AI Assisted Coding

Assignment-01

NAME : P.CHANDRA VARDHAN REDDY

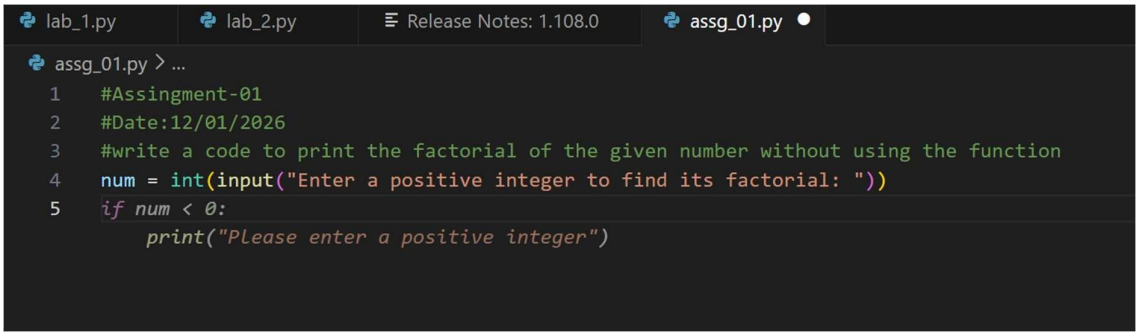
HALL TICKET NO: 2303A51034

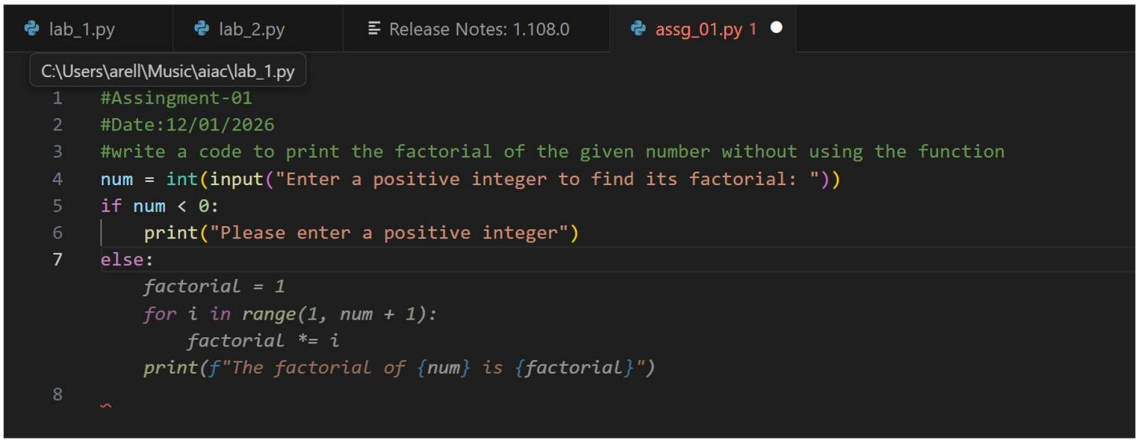
BATCH-01

- A working Python program generated with Copilot assistance

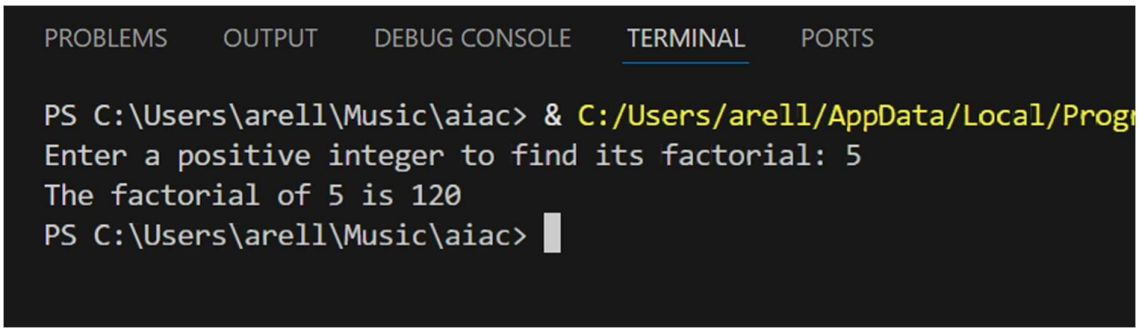
```
❖ #Assingment-01
❖ #Date:12/01/2026
❖ #write a code to print the factorial of the given number without using the function ❖ num =
int(input("Enter a positive integer to find its factorial: ")) ❖ if num < 0:
❖ print("Please enter a positive integer") ❖ else:
❖ factorial = 1
❖ for i in range(1, num + 1):
❖ factorial *= i
❖ print(f"The factorial of {num} is {factorial}")
```

