ASSIGNMENT-1.1

Name: D. Sravika Reddy Hall-Ticket No: 2403a510d0

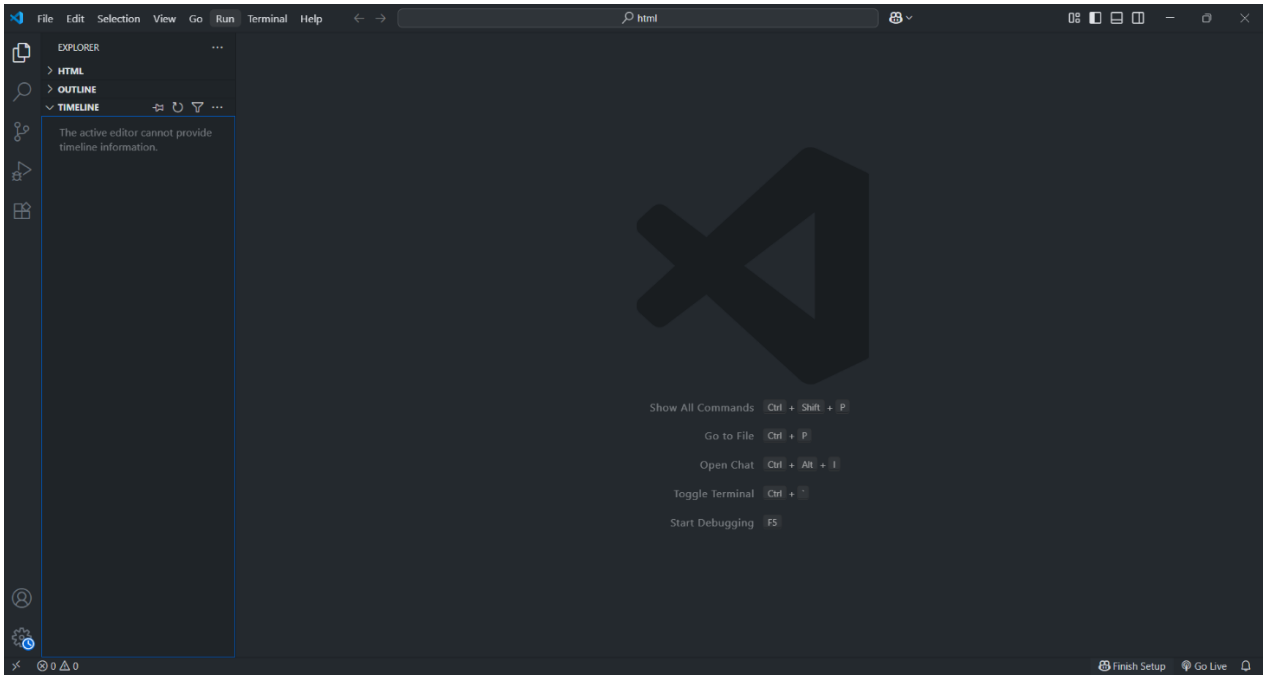
Batch No: 05 Course: AI Assisted Coding

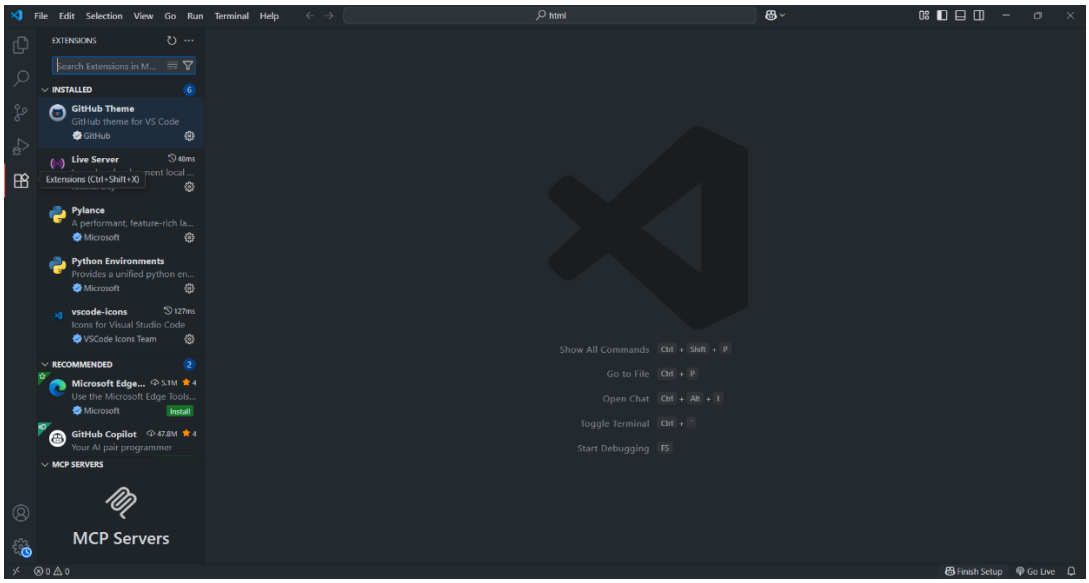
Task 0

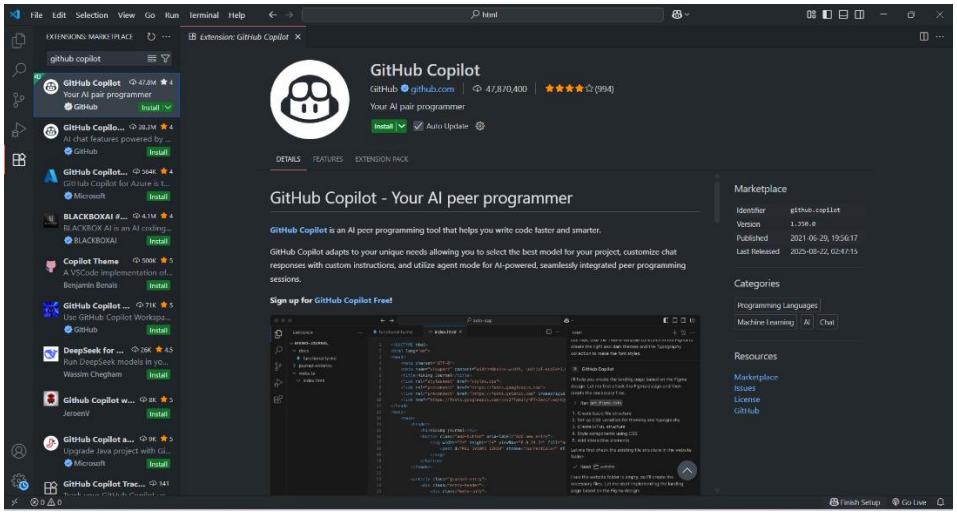
● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

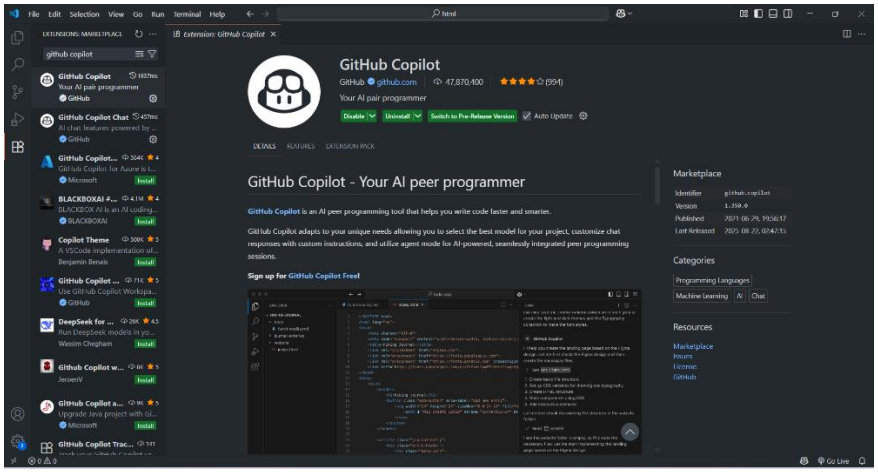
Expected Output

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









Task 1: Factorial without Functions

● Description:

Use GitHub Copilot to generate a Python program that calculates the factorial of a number without defining any functions (using loops directly in the main code).

