

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING		
<b>Program Name:</b> M. Tech/MCA		<b>Assignment Type:</b> Lab		
<b>Course Coordinator Name</b>		Venkataramana Veeramsetty		
<b>Course Code</b>		<b>Course Title</b>	AI Assisted Problem Solving Using Python	
<b>Year/Sem</b>	I/I	<b>Regulation</b>	R24	
<b>Date and Day of Assignment</b>	Week1 - Monday	<b>Time(s)</b>		
<b>Duration</b>	2 Hours	<b>Applicable to Batches</b>	M. Tech/MCA	
<b>AssignmentNumber:</b> 1.3(Present assignment number)/24(Total number of assignments)				
<b>Q.No.</b>	<b>Question</b>		<i>Expected Time to complete</i>	
1	<p>Lab 1: Environment Setup – GitHub Copilot and VS Code Integration</p> <p><b>Lab Objectives:</b></p> <ul style="list-style-type: none"> <li>To install and configure GitHub Copilot in Visual Studio Code.</li> <li>To explore AI-assisted code generation using GitHub Copilot.</li> <li>To analyze the accuracy and effectiveness of Copilot's code suggestions.</li> <li>To understand prompt-based programming using comments and code context</li> </ul> <p><b>Lab Outcomes (LOs):</b> After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>Set up GitHub Copilot in VS Code successfully.</li> <li>Use inline comments and context to generate code with Copilot.</li> <li>Evaluate AI-generated code for correctness and readability.</li> <li>Compare code suggestions based on different prompts and programming styles.</li> </ul> <p><b>Task Description#1</b></p> <ul style="list-style-type: none"> <li>Install and configure GitHub Copilot in VS Code. Take screenshots of each step.</li> </ul> <p><b>Expected Output#1</b></p> <ul style="list-style-type: none"> <li>Install and configure GitHub Copilot in VS Code. Take screenshots of each step.</li> </ul>		Week1 - Wednesday	

**Task Description#2**

- Use Copilot to generate a `is_prime()` Python function.

**Expected Output#2**

- Function to check primality with correct logic.

**Task Description#3**

- Write a comment like `# Function to reverse a string and use Copilot to generate the function.`

**Expected Output#3**

- Auto-completed reverse function

**Task Description#4**

- Generate both recursive and iterative versions of a factorial function using comments.

**Expected Output#4**

- Two working factorial implementations

**Task Description#5**

- Use Copilot to find the largest number in a list. Assess code quality and efficiency.

**Expected Output#5**

- A valid function with your review

**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:**

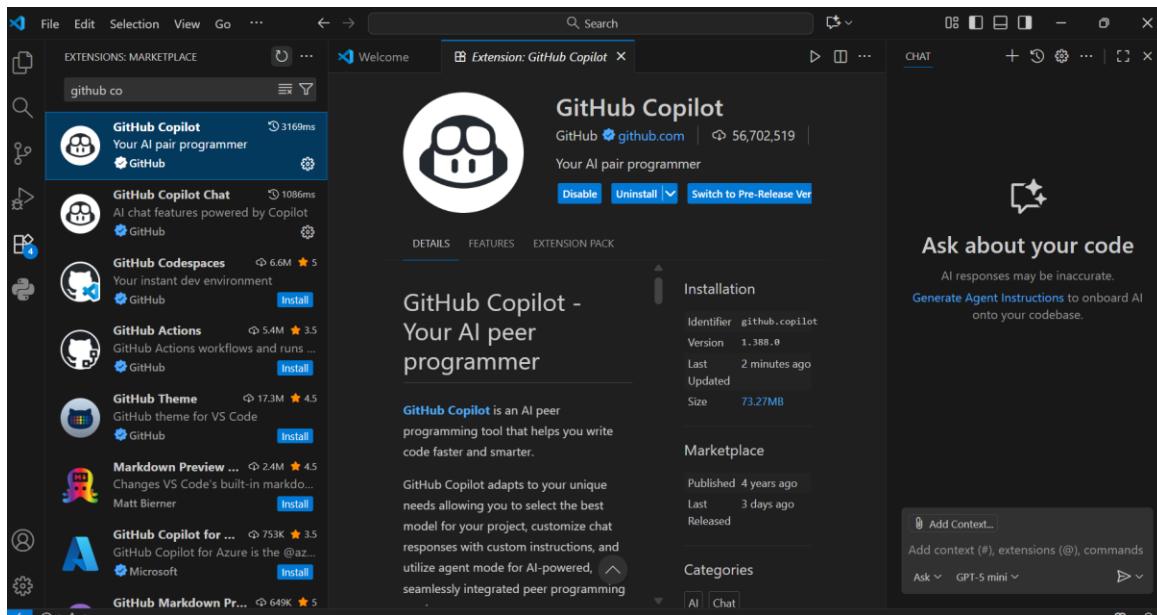
Criteria	Max Marks
Successful Setup of Copilot (Task #1)	2
<code>is_prime()</code> Python function (Task #2)	2
Reverse a string function (Task #3)	2
Factorial Function (Task #4)	2
Find the largest number (Task #5)	2
<b>Total</b>	<b>10 Marks</b>