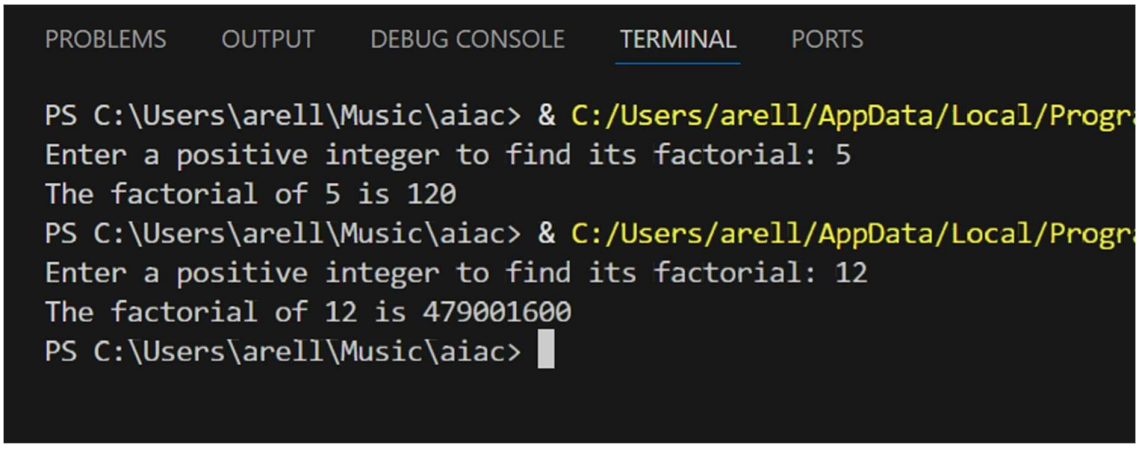
- Screenshot(s) showing:





- Sample input/output screenshots





- Brief reflection (5–6 lines):

  GitHub Copilot was used to generate the logic quickly without putting much thought into the program structure. It advocated a fresh and clean approach to code and made it readable without having functions in use. Following its suggestions, I reduced the need for me to memorize exact syntax, freeing my mind to focus on the core logic. Copilot also helped me write simple and efficient code in a neat way. Overall, it was faster to develop, less stressful and better in terms of clarity.

- How helpful was Copilot for a beginner?