● Expected Output:

o A working program that correctly calculates the factorial for user-provided input.

o Screenshots of the code generation process

Here is a simple Python program that calculates the factorial of a number using a loop, without defining any functions:

```
# Calculate factorial without using functions
num = int(input("Enter a number: "))
factorial = 1

for i in range(1, num + 1):
    factorial *= i

print("Factorial of", num, "is", factorial)
```

Output:

```
Enter a number: 5
Factorial of 5 is 120
```

Explanation:

Explanation

- The program asks the user to enter a number.
- It initializes `factorial` to 1.
- It uses a `for` loop from 1 to the entered number (inclusive), multiplying `factorial` by each value of `i`.
- After the loop, `factorial` contains the result.
- It prints the factorial of the entered number.

For example, if you enter `5`, the calculation is:
`1 × 2 × 3 × 4 × 5 = 120`

Task 2: Improving Efficiency

● Description:

Examine the Copilot-generated code from Task 1 and demonstrate
how its efficiency can be improved (e.g., removing unnecessary variables, optimizing loops).

● Expected Output:

o Original and improved versions of the code.

o Explanation of how the improvements enhance performance

Original Version

```
num = int(input("Enter a number: "))
factorial = 1

for i in range(1, num + 1):
    factorial *= i

print("Factorial of", num, "is", factorial)
```

Improved Version

```
num = int(input("Enter a number: "))
result = 1

for i in range(2, num + 1):
    result *= i

print("Factorial of", num, "is", result)
```

Explanation:

Explanation of Improvements

- The loop now starts from 2 instead of 1, since multiplying by 1 is unnecessary.
- The variable name `result` is more descriptive.
- This reduces one unnecessary multiplication, making the loop slightly more efficient, especially for large numbers.
- The code is cleaner and easier to read, with no redundant operations.

Task: Factorial with Functions

● Description:

Use GitHub Copilot to generate a Python program that calculates the factorial of a number using a user-defined function.

● Expected Output:

o Correctly working factorial function with sample outputs.

o Documentation of the steps Copilot followed to generate the function.

Here is a Python program that calculates the factorial of a number using a user-defined function:

```python
def factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

num = int(input("Enter a number: "))
print("Factorial of", num, "is", factorial(num))
```

Sample Output

```
Enter a number: 6
Factorial of 6 is 720
```

Explanation:

Documentation of Steps

1. Defined a function `factorial(n)` that takes an integer `n` as input.
2. Initialized `result` to 1.
3. Used a loop from 2 to `n` (inclusive), multiplying `result` by each value.
4. Returned the final result.
5. Prompted the user for input and called the function.
6. Printed the output.

Task4: Comparative Analysis – With vs Without Functions

● Description:

Differentiate between the Copilot-generated factorial program with functions and without functions in terms of logic, reusability, and execution.

● Expected Output:

o A comparison table or short report explaining the differences.

Comparative Analysis: Factorial With vs Without Functions

Aspect	Without Functions	With Functions
Logic	All code is in the main block; uses a loop directly for calculation.	Encapsulates logic in a function; main block calls the function.
Reusability	Not reusable; must rewrite or copy code to use elsewhere.	Highly reusable; can call `factorial(n)` multiple times with different inputs.
Execution	Executes once for a single input; not modular.	Can execute for multiple inputs; modular and easier to test.
Readability	Simple for small scripts, but harder to maintain for larger programs.	More readable and maintainable, especially in larger projects.
Extensibility	Difficult to extend or modify logic.	Easy to extend (e.g., add error handling, recursion, etc.).

Task5: Iterative vs Recursive Factorial

● Description:

Prompt GitHub Copilot to generate both iterative and recursive versions of the factorial function.

● Expected Output:

o Two correct implementations.

o A documented comparison of logic, performance, and execution flow between iterative and recursive approaches

Iterative Factorial Function

```
def factorial_iterative(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

Recursive Factorial Function

```
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)
```

Comparison Table:

Comparison Table

Aspect	Iterative Approach	Recursive Approach
Logic	Uses a loop to multiply numbers up to `n`.	Calls itself with `n-1` until base case is reached.
Performance	More memory efficient; no function call overhead.	Uses more memory due to call stack; can cause stack overflow for large `n`.
Execution Flow	Linear, straightforward, single pass.	Multiple function calls; each call waits for the next.
Readability	Simple and easy to follow for most users.	Elegant and concise, but may be harder for beginners.
Extensibility	Easy to add error handling or modify.	Can be extended for mathematical proofs or tree structures.