AIAC – 1.2

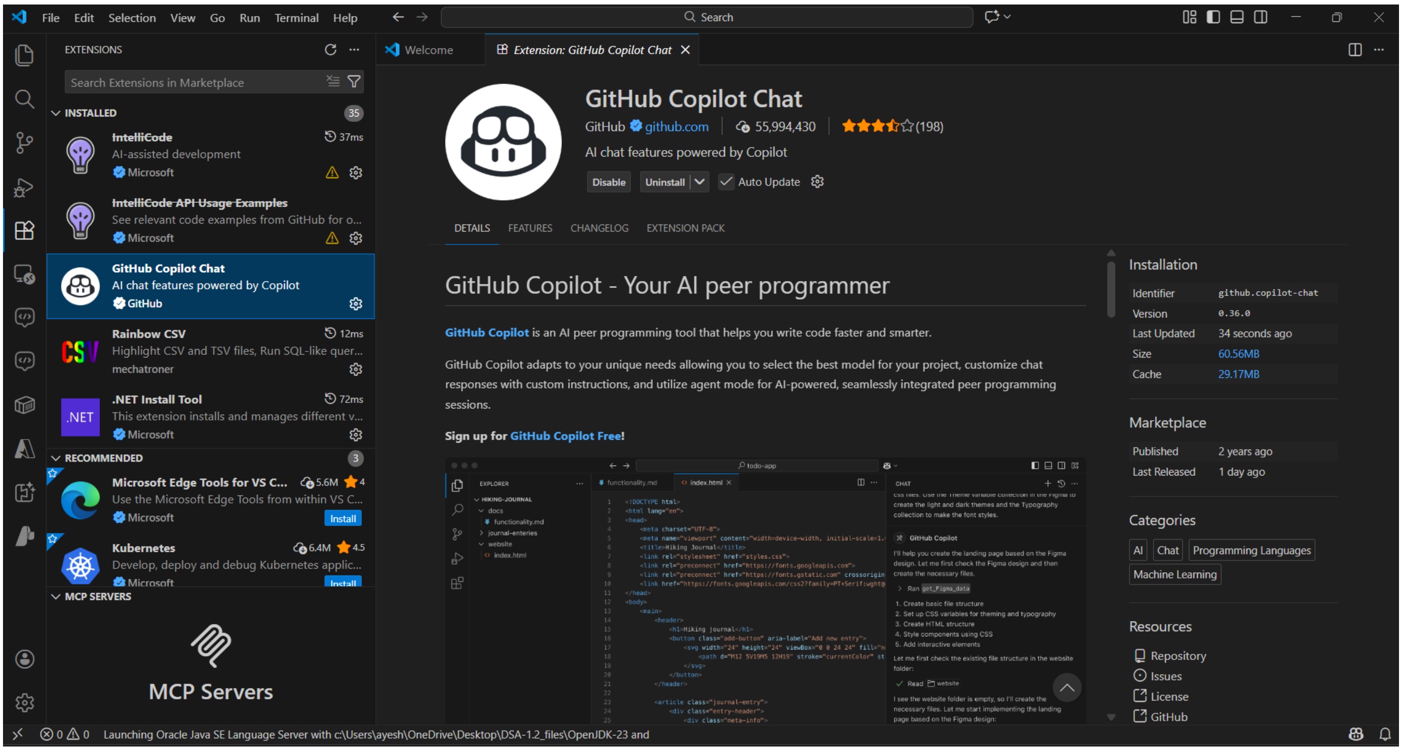
2303A52065

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.


File Edit Selection View Go Run Terminal Help
Search
EXTENSIONS
Search Extensions in Marketplace
INSTALLED 35
Welcome
Extension: GitHub Copilot Chat
Disable
Uninstall
Auto Update
DETAILS FEATURES CHANGELOG EXTENSION PACK
RECOMMENDED 3
MCP SERVERS
MCP Servers
Install

IntelliCode 37ms
AI-assisted development
Microsoft

~~**IntelliCode API Usage Examples**~~
See relevant code examples from GitHub for o...
Microsoft

GitHub Copilot Chat
AI chat features powered by Copilot
GitHub

Rainbow CSV 12ms
Highlight CSV and TSV files, Run SQL-like quer...
mechatroner

.NET Install Tool 72ms
This extension installs and manages different v...
Microsoft

Microsoft Edge Tools for VS C... 5.6M 4
Use the Microsoft Edge Tools from within VS C...
Microsoft

Kubernetes 6.4M 4.5
Develop, deploy and debug Kubernetes applic...
Microsoft

GitHub Copilot Chat

GitHub github.com 55,994,430 (198)

AI chat features powered by Copilot

GitHub Copilot - Your AI peer programmer

GitHub Copilot is an AI peer programming tool that helps you write code faster and smarter.