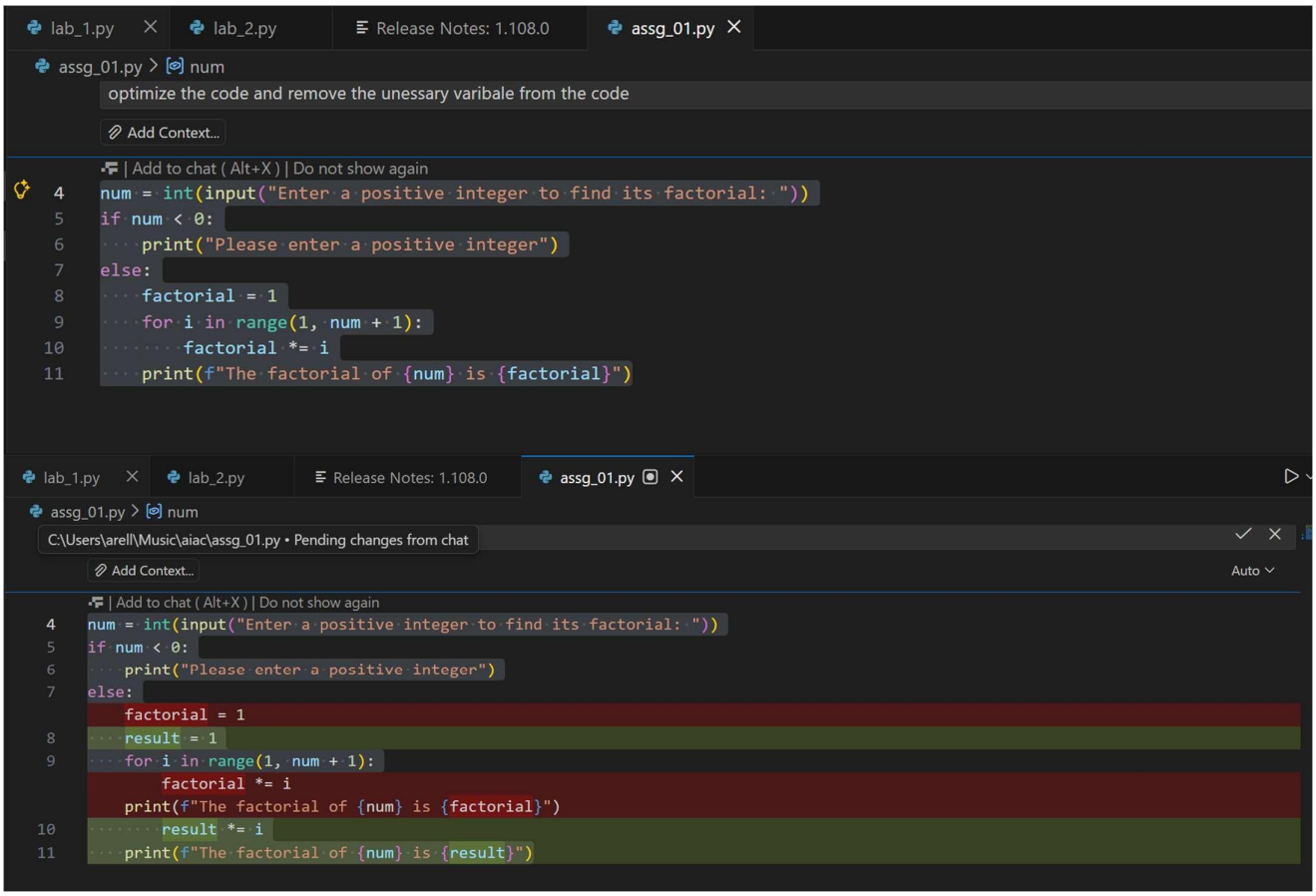
  ->copilot is very useful for beginners because it provides instant code suggestion when the user is stuck in middle. It will help beginner by suggesting the code so that user can understand the code and memorized and it makes learning programming easier and more confident

- Did it follow best practices automatically?

  ->Yes it follows best practices automatically for user by writing the code itself copilot will guess the next lines by that user can also more confident and user also get to know what next should be .It automatically best practices

Task-02

- Screenshot(s) showing:



code

```
#Assingment-01
#Date:12/01/2026
#write a code to print the factorial of the given number without using the function num =
int(input("Enter a positive integer to find its factorial: ")) if num < 0:
    print("Please enter a positive integer") else:
    result = 1      for i in range(1, num + 1):
        result *= i     print(f"The factorial of {num} is {result}")
```

side by side comparition

```
n=int(input("Enter a number: ")) result = 1
if n < 0:
print("Factorial is not defined for negative numbers")


else: for i in range(1, n + 1):
result = result * i print("Factorial of", n, "is",
result)
```

optimal code

```
n = int(input("Enter a number: ")) if n < 0:
print("Factorial is not defined for negative numbers") else:
result = 1 for i in range(2, n + 1):
result *= i print(f"Factorial of {n} is {result}")
```

task-03

Modular Design Using AI Assistance (Factorial with Functions)

lab_1.py lab_2.py assg_01.py ●

assg_01.py > ...

```
12
13  #write a program to print factorial of the given number by using the fuction
14  def factorial(n):
        if n < 0:
            return "Please enter a positive integer"
        elif n == 0 or n == 1:
            return 1
        else:
            result = 1
            for i in range(2, n + 1):
                result *= i
            return result
```

```python
#write a program to print factorial of the given number by using the fuction
def factorial(n):
    if n < 0:
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # check for 0! and 1!
        return 1
    else:
        result = 1
        for i in range(2, n + 1): # loop from 2 to n
            result *= i
        return result
```

