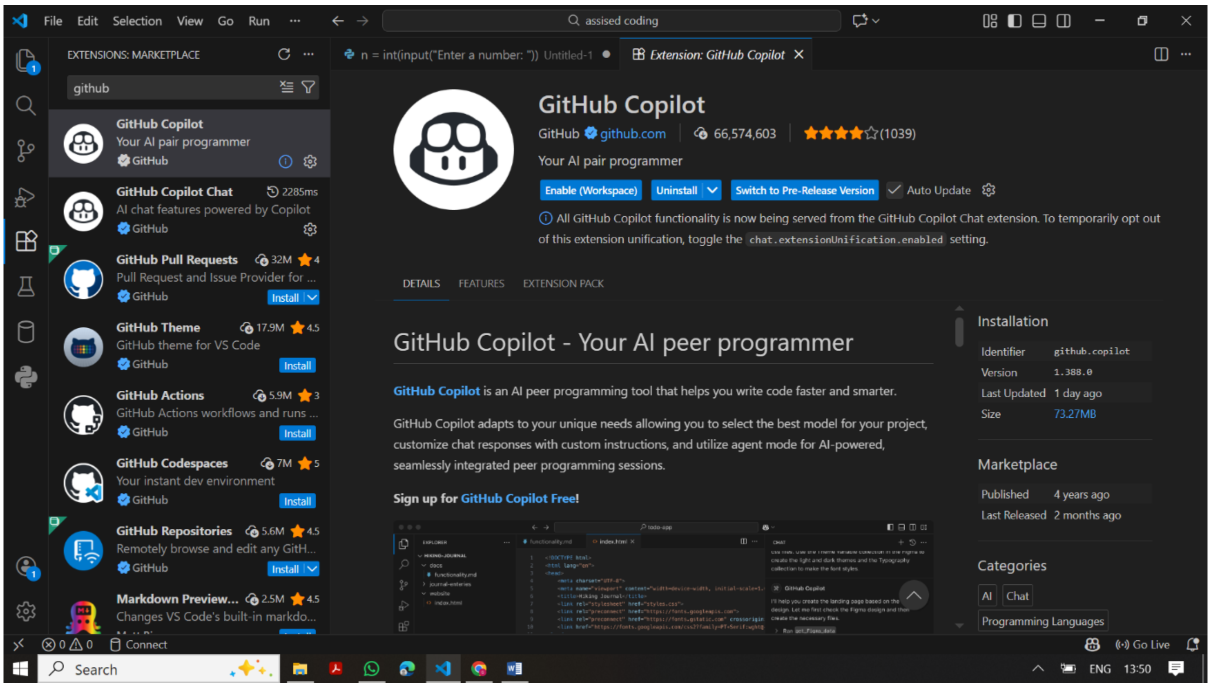
ASSIGNMENT-1

2303A51893

8-1-26

Task-1:

Install and configure Git Hub Copilot in VS Code. Take screenshots of each step.



GitHub Copilot
Your AI pair programmer
GitHub

GitHub Copilot Chat 2285ms
AI chat features powered by Copilot
GitHub

GitHub Pull Requests 32M 4
Pull Request and Issue Provider for ...
GitHub

GitHub Theme 17.9M 4.5
GitHub theme for VS Code
GitHub

GitHub Actions 5.9M 3
GitHub Actions workflows and runs ...
GitHub

GitHub Codespaces 7M 5
Your instant dev environment
GitHub

GitHub Repositories 5.6M 4.5
Remotely browse and edit any GitH...
GitHub

Markdown Preview... 2.5M 4.5
Changes VS Code's built-in markdo...

