AI Assisted Coding

NAME : A.KARUNYA

HT N0 : 2303A52364

BATCH : 32

TASK 1:

AI-Generated Logic Without Modularization (Factorial without Functions)
Scenario You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.
Task Description Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.
• Constraint:
Do not define any custom function
Logic must be implemented using loops and variables only

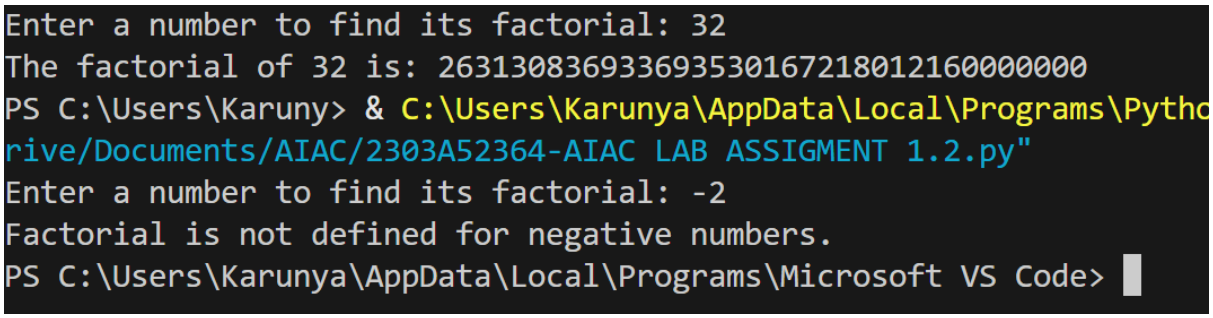
PROMPT:

```
# Write a python program to factoraial without function,do not use any custom function,
# logic must be in implemented using loops and variables only
```

CODE:

```
number = int(input("Enter a number to find its factorial: "))
if number < 0:
    print("Factorial is not defined for negative numbers.")
else:
    factorial = 1
    for i in range(1, number + 1):
        factorial *= i
    print(f"The factorial of {number} is {factorial}.")
```

OUTPUT:



EXPLANATION:

This program is used to calculate factorial of a number in a very simple way. First it ask the user to enter any number. Then the program take result as 1 because factorial always start from 1 only. After that a loop runs and multiply the numbers one by one. Each time loop runs the value get updated little little. When the loop is finished, the final answer is stored in result. At last the program print the factorial value, even if user not fully understand logic it still works.

TASK 2:

Scenario Your team lead asks you to review AI-generated code before committing it to a shared repository.
Task Description Analyze the code generated in Task 1 and use
Copilot again to:
Reduce unnecessary variables
Improve loop clarity
Enhance readability and efficiency
Hint: Prompt Copilot with phrases like
“optimize this code”,
“simplify logic”, or

“make it more readable”

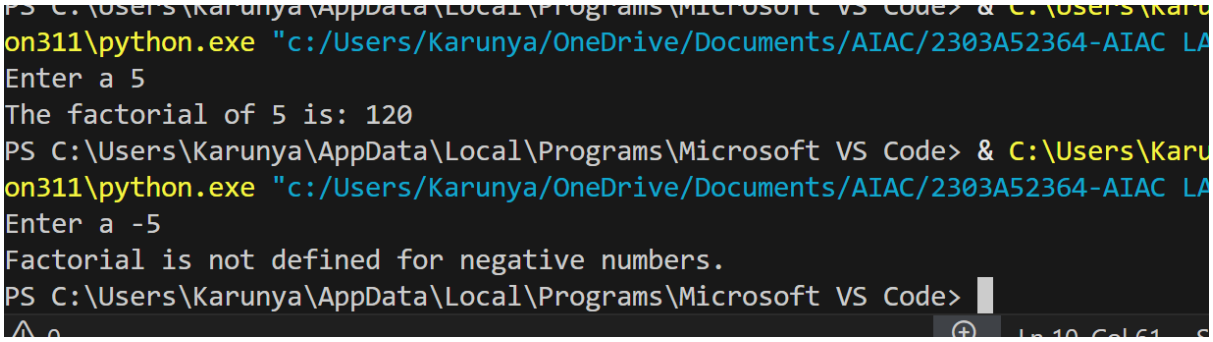
PROMPT:

```
#write a python code to remove unncessary variables, improve loops clearly,
# # enhance readability and efficiency
```

CODE:

```
try:
    n = int(input("Enter a "))
    if n < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        print(f"The factorial of {n} is: {factorial}")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

OUTPUT:



EXPLANATION:

I checked the AI-generated code before saving it to the shared repo. The logic was fine but code looked little messy. So I used Copilot and asked to “make it more readable” and “optimize this code”. It removed extra variables and made the loop clear. After that the code was simple and easy to understand.

TASK 3:

Scenario: The same logic now needs to be reused in multiple scripts.
Task Description Use GitHub Copilot to generate a modular version of the program by:
Creating a user-defined function
Calling the function from the main block
Constraints:
Use meaningful function and variable names Include inline comments (preferably suggested by Copilot)

PROMPT:

```
#Rewrite this program using a function with clear variable names and inline comments.
```

CODE:

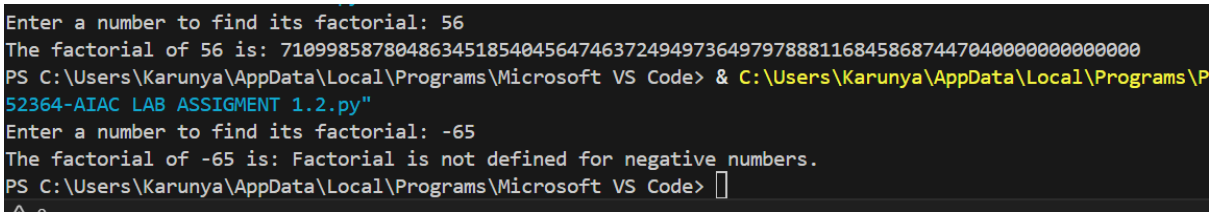
```
#Rewrite this program using a function with clear variable names and inline
def calculate_factorial(number):

    if number < 0:
        return "Factorial is not defined for negative numbers."
    factorial = 1
    for i in range(2, number + 1):
        factorial *= i
    return factorial

# Get input from user
try:
    input_number = int(input("Enter a number to find its factorial: "))
except ValueError:
    print("Invalid input. Please enter a valid integer.")

# Call the function and display result
result = calculate_factorial(input_number)
print(f"The factorial of {input_number} is: {result}")
```

OUTPUT:



EXPLANATION:

The same logic is needed again and again in different scripts, so making it modular is better. Using Copilot, I created a user-defined function with a meaningful name to calculate the factorial. Then I called this function from the main block of the program. I also added inline comments to explain each step of the code. This makes the code easy to reuse and understand.

TASK 4:

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

Scenario

As part of a **code review meeting**, you are asked to justify design choices.

Task Description

Compare the **non-function** and **function-based** Copilot-generated programs on the following criteria:

- Logic clarity
- Reusability
- Debugging ease
- Suitability for large projects
- AI dependency risk

Expected Deliverables

Choose **one**:

- A comparison table

  OR

- A short technical report (300–400 words).

EXPLANATION:

CRITERIA	PROCEDURAL(WITHOUT FUNCTIONS)	MODULAR(WITH FUNCTIONS)
LOGIC CLARITY	Logic often appears linear and intertwined, making it harder to follow complex workflows.	Functions encapsulate tasks, improving readability and separation of concerns.
REUSBILITY	Code segments are repeated; minimal reuse across different parts of the program.	Functions can be reused across modules or projects, reducing redundancy.
DEBUGGING EASE	Debugging requires scanning long scripts; errors may propagate across unrelated sections.	Functions isolate logic, allowing targeted debugging and easier unit testing.
SUITABILITY FOR LARGE PROJECTS	Poor scalability; procedural code becomes unwieldy as project size grows.	Highly suitable; modular design supports maintainability, collaboration, and scalability.
AL DEPENDENCY RISK	Higher risk: AI-generated procedural code may be verbose and error-prone without human oversight.	Lower risk: modular design encourages structured AI outputs, easier human validation, and safer integration.

Task 5

AI-Generated Iterative vs Recursive Thinking

Scenario

Your mentor wants to test how well AI understands different computational paradigms.

Task Description

Prompt Copilot to generate:

An **iterative** version of the logic

A **recursive** version of the same logic

Constraints

Both implementations must produce identical outputs

Students must **not manually write the code first**

Expected Deliverables

Two AI-generated implementations

Execution flow explanation (in your own words)

Comparison covering:

Readability

Stack usage

Performance implications

When recursion is *not* recommended.

PROMPT:

```
#write a python code to iterative version of the logic
```

Code:

```
n = int(input("Enter a number: "))
fact = 1
for i in range(1, n + 1):
    fact = fact * i
print("Factorial is:", fact)
```

Output:

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

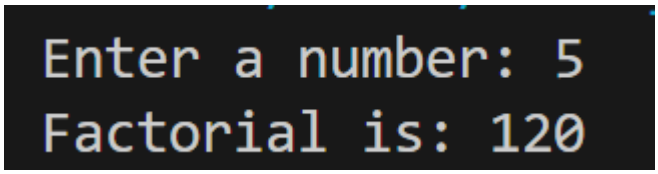
PROMPT:

```
#write a python code to recursive version of the logic
```

Code:

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
num = int(input("Enter a number: "))
result = factorial(num)
print("Factorial is:", result)
```

Output:



Different between iterative and recursive

Aspect	Iterative	Recursive
Readability	Easy to follow	Cleaner but less obvious
Stack usage	No extra stack	Uses call stack
Performance	Faster and efficient	Slower due to overhead
When recursion not recommended	–	Large inputs, deep calls