Lab 1.2: Environment Setup – GitHub Copilot and VS Code Integration + AI-Assisted Coding Workflow

Name:B.Tejaskumar

HallTicketNo:2303A51489

Week:1(Monday)

Course:PythonProgrammin

g

Task 0: GitHub Copilot Installation

Steps:
1.InstallVisualStudioCode.
2.OpenExtensionspanel.
3.Searchfor'GitHubCopilot'.
4.ClickInstall.
5.SigninwithGitHubaccount.
6.EnableCopilotinsettings.

(Attachscreenshotsmanuallyinfinalsubmission.)

Task 1: Fibonacci Without Functions (Procedural)

CopilotPromptUsed:GenerateFibonaccisequencewithoutusingfunctions

```
n=int(input('Enternumberofterms:'))
a,b=0,1
print('Fibonaccisequence:'
) foriinrange(n):
    print(a,end='')
    a,b=b,a+b
```

SampleInput:5

Output:01123

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

```
n = int(input("Enter a number: "))

fact = 1
i = 1

while i <= n:
    fact = fact * i
    i += 1

print("Factorial is:", fact)
```

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

Task 2: Optimized Fibonacci Code

Improvements:

- Removed redundant variables
- Simplified loop logic
- Used tuple unpacking for cleaner swapping
- Improved readability

```
n = int(input('Enter number of terms: '))
prev, curr = 0, 1
for _ in range(n):
   print(prev, end=' ')
   prev, curr = curr, prev + curr
```

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

Original AI Code

```
n = int(input("Enter a number: "))

fact = 1
i = 1

while i <= n:
    fact = fact * i
    i += 1

print("Factorial is:", fact)
```

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

Task 3: Fibonacci Using Functions (Modular Approach)

Copilot Prompt Used: Create function-based Fibonacci program

```
def fibonacci(n):
  a, b = 0, 1
  sequence = []
  for _ in range(n):
    sequence.append(a)
    a, b = b, a + b
  return sequence

n = int(input('Enter number of terms: '))
print(fibonacci(n))
```

Sample Input: 5

Output: [0, 1, 1, 2, 3]

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

```
]: def calculate_factorial(number):
    # This function computes factorial using iteration
    result = 1

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

    return result


# Main execution block
n = int(input("Enter a number: "))

answer = calculate_factorial(n)

print("Factorial is:", answer)
```

```
Enter a number: 2
Factorial is: 2
```

Task 4: Comparative Analysis

Procedural vs Modular:

- Code Clarity: Modular is cleaner.
- Reusability: Function-based code reusable.
- Debugging: Easier in modular approach.
- Suitable for large systems: Modular approach preferred.

Task 5: Iterative vs Recursive Fibonacci

Iterative Implementation:

```
def fib_iterative(n):
  a, b = 0, 1
  for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

Recursive Implementation:

```
def fib_recursive(n):
  if n <= 1:
    return n
  return fib_recursive(n-1) + fib_recursive(n-2)
```

```
for i in range(5):
    print(fib_recursive(i), end=' ')
```

Task 5: AI-Generated Iterative vs Recursive Thinking

Iterative Version

```
n [ ]: def factorial_iterative(n):
           result = 1

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

           return result
```

Recursive Version

```
n [ ]: def factorial_recursive(n):
           # Base case
           if n == 0 or n == 1:
               return 1

           return n * factorial_recursive(n - 1)
```

Comparison:
• Iterative: O(n) time, O(1) space.
• Recursive: O(2^n) time, higher memory usage.
• Recursion should be avoided for large n.

Conclusion

This lab demonstrated GitHub Copilot integration, AI-assisted programming, code optimization, modular design, and algorithm comparison.
AI improves productivity but requires human validation.