GitHub Copilot

GitHub github.com | 66,574,603 | (1039)

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	1 day ago
Size	73.27MB

Marketplace

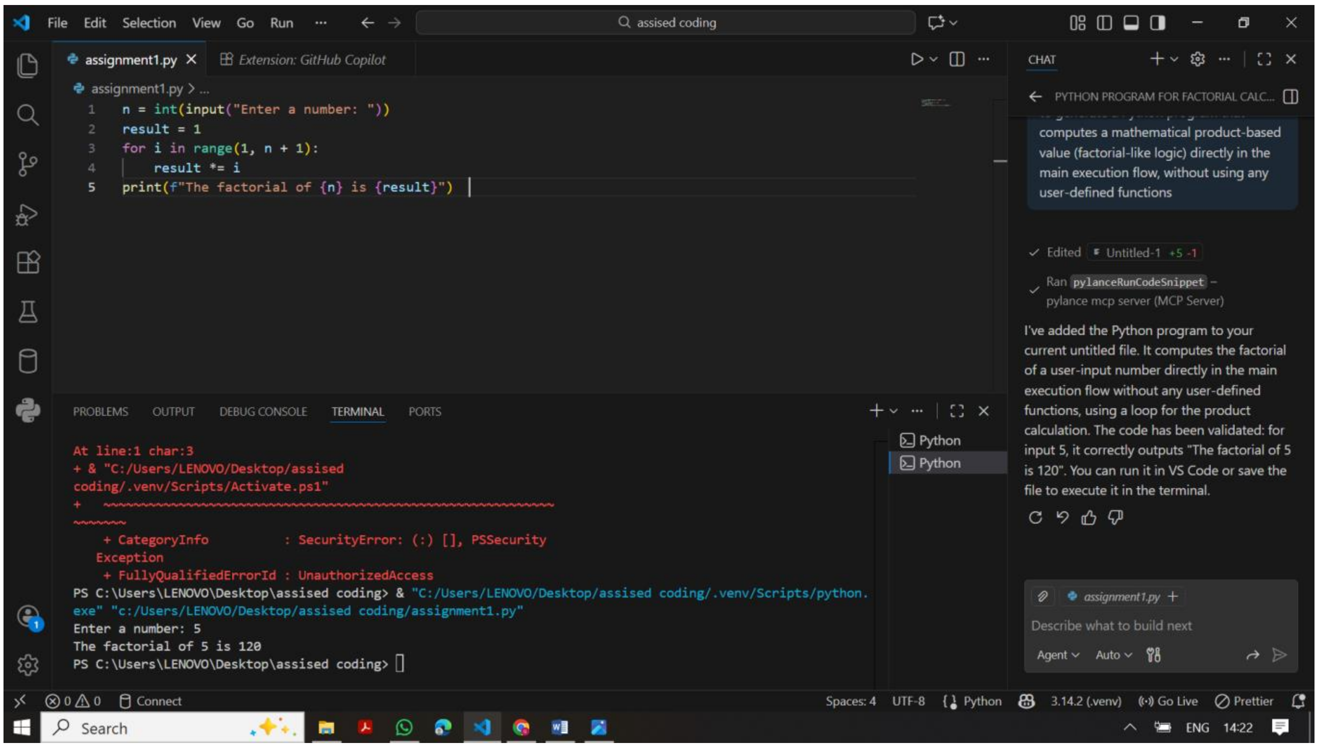
Published	4 years ago
Last Released	2 months ago

Categories

AI Chat Programming Languages

0 0

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



```python
n = int(input("Enter a number: "))
result = 1
for i in range(1, n + 1):
    result *= i
print(f"The factorial of {n} is {result}")
```

```
At line:1 char:3
+ & "C:/Users/LENOVO/Desktop/assised
coding/.venv/Scripts/Activate.ps1"
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurity
   Exception
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.
exe" "c:/Users/LENOVO/Desktop/assised coding/assignment1.py"
Enter a number: 5
The factorial of 5 is 120
PS C:\Users\LENOVO\Desktop\assised coding>
```

