's code suggestions.

- To understand prompt-based programming using comments and code context

## **Lab Outcomes (LOs):**

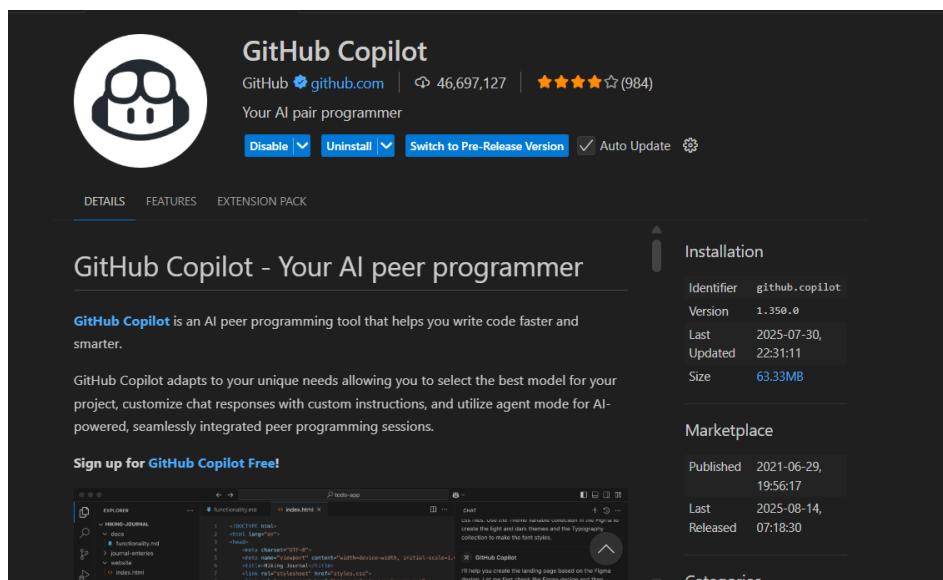
After completing this lab, students will be able to:

- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.

- Compare code suggestions based on different prompts and programming styles.

## Task Description #1

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step



## Expected Output #1

- Successfully install and activate GitHub Copilot in VS Code. Include screenshots showing installation, authentication via

GitHub, and an example suggestion from Copilot.

## Task Description #2

- A function in Python that returns the maximum of three numbers using GitHub Copilot. Use an appropriate comment as a prompt.

## Expected Output #2

- Python function that takes three inputs and returns the largest value. Include the code and output.

The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The workspace contains a folder named 'AI PROGRAMMING' which includes several Python files like 'decimal\_to\_binary.py', 'fibonacci.py', 'factorial.py', etc. The 'fibonacci.py' file is currently selected and shown in the editor. The code defines a function 'greatest\_of\_three(a, b, c)' that returns the maximum of three numbers. It includes a docstring and a print statement for testing. The 'CHAT' panel on the right shows an interaction with GitHub Copilot where it suggests a function to find the greatest item in a list. The 'TERMINAL' panel at the bottom shows the command 'python -u "d:\VAI\programming\fibonacci.py"' being run, followed by user input for three numbers (34, 54, 67) and the resulting output: 'Greatest value is: 67'. The status bar at the bottom indicates the system is running at 28°C and is mostly cloudy.

```
def greatest_of_three(a, b, c):
    """
    Returns the greatest value among three numbers.
    """
    return max(a, b, c)

# Example usage:
x = int(input("Enter first number: "))
y = int(input("Enter second number: "))
z = int(input("Enter third number: "))
print("Greatest value is:", greatest_of_three(x, y, z))
```

```
PS D:\VAI\programming> python -u "d:\VAI\programming\fibonacci.py"
Enter first number: 34
Enter second number: 54
Enter third number: 67
Greatest value is: 67
PS D:\VAI\programming>
```

Observation:

This Python program is designed to find the **greatest value among three input numbers** using the built-in `max()` function.

---

## **How the Program Works:**

### 1. **Function Definition:**

2. `def greatest_of_three(a, b, c):`

3.     `return max(a, b, c)`

- The function `greatest_of_three()` takes three parameters and returns the maximum using Python's built-in `max()` function.

### 4. **User Input:**

- The program prompts the user to enter three numbers:

◦ `x = int(input("Enter first number: "))`

◦ `y = int(input("Enter second number: "))`

◦ `z = int(input("Enter third number: "))`

- The inputs are converted to integers using int().

