# AI ASSISTED CODING

# Environment Setup-GitHub Copilot & VS Code

Name: N.Prudhvi H.no: 2403A510G7

Batch-06

## Task-1

```
Calculate the
      factorial of a number without defining any functions (using loops
      directly in the main code)
  Ask or edit in context
 Accept Close ひ ∨
 # Calculate factorial of a number using a loop (no functions)
 n = int(input("Enter a number: "))
 fact = 1
 for i in range(1, n + 1):
     fact *= i
 print("Factorial of", n, "is", fact)
C:\Users\venub\OneDrive\Desktop\AIAC_Lab\Lab_1>python 1.1.py
Enter a number: 1
Factorial of 1 is 1
C:\Users\venub\OneDrive\Desktop\AIAC Lab\Lab 1>python 1.1.py
Enter a number: 9
Factorial of 9 is 362880
```

#### Task-2

```
# Calculate factorial of a number using a loop (no functions)
# Task1
n = int(input("Enter a number: "))
fact = 1

for i in range(1, n + 1):
    fact *= i

print("Factorial of", n, "is", fact)

# Calculate factorial of a number efficiently using math.factorial
# Improved code(Task2)
import math

n = int(input("Enter a number: "))
fact = math.factorial(n)

print("Factorial of", n, "is", fact)
```

The loop-based factorial calculation is already quite efficient for small numbers. However, you can improve efficiency slightly by using Python's built-in math.factorial() function, which is implemented in C and optimized for performance. If you must use a loop and no functions, you can minimize overhead by using local variables and avoiding unnecessary operations.

## Task-3

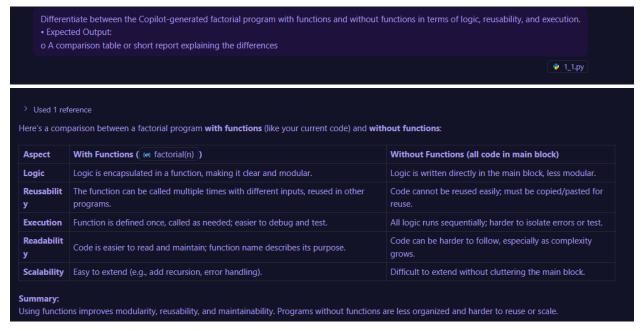
```
# Calculate factorial of a number using a user-defined function

def factorial(n):
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact

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

C:\Users\venub\OneDrive\Desktop\AIAC\_Lab\Lab\_1>python 1\_1.py
Enter a number: 4
Factorial of 4 is 24

## Task-4



## Descriptive form of Task-4:

Using functions encapsulates the factorial logic, making the code modular and easier to understand. Without functions, the logic is embedded in the main block, which can make the code harder to follow and maintain. Reusability: Functions allow the factorial logic to be reused in other parts of the program or in different programs. Without functions, reusing the logic requires copying and pasting code. Execution: Functions can be tested and debugged independently, improving reliability. Without functions, errors are harder to isolate.

## Task-5

```
For the above factorial codes i want the both iterative and recursive versions of the factorial function.
                                                                         1_1.py
# Calculate factorial of a number using user-defined functions
def factorial iterative(n):
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)
n = int(input("Enter a number: "))
print("Iterative: Factorial of", n, "is", factorial_iterative(n))
print("Recursive: Factorial of", n, "is", factorial recursive(n))
 C:\Users\venub\OneDrive\Desktop\AIAC Lab\Lab 1>python 1 1.py
 Enter a number: 8
 Iterative: Factorial of 8 is 40320
 Recursive: Factorial of 8 is 40320
 C:\Users\venub\OneDrive\Desktop\AIAC_Lab\Lab_1>python 1_1.py
 Enter a number: 0
 Iterative: Factorial of 0 is 1
 Recursive: Factorial of 0 is 1
```

## Descriptive form of Task-5:

The iterative approach uses a loop to multiply numbers from 1 up to the input value. This method is straightforward, efficient,

and avoids the overhead of repeated function calls. It is well-suited for large input values, as it does not risk exceeding Python's recursion depth limit.

The recursive approach defines the factorial in terms of itself, calling the function repeatedly with decremented values until reaching the base case. This method is elegant and closely matches the mathematical definition of factorial. However, it can be less efficient for large numbers due to the overhead of multiple function calls and the risk of stack overflow if the recursion depth is too great.