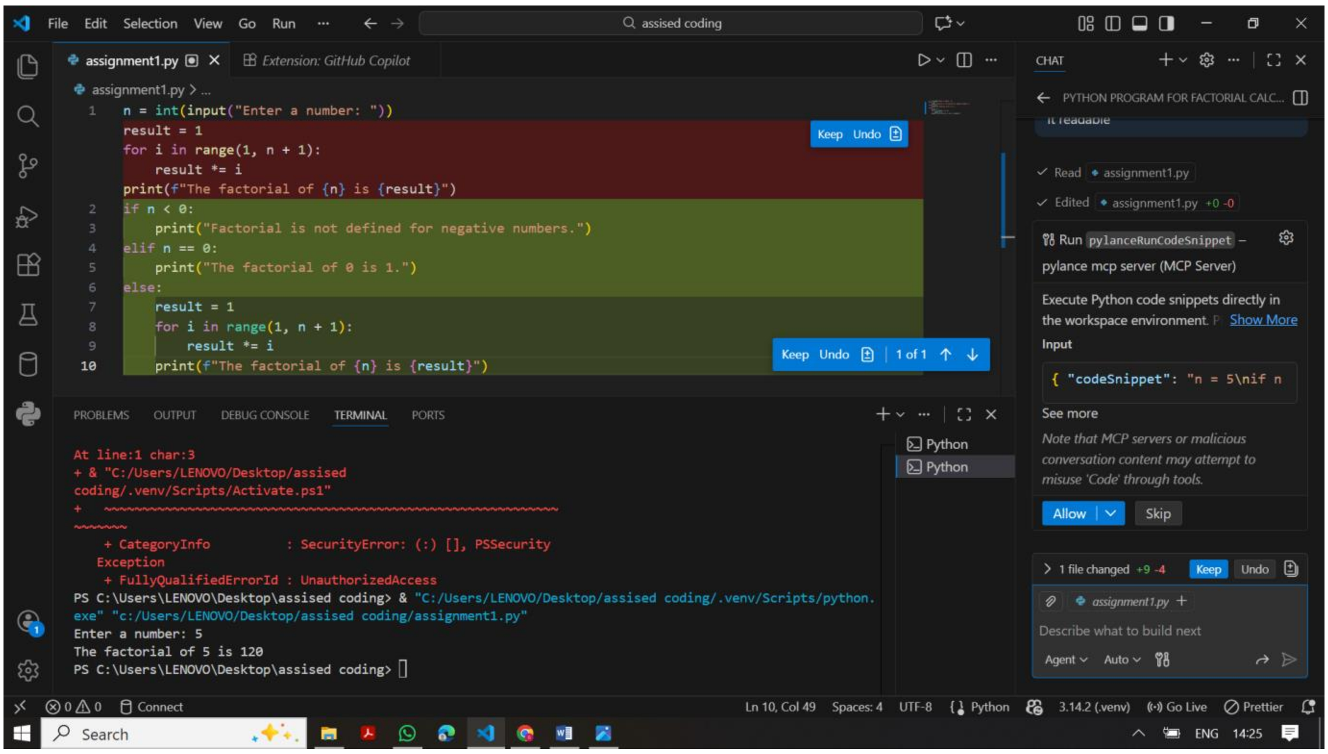
PYTHON PROGRAM FOR FACTORIAL CALC...

computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions

✓ Edited Untitled-1 +5 -1

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

I've added the Python program to your current untitled file. It computes the factorial of a user-input number directly in the main execution flow without any user-defined functions, using a loop for the product calculation. The code has been validated: for input 5, it correctly outputs "The factorial of 5 is 120". You can run it in VS Code or save the file to execute it in the terminal.



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

```
At line:1 char:3
+ & "C:/Users/LENOVO/Desktop/assised
coding/.venv/Scripts/Activate.ps1"
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurity
   Exception
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.
exe" "c:/Users/LENOVO/Desktop/assised coding/assignment1.py"
Enter a number: 5
The factorial of 5 is 120
PS C:\Users\LENOVO\Desktop\assised coding>
```

✓ Read assignment1.py

✓ Edited assignment1.py +0 -0

Run `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

Execute Python code snippets directly in the workspace environment. Show More

Input

`{ "codeSnippet": "n = 5\nif n`

See more

Note that MCP servers or malicious conversation content may attempt to misuse 'Code' through tools.