## Task Description#1

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

### Steps:

- Open VS Code.
- To open Extensions, press “**Ctrl + Shift + X**”.
- In the search bar, type “**GitHub Copilot**”.
- Click **Install** on the official extension by GitHub.
- When prompted, sign in with your GitHub account.



## Task Description#2

Use Copilot to generate a `is_prime()` Python function.

**Prompt:** Provide me a Function to check if a number is prime

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows an unsaved file named "function to check is prime Untitled-1".
- Code Editor:** Displays the generated Python code for `is_prime`. The code uses a quick check for small factors and then tests  $6k \pm 1$  candidates.
- Chat Panel:** Shows a conversation with GPT-5 mini. The user asks for an updated prime check, and the AI provides a faster version using a  $6k \pm 1$  loop instead of a full range loop.
- Terminal:** Shows the following Python code:

```
try:
    num = int(input("Enter an integer: ").strip())
except ValueError:
    print("Invalid input – please enter an integer.")
else:
    if is_prime(num):
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is not a prime number.)
```

## Expected Output#2

Function to check primality with correct logic.

**Practical output:**

The terminal window shows the following output:

```
3 is a prime number.
PS C:\Users\rimsha> []
```

## Task Description#3

Write a comment like # Function to reverse a string and use Copilot to generate the function.

**Prompt:** Give me a function to reverse a string

```
# function to check is prime Untitled-1 ● # Function to reverse a string Untitled-2 ● ▷ ▴ □ ...  
1  # Function to reverse a string  
2  # ...existing code...  
3  
4  def reverse_string(s: str) -> str:  
5      """Return a new string that is the reverse of s.""""  
6      return s[::-1]  
7  
8  # ...existing code...  
9  if __name__ == "__main__":  
10     print(reverse_string(input("Enter a string: ")))
```