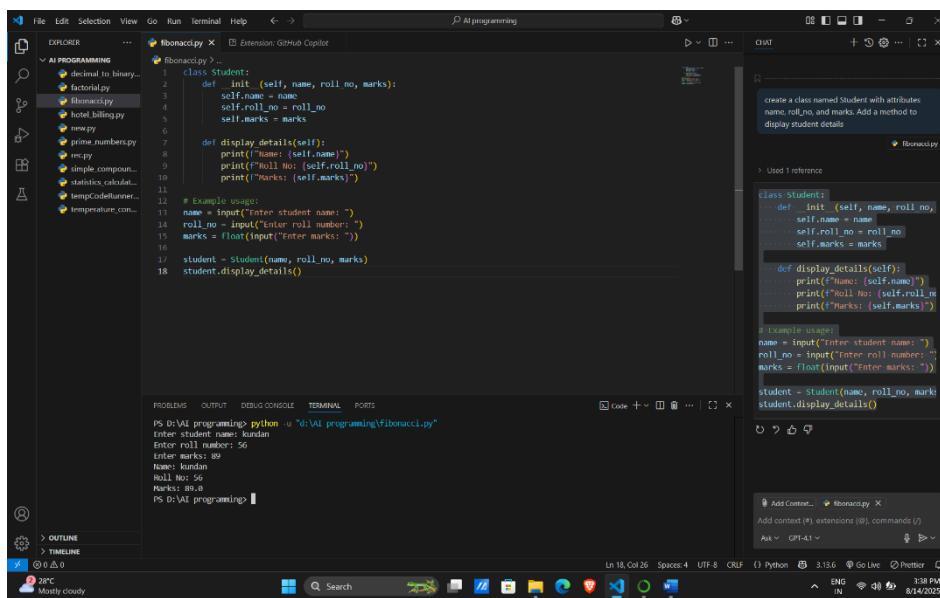
## 5. Function Call and Output:

- The function is called with the three user inputs, and the greatest number is displayed:

```
print("Greatest value is:",
greatest_of_three(x, y, z))
```

### Task Description #4

- Prompt GitHub Copilot to create a class named Student with attributes name, roll\_no, and marks. Add a method to display student details.



The screenshot shows the VS Code interface with the GitHub Copilot extension active. The Explorer sidebar shows several Python files. The main editor area displays the following code for a `Student` class:

```
class Student:
    def __init__(self, name, roll_no, marks):
        self.name = name
        self.roll_no = roll_no
        self.marks = marks

    def display_details(self):
        print("Name: " + self.name)
        print("Roll No: " + str(self.roll_no))
        print("Marks: " + str(self.marks))

# Example usage:
name = input("Enter student name: ")
roll_no = input("Enter roll number: ")
marks = float(input("Enter marks: "))
student = Student(name, roll_no, marks)
student.display_details()
```

The right-hand panel shows the GitHub Copilot interface with the prompt: "create a class named Student with attributes name, roll\_no, and marks. Add a method to display student details". Below the prompt, it says "Used 1 reference" and shows the generated code. At the bottom, there's a terminal window showing the execution of the script and its output.

## **Expected Output #4**

- Python class definition with an initializer and a display method. Include object creation and output.

This Python program demonstrates the use of **object-oriented programming (OOP)** by defining a class named Student. It encapsulates student-related data and provides a method to display it.

Observation:

---



### **Key Functionalities:**

1. **Class Definition:**
2. **class Student:**
  - A class Student is created to model a student with relevant attributes and behaviors.
3. **Constructor Method (`__init__`):**

4. `def __init__(self, name, roll_no, marks):`
5.     `self.name = name`
6.     `self.roll_no = roll_no`
7.     `self.marks = marks`
  - Initializes three instance variables:
    - name: Student's name
    - roll\_no: Roll number
    - marks: Marks obtained (as a float)
8. **Display Method:**
9. `def display_details(self):`
10.    `print("Name:", self.name)`
11.    `print("Roll No:", self.roll_no)`
12.    `print("Marks:", self.marks)`
  - This method prints out the values of the student's attributes.
13. **User Input & Object Creation:**
14. `name = input("Enter student name: ")`

```

15. roll_no = input("Enter roll number: ")
16. marks = float(input("Enter marks: "))
17. student = Student(name, roll_no,
marks)

student.display_details()

```

## Task Description #5

- Ask GitHub Copilot to generate a Python function that takes a string as input and returns the frequency of each word.

The screenshot shows the VS Code interface with the Fibonacci.py file open in the editor. The code defines a word\_frequency function that splits a string into words, converts them to lowercase, and counts their occurrences using a dictionary. A comment in the code specifies an example usage. In the Chat window, GitHub Copilot has provided a detailed explanation of the code and its functionality, along with a copy-and-paste version of the code for the user to use.

```

1 def word_frequency(s):
2     """
3         Returns a dictionary with the frequency of each word in the input string.
4         """
5     words = s.split()
6     freq = {}
7     for word in words:
8         word = word.lower() # Optional: make it case-insensitive
9         freq[word] = freq.get(word, 0) + 1
10    return freq
11
12 # Example usage:
13 text = input("Enter a string: ")
14 print("Word frequencies:", word_frequency(text))

```

## Expected Output #5

- Python function that returns word

frequency using a dictionary. Provide sample input and output.

**Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**

**Evaluation Criteria:**