Allow | Skip

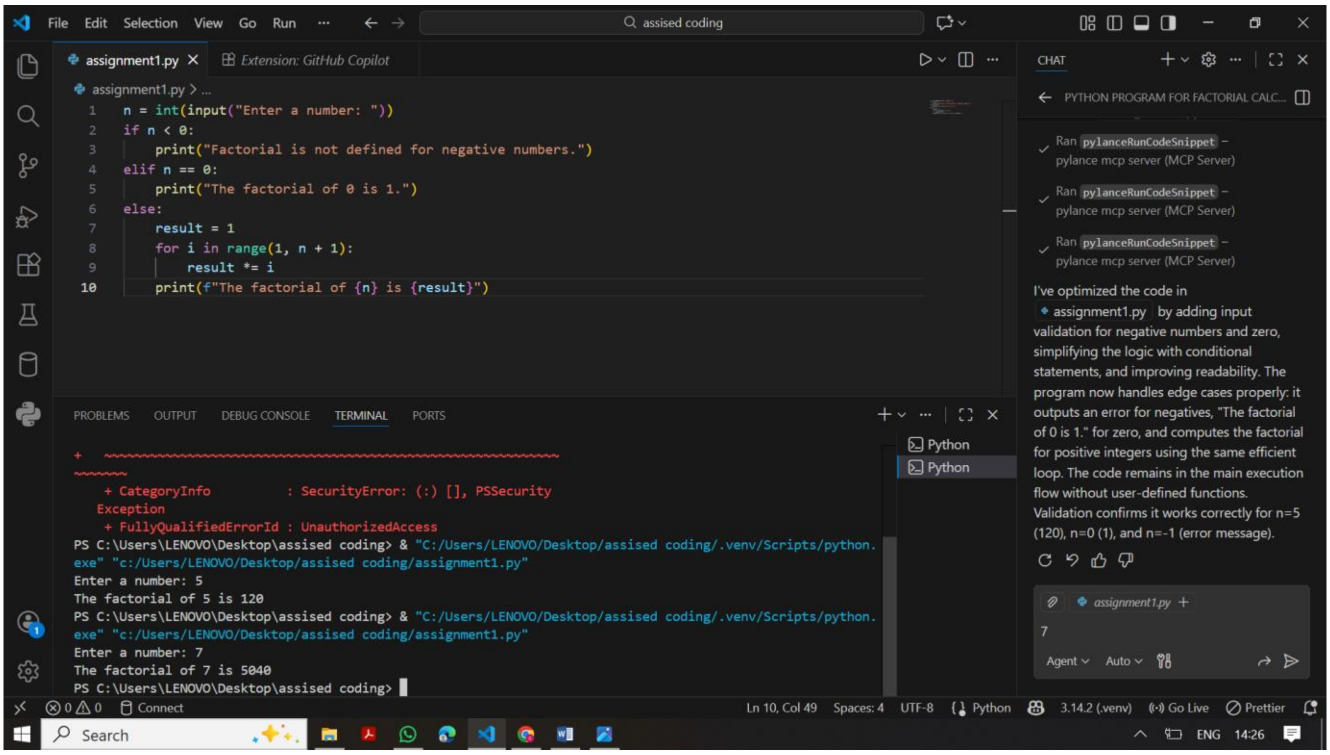
1 file changed +9 -4 Keep Undo

The program takes a number from the user as input.
It uses a loop to multiply numbers from 1 to the given number.
The result of this multiplication gives the factorial value.
Finally, it prints the factorial of the entered number.

How helpful was Copilot for a beginner?
Copilot is very helpful for a beginner because it quickly suggests correct and readable code, reducing the time needed to think about syntax.
It makes learning easier by providing a working structure, like loops and variable initialization, as seen in the factorial program.

Did it follow best practices automatically?
Copilot generally follows best practices such as clear variable names and simple logic flow.
However, beginners still need to understand the code instead of blindly accepting suggestions.
It may not always handle edge cases (like negative numbers) unless explicitly prompted.
Overall, Copilot is a strong learning aid but not a replacement for conceptual understanding.