GitHub Copilot adapts to your unique needs allowing you to select the best model for your project, customize chat responses with custom instructions, and utilize agent mode for AI-powered, seamlessly integrated peer programming sessions.

Sign up for GitHub Copilot Free!

Installation

Identifier	github.copilot-chat
Version	0.36.0
Last Updated	34 seconds ago
Size	60.56MB
Cache	29.17MB

Marketplace

Published	2 years ago
Last Released	1 day ago

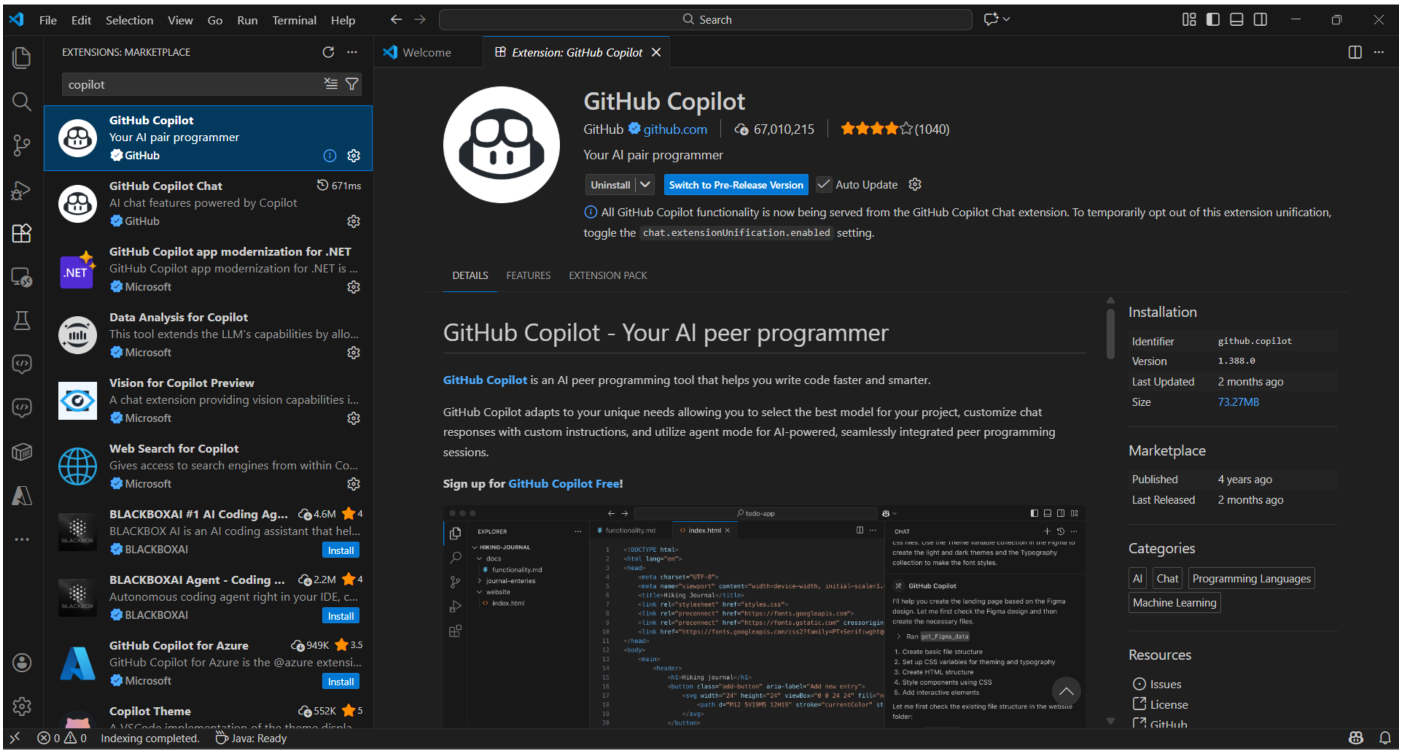
Categories

AI Chat Programming Languages Machine Learning

Resources

Repository
Issues
License
GitHub

0 0 Launching Oracle Java SE Language Server with c:\Users\ayesh\OneDrive\Desktop\DSA-1.2_files\OpenJDK-23 and