## Expected Output#3

## Auto-completed reverse function

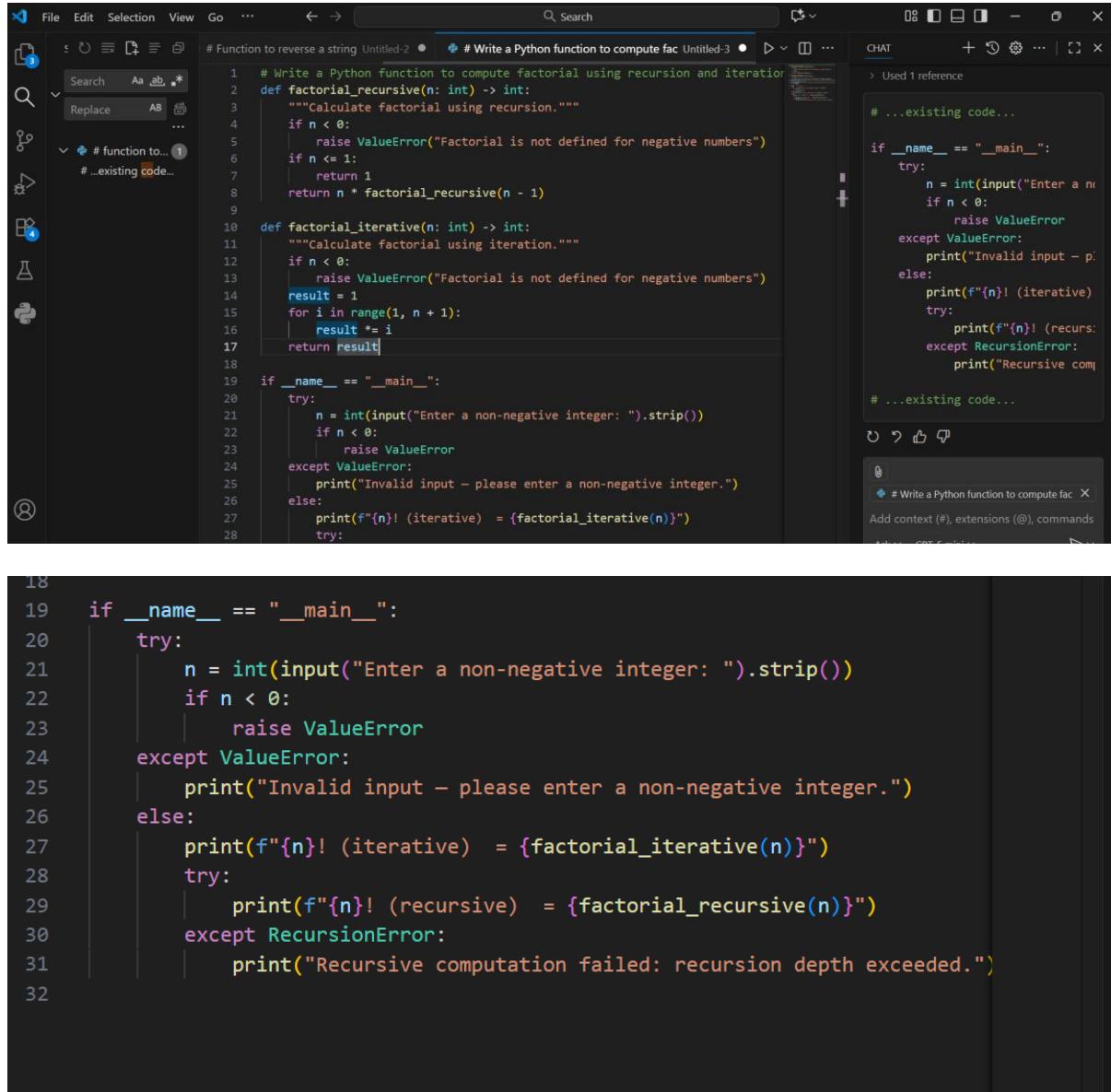
## Practical output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS [Code] + ⌂ ⌄ ⌅ ⌆ ⌇ | [ ] ×  
PS C:\Users\rimsha> python -u "C:\Users\rimsha\AppData\Local\Temp\tempCodeRunnerFile.python"  
Enter a string: RIMSHA  
AHSMIR  
PS C:\Users\rimsha> [ ]
```

## Task Description#4

Generate both recursive and iterative versions of a factorial function using comments.

**Prompt:** Write a Python function to compute factorial using recursion and iteration with user input.



```
# Function to reverse a string Untitled-2 ● # Write a Python function to compute fac Untitled-3 ●
File Edit Selection View Go ... Search CHAT + ⚙️ ...
1 # Write a Python function to compute factorial using recursion and iteration
2 def factorial_recursive(n: int) -> int:
3     """Calculate factorial using recursion."""
4     if n < 0:
5         raise ValueError("Factorial is not defined for negative numbers")
6     if n == 1:
7         return 1
8     return n * factorial_recursive(n - 1)
9
10 def factorial_iterative(n: int) -> int:
11     """Calculate factorial using iteration."""
12     if n < 0:
13         raise ValueError("Factorial is not defined for negative numbers")
14     result = 1
15     for i in range(1, n + 1):
16         result *= i
17     return result
18
19 if __name__ == "__main__":
20     try:
21         n = int(input("Enter a non-negative integer: ").strip())
22         if n < 0:
23             raise ValueError
24     except ValueError:
25         print("Invalid input – please enter a non-negative integer.")
26     else:
27         print(f"{n}! (iterative) = {factorial_iterative(n)}")
28         try:
29             print(f"{n}! (recursive) = {factorial_recursive(n)}")
30         except RecursionError:
31             print("Recursive computation failed: recursion depth exceeded.")
32
# ...existing code...
if __name__ == "__main__":
    try:
        n = int(input("Enter a non-negative integer: ").strip())
        if n < 0:
            raise ValueError
    except ValueError:
        print("Invalid input – please enter a non-negative integer.")
    else:
        print(f"{n}! (iterative) = {factorial_iterative(n)}")
        try:
            print(f"{n}! (recursive) = {factorial_recursive(n)}")
        except RecursionError:
            print("Recursive computation failed: recursion depth exceeded.")