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


File Edit Selection View Go Run
assised coding
assignment1.py
Extension: GitHub Copilot
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
Python
CHAT
PYTHON PROGRAM FOR FACTORIAL CALC...
Ln 10, Col 49 Spaces: 4 UTF-8 Python 3.14.2 (.venv) Go Live Prettier
Search
ENG 14:26

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

```
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurity
   Exception
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.
exe" "c:/Users/LENOVO/Desktop/assised coding/assignment1.py"
Enter a number: 5
The factorial of 5 is 120
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.
exe" "c:/Users/LENOVO/Desktop/assised coding/assignment1.py"
Enter a number: 7
The factorial of 7 is 5040
PS C:\Users\LENOVO\Desktop\assised coding>
```

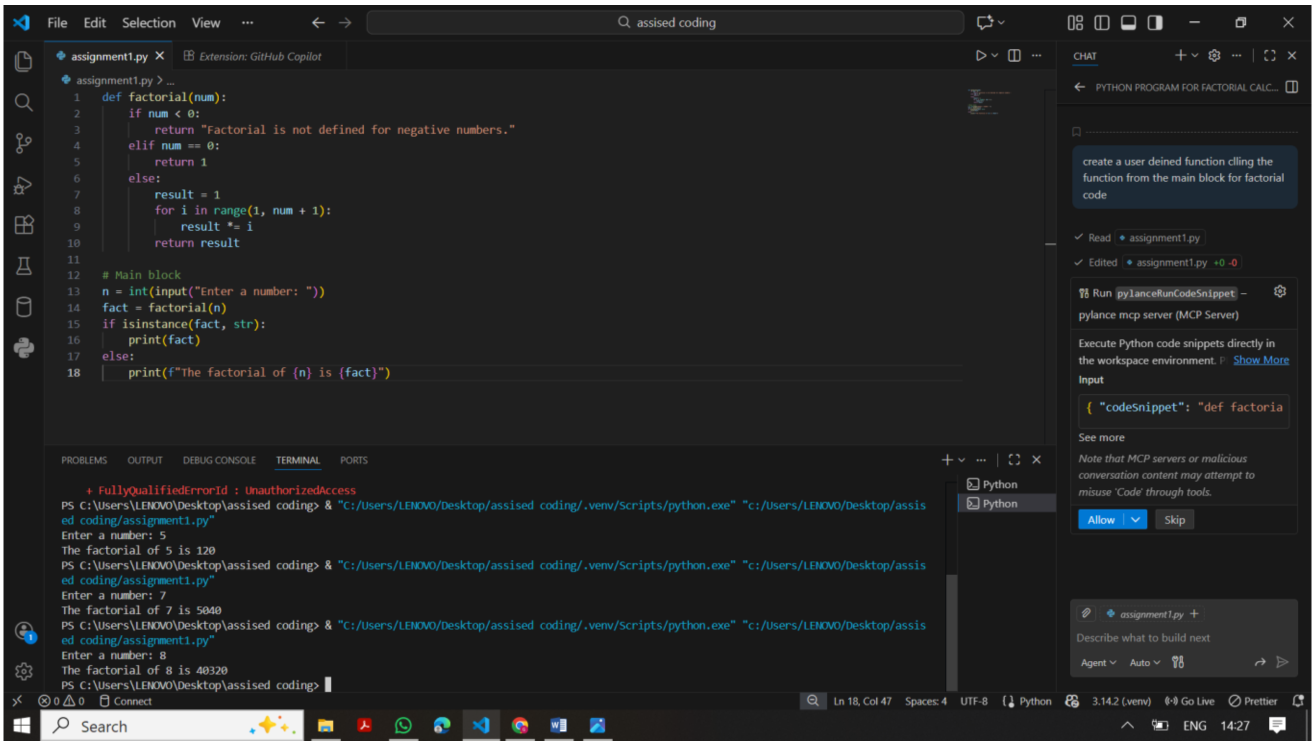
✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

I've optimized the code in assignment1.py by adding input validation for negative numbers and zero, simplifying the logic with conditional statements, and improving readability. The program now handles edge cases properly: it outputs an error for negatives, "The factorial of 0 is 