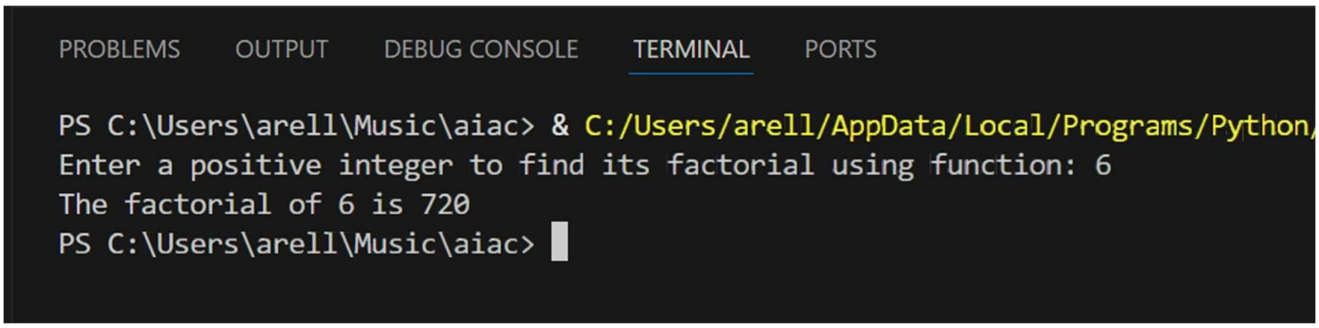
lab_1.py | lab_2.py | assg_01.py

assg_01.py > factorial

```python
#write a program to print factorial of the given number by using the fuction
def factorial(n):
    if n < 0:# check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # check for 0! and 1!
        return 1
    else:
        result = 1
        for i in range(2, n + 1): # loop from 2 to n
            result *= i # multiply result by i
        return result # return the factorial value
number = int(input("Enter a positive integer to find its factorial using function: "))
print(f"The factorial of {number} is {factorial(number)}")
```

```
#write a program to print factorial of the given number by using the fuction def factorial(n):
    if n < 0:# check for negative input         return "Please enter a
positive integer"     elif n == 0 or n == 1: # check for 0! and 1!
        return 1     else:
        result = 1         for i in range(2, n + 1): # loop from 2 to n             result *= i # multiply
result by i         return result # return the factorial value number = int(input("Enter a positive
integer to find its factorial using function: ")) print(f"The factorial of {number} is
{factorial(number)}")
```

- Sample input/output screenshots



Short note:

How modularity improves reusability.

Modularity improves reusability by breaking a program into independent, self-contained parts such as functions or modules. Each module can be reused in different programs or in multiple places within the same program without rewriting code. This reduces duplication and saves development time. Modular code is also easier to test, debug, and maintain because changes in one module do not affect the entire program. Overall, modularity makes software more flexible and scalable..

Task 4:

Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions

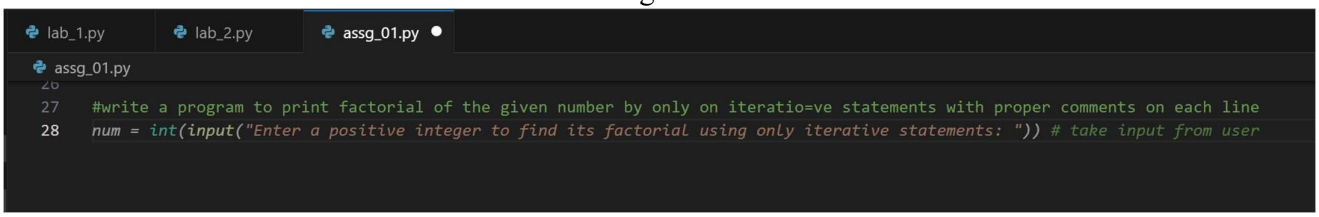
꾀 Comparative Analysis: Procedural vs Modular AI Code

Criteria	Procedural Code (Without Functions)	Modular Code (With Functions)
Logic Clarity	The logic is written in a single block, which is easy to understand for	The logic is organized inside a function, making the program
	small programs but becomes difficult to follow as the code grows.	more structured and easier to understand.

Reusability	The code cannot be reused easily and must be rewritten if the same logic is needed again.	The function can be reused multiple times across the program or in other programs.
Debugging Ease	Debugging is harder because the entire code must be checked for errors.	Debugging is easier since errors are confined within the function.
Suitability for Large Projects	This approach is not suitable for large projects due to poor structure and maintainability.	This approach is suitable for large projects because modular design improves organization and scalability.
AI Dependency Risk	Since the logic is simple, dependency on AI is minimal.	Over-dependence on AIgenerated functions without understanding the logic can introduce risks.