# ...existing code...
Add context (#), extensions (@), commands
# Write a Python function to compute fac
# Write a Python function to compute fac
```

## Expected Output#4

Two working factorial implementations

**Practical output:**

```
PS C:\Users\rimsha> python -u "C:\Users\rimsha\AppData\Local\Temp\tempCodeRunnerFile.python"
Enter a non-negative integer: 5
5! (iterative) = 120
5! (iterative) = 120
5! (recursive) = 120
PS C:\Users\rimsha> []
```

## Task Description#5

Use Copilot to find the largest number in a list. Assess code quality and efficiency.

**Prompt:** To finfd a largest number in the list Then assess the code quality and efficiency.

The screenshot shows a dark-themed version of VS Code. In the center, there is a code editor with the following Python script:

```
# Function to reverse a string Untitled-2 • # Write a Python function to compute fac Untitled-3 • Untitled-4 •
# to finfd a largest number in the list Then assess the code quality and efficiency.

from typing import Iterable, List, Union

Number = Union[int, float]

def parse_numbers(s: str) -> List[Number]:
    """Parse a string of numbers separated by commas/whitespace into a list of floats/integers.
    parts = [p.strip() for p in s.replace(",", " ").split()]
    nums: List[Number] = []
    for p in parts:
        if not p:
            continue
        try:
            if "." in p or "e" in p.lower():
                nums.append(float(p))
            else:
                nums.append(int(p))
        except ValueError:
            raise ValueError(f"Invalid number: {p}")
    return nums

def max_in_list(nums: Iterable[Number]) -> Number:
    """Return the largest element from nums. Raises ValueError for empty iterable."""
    it = iter(nums)
    try:
        max_val = next(it)
    except StopIteration:
        raise ValueError("max_in_list() arg is an empty iterable")
    for x in it:
        if x > max_val:
            max_val = x
    return max_val

if __name__ == "__main__":
    try:
        raw = input("Enter numbers separated by spaces or commas: ").strip()
        numbers = parse_numbers(raw)
        largest = max_in_list(numbers)
    except ValueError as e:
        print(f"Error: {e}")
    else:
        print(f"Largest number: {largest}")
    
```

The right side of the interface shows a sidebar with the task description and some status messages.

This screenshot shows a similar setup in VS Code, but the code in the editor has been modified. The `max_in_list` function now includes a user input loop at the bottom:

```
def max_in_list(nums: Iterable[Number]) -> Number:
    """Return the largest element from nums. Raises ValueError for empty iterable."""
    it = iter(nums)
    try:
        max_val = next(it)
    except StopIteration:
        raise ValueError("max_in_list() arg is an empty iterable")
    for x in it:
        if x > max_val:
            max_val = x
    return max_val

if __name__ == "__main__":
    try:
        raw = input("Enter numbers separated by spaces or commas: ").strip()
        numbers = parse_numbers(raw)
        largest = max_in_list(numbers)
    except ValueError as e:
        print(f"Error: {e}")
    else:
        print(f"Largest number: {largest}")
    
```

## Expected Output#5

A valid function with your review

**Practical output:**

The terminal window shows the command `python -u "C:/Users/rimsha/AppData/Local/Temp/tempCodeRunnerFile.python"` being run. The user then enters a series of numbers separated by spaces or commas, and the script prints out the largest number found.

```
PS C:\Users\rimsha> python -u "C:/Users/rimsha/AppData/Local/Temp/tempCodeRunnerFile.python"
Enter numbers separated by spaces or commas: 1 6 3 8 9 3
Largest number: 9
Largest number: 9
PS C:\Users\rimsha>
```