File Edit Selection View Go Run Terminal Help
Search
EXTENSIONS: MARKETPLACE
copilot
Welcome
Extension: GitHub Copilot
Uninstall
Switch to Pre-Release Version
Auto Update
DETAILS FEATURES EXTENSION PACK
Install

GitHub Copilot
Your AI pair programmer
GitHub

GitHub Copilot Chat 671ms
AI chat features powered by Copilot
GitHub

GitHub Copilot app modernization for .NET
GitHub Copilot app modernization for .NET is ...
Microsoft

Data Analysis for Copilot
This tool extends the LLM's capabilities by allo...
Microsoft

Vision for Copilot Preview
A chat extension providing vision capabilities i...
Microsoft

Web Search for Copilot
Gives access to search engines from within Co...
Microsoft

BLACKBOXAI #1 AI Coding Ag... 4.6M 4
BLACKBOX AI is an AI coding assistant that hel...
BLACKBOXAI

BLACKBOXAI Agent - Coding ... 2.2M 4
Autonomous coding agent right in your IDE, c...
BLACKBOXAI

GitHub Copilot for Azure 949K 3.5
GitHub Copilot for Azure is the @azure extensi...
Microsoft

Copilot Theme 552K 5

GitHub Copilot

GitHub github.com 67,010,215 (1040)

Your AI pair programmer

All GitHub Copilot functionality is now being served from the GitHub Copilot Chat extension. To temporarily opt out of this extension unification, toggle the `chat.extensionUnification.enabled` setting.

GitHub Copilot - Your AI peer programmer

GitHub Copilot is an AI peer programming tool that helps you write code faster and smarter.

GitHub Copilot adapts to your unique needs allowing you to select the best model for your project, customize chat responses with custom instructions, and utilize agent mode for AI-powered, seamlessly integrated peer programming sessions.

Sign up for GitHub Copilot Free!

Installation

Identifier	github.copilot
Version	1.388.0
Last Updated	2 months ago
Size	73.27MB

Marketplace

Published	4 years ago
Last Released	2 months ago

Categories

AI Chat Programming Languages Machine Learning

Resources

Issues
License
GitHub

0 0 Indexing completed. Java: Ready

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

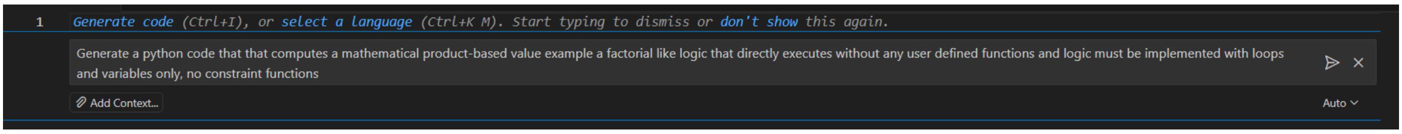
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- Constraint:

➢ Do not define any custom function

➢ Logic must be implemented using loops and variables only

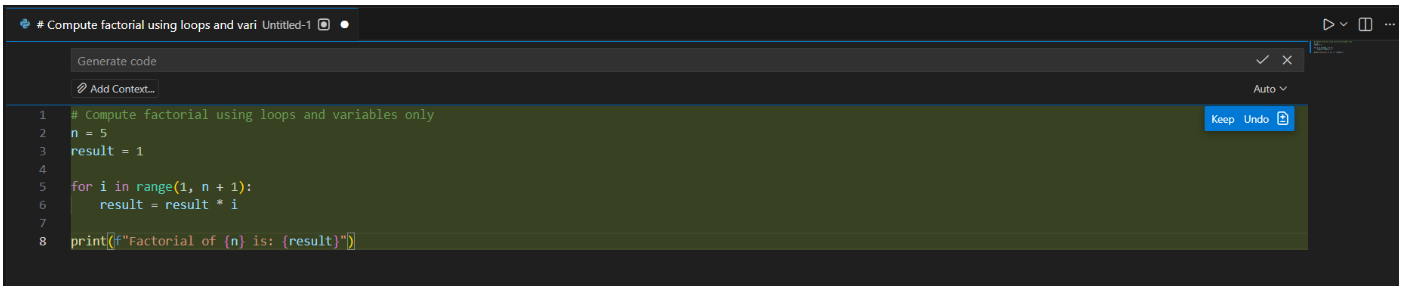
Prompt:



```
Generate code (Ctrl+I), or select a language (Ctrl+K M). Start typing to dismiss or don't show this again.
```

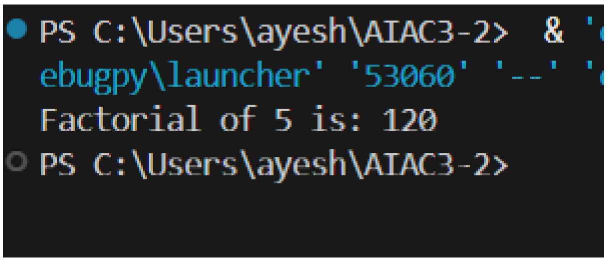
Generate a python code that that computes a mathematical product-based value example a factorial like logic that directly executes without any user defined functions and logic must be implemented with loops and variables only, no constraint functions

Generated Code:



```python
# Compute factorial using loops and variables only
n = 5
result = 1

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

print(f"Factorial of {n} is: {result}")
```

Output:



```
PS C:\Users\ayesh\AIAC3-2>  & '
ebugpy\launcher' '53060' '--' '
Factorial of 5 is: 120
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

GitHub Copilot was helpful in generating the factorial logic quickly using only loops and variables, without requiring any custom functions. It initialized proper variables and made learning easier without worrying about syntax errors and focus on understanding the logic.

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

❖ Scenario

Your team lead asks you to review AI-generated code before committing it to a shared repository.

❖ Task Description

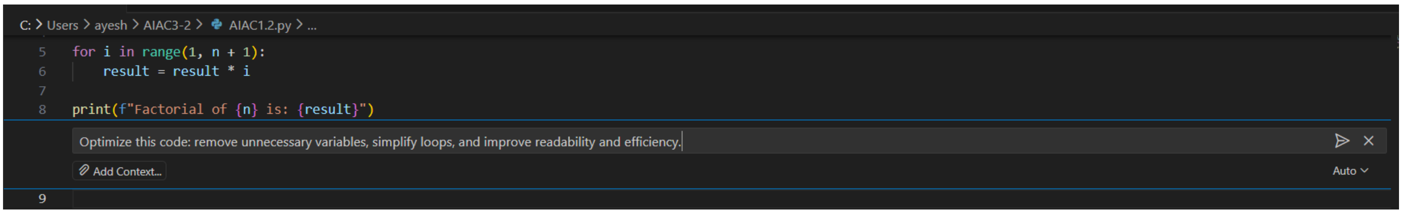
Analyze the code generated in Task 1 and use Copilot again to:

➢ Reduce unnecessary variables

➢ Improve loop clarity

➢ Enhance readability and efficiency

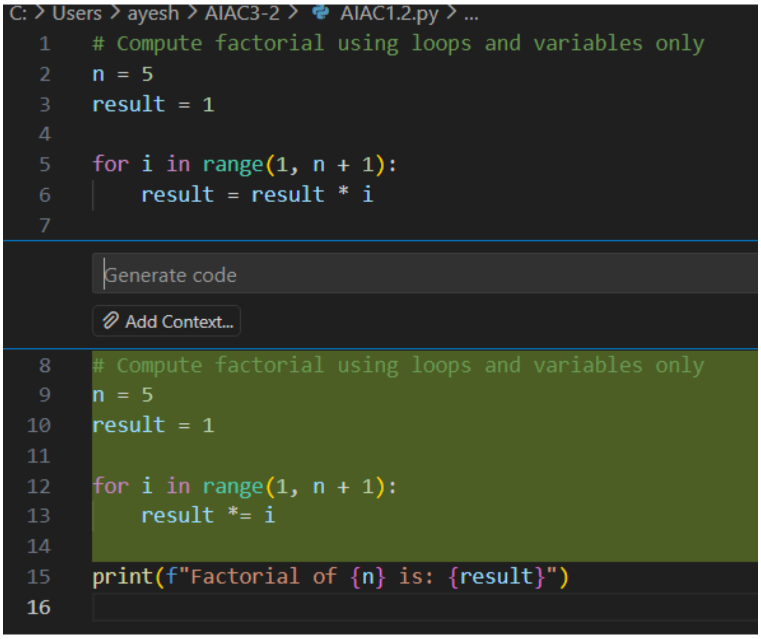
Prompt:



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

print(f"Factorial of {n} is: {result}")
```

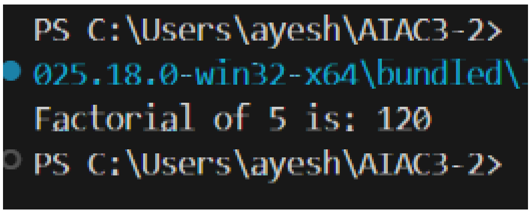
Optimize this code: remove unnecessary variables, simplify loops, and improve readability and efficiency.

Generated Code:



```python
# Compute factorial using loops and variables only
n = 5
result = 1

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

# Compute factorial using loops and variables only
n = 5
result = 1

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

print(f"Factorial of {n} is: {result}")
```

Output:



```
PS C:\Users\ayesh\AIAC3-2>
025.18.0-win32-x64\bundled\
Factorial of 5 is: 120
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

The optimized code improves readability by using a cleaner multiplication operator while keeping the same logic. The optimized version follows better Python coding practices without changing the output. Overall, the code becomes more concise and readable while maintaining the same performance.

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

❖ 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

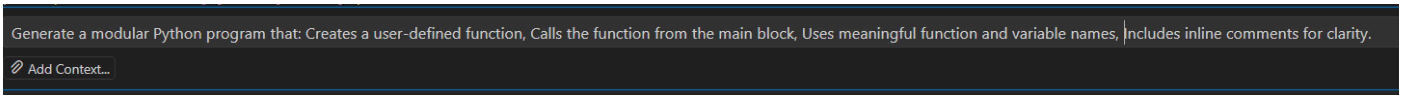
➢ Calling the function from the main block

❖ Constraints

➢ Use meaningful function and variable names

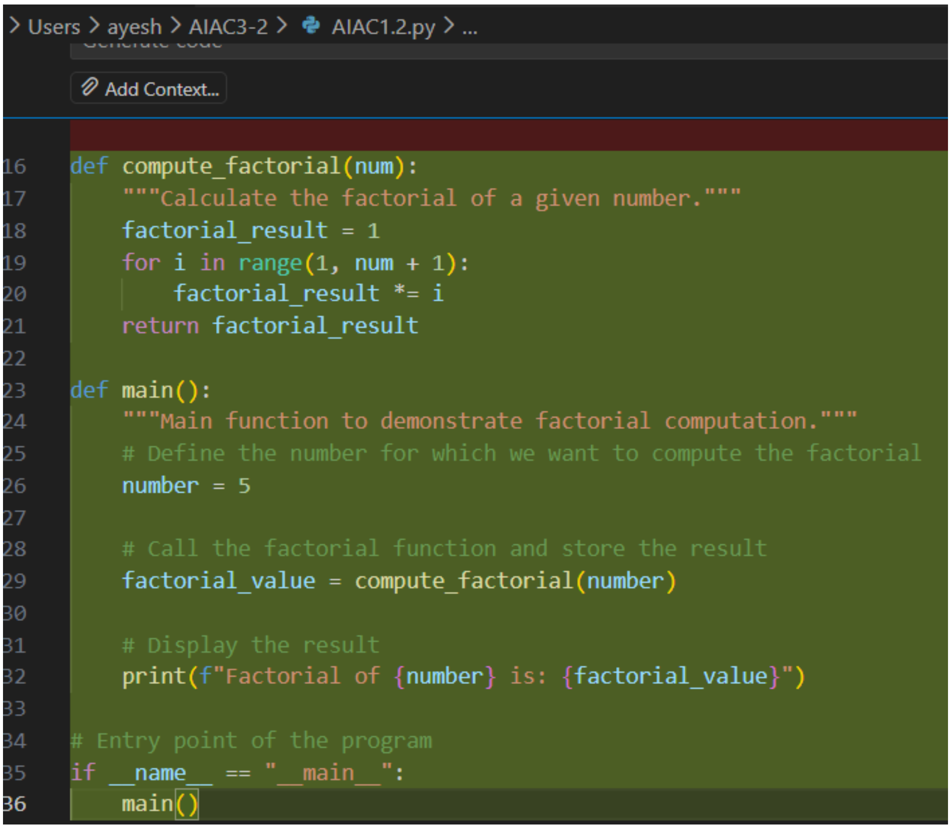
➢ Include inline comments (preferably suggested by Copilot)

Prompt:



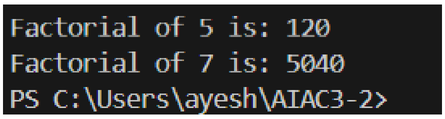
Generate a modular Python program that: Creates a user-defined function, Calls the function from the main block, Uses meaningful function and variable names, Includes inline comments for clarity.

Generated Code:



```python
def compute_factorial(num):
    """Calculate the factorial of a given number."""
    factorial_result = 1
    for i in range(1, num + 1):
        factorial_result *= i
    return factorial_result

def main():
    """Main function to demonstrate factorial computation."""
    # Define the number for which we want to compute the factorial
    number = 5

    # Call the factorial function and store the result
    factorial_value = compute_factorial(number)

    # Display the result
    print(f"Factorial of {number} is: {factorial_value}")

# Entry point of the program
if __name__ == "__main__":
    main()
```

Output:



```
Factorial of 5 is: 120
Factorial of 7 is: 5040
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

The program was modularized by creating a separate compute_factorial() function, allowing the same logic to be reused easily. The main() block clearly separates execution flow from logic, improving code structure. Meaningful function and variable names make the program easy to understand. Inline comments improve readability and help explain each step of the computation.

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

Criteria	**Procedural Code (Without Functions)**	**Modular Code (With Functions)**
Logic Clarity	Logic is written in one block, so it is easy to understand for small programs.	Logic is clearly separated into functions, making the flow more organized.
Reusability	Code cannot be reused easily and must be rewritten each time.	Functions can be reused in multiple programs without rewriting code.
Debugging Ease	Debugging becomes difficult as the program grows larger.	Easier to debug because errors can be traced inside specific functions.
Suitability for Large Projects	Not suitable for large projects due to lack of structure.	Suitable for large projects because code is modular and well-structured.
AI Dependency Risk	More dependent on AI for rewriting code again and again.	Less dependent on AI since reusable functions reduce repeated generation.

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

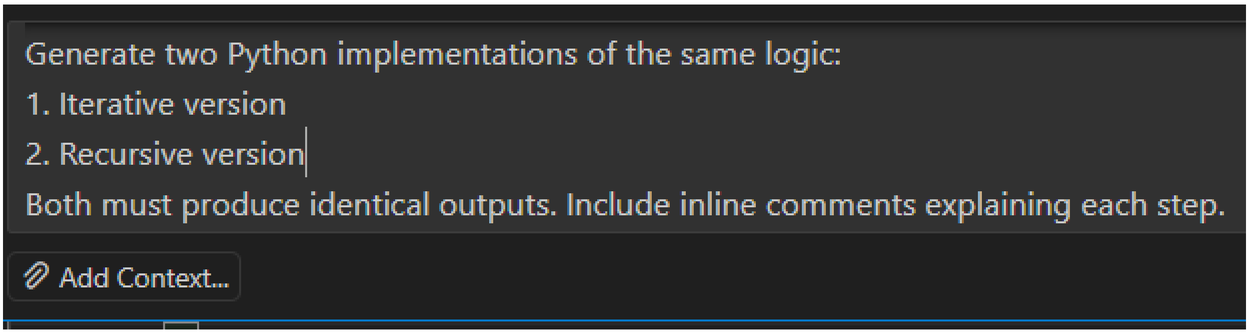
A recursive version of the same logic

❖ Constraints

Both implementations must produce identical outputs

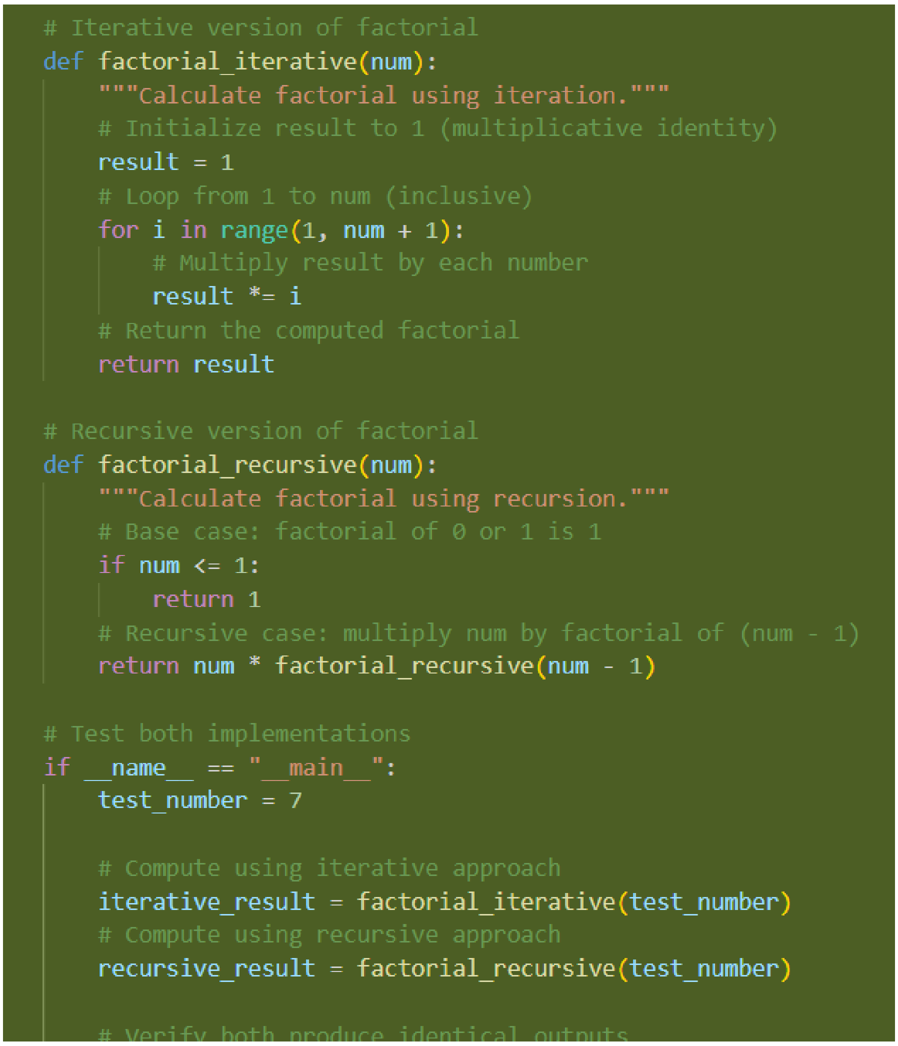
Students must not manually write the code first

Prompt:



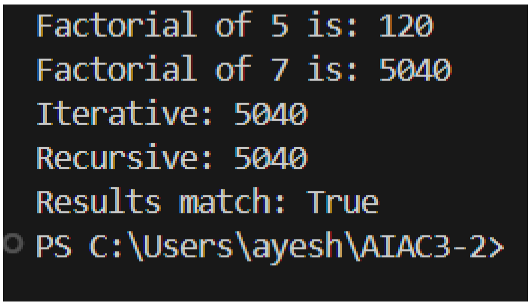
Generate two Python implementations of the same logic:
1. Iterative version
2. Recursive version
Both must produce identical outputs. Include inline comments explaining each step.

Generated Code:



```python
# Iterative version of factorial
def factorial_iterative(num):
    """Calculate factorial using iteration."""
    # Initialize result to 1 (multiplicative identity)
    result = 1
    # Loop from 1 to num (inclusive)
    for i in range(1, num + 1):
        # Multiply result by each number
        result *= i
    # Return the computed factorial
    return result

# Recursive version of factorial
def factorial_recursive(num):
    """Calculate factorial using recursion."""
    # Base case: factorial of 0 or 1 is 1
    if num <= 1:
        return 1
    # Recursive case: multiply num by factorial of (num - 1)
    return num * factorial_recursive(num - 1)

# Test both implementations
if __name__ == "__main__":
    test_number = 7

    # Compute using iterative approach
    iterative_result = factorial_iterative(test_number)
    # Compute using recursive approach
    recursive_result = factorial_recursive(test_number)

    # Verify both produce identical outputs
```

Output:



```
Factorial of 5 is: 120
Factorial of 7 is: 5040
Iterative: 5040
Recursive: 5040
Results match: True
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

The program calculates factorial using both iterative and recursive methods. In the iterative approach, a loop multiplies numbers step by step, while in the recursive approach the function keeps calling itself until the base case is reached. Both methods give the same output.

The iterative version is easier to understand and more efficient since it doesn't use extra memory. The recursive version uses the call stack and can be slow or cause errors for large inputs, so it's not recommended in such cases.