Task 5:

AI-Generated Iterative vs Recursive Thinking




```
#write a program to print factorial of the given number by only on iteratio=ve statements with proper comments on each line
num = int(input("Enter a positive integer to find its factorial using only iterative statements: ")) # take input from user
if num < 0: # check for negative input
    print("Please enter a positive integer")
else:
    factorial = 1 # initialize factorial variable
    i = 2 # start from 2
    while i <= num: # loop until i is less than or equal to num
        factorial *= i # multiply factorial by i
        i += 1 # increment i by 1
    print(f"The factorial of {num} is {factorial}") # print the result
```

```
#write a program to print factorial of the given number by only on iteratio=ve statements with proper comments on each line  num = int(input("Enter a positive integer to find its factorial using only iterative statements: ")) # take input from user if num < 0: # check for negative input
print("Please enter a positive integer")
else: # if input is valid     factorial = 1 # initialize factorial variable     i = 2 # start from 2
iteration_count = 0 # initialize iteration counter     while i <= num: # loop until i is less than or equal to num         factorial *= i # multiply factorial by i         i += 1 # increment i by 1         iteration_count += 1 # increment iteration counter     print(f"The factorial of {num} is {factorial}") # print the result
print(f"Number of iterations: {iteration_count}") # print iteration count
```

```
Enter a positive integer to find its factorial using only iterative statements: 20
The factorial of 20 is 2432902008176640000
Number of iterations: 19
PS C:\Users\arell\Music\aiac>
```

```
Enter a positive integer to find its factorial using only iterative statements: 100
The factorial of 100 is 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
Number of iterations: 99
```

outputs

Using Recursion:

```python
#write a program to print factorail of the given number by using recursion with proper comments on each line
def factorial(n):
    if n < 0: # check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # base case for recursion
        return 1
    else:
        return n * factorial(n - 1) # recursive call
```

```python
#write a program to print factorail of the given number by using recursion with proper comments on each line
def factorial(n):
    if n < 0: # check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # base case for recursion
        return 1
    else:
        return n * factorial(n - 1) # recursive case
number = int(input("Enter a positive integer to find its factorial using recursion: ")) # take input from user
print(f"The factorial of {number} is {factorial(number)}") # print the result

```

```python
#write a program to print factorail of the given number by using recursion with proper comments on each line
def factorial(n, count=[0]): # define factorial function with iteration counter
    count[0] += 1 # increment iteration counter
    if n < 0: # check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # base case for recursion
        return 1
    else:
        return n * factorial(n - 1, count) # recursive case
count = [0] # initialize iteration counter
number = int(input("Enter a positive integer to find its factorial using recursion: ")) # take input from user
print(f"The factorial of {number} is {factorial(number, count)}") # print the result
print(f"Number of iterations: {count[0]}") # print iteration count
```

```
#write a program to print factorail of the given number by using recursion with proper comments on each line def factorial(n, count=[0]): # define factorial function with iteration counter count[0] += 1 # increment iteration counter if n < 0: # check for negative input return "Please enter a positive integer" elif n == 0 or n == 1: # base case for recursion return 1 else:
    return n * factorial(n - 1, count) # recursive case count = [0] # initialize iteration counter number = int(input("Enter a positive integer to find its factorial using recursion: ")) # take input from user print(f"The factorial of {number} is {factorial(number, count)}") # print the result print(f"Number of iterations: {count[0]}") # print iteration count
```

output:

```
Enter a positive integer to find its factorial using recursion: 20
The factorial of 20 is 2432902008176640000
Number of iterations: 20
```

```
Enter a positive integer to find its factorial using recursion: 100
The factorial of 100 is 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272
23758251185210916864000000000000000000000000
Number of iterations: 100
```

Comparison: Iterative vs Recursive Approach

Criteria	Iterative Approach	Recursive Approach
Readability	Easy to understand for beginners, logic is straightforward.	Code is shorter and elegant but may be confusing for beginners.
Stack Usage	Does not use call stack, memory usage is minimal.	Uses call stack for every function call, increases memory usage.
Performance Implications	Faster and more efficient for large inputs.	Slower due to repeated function calls and stack overhead.
When Recursion Is Not Recommended	Not applicable (safe for large inputs).	Not recommended for large inputs due to stack overflow risk and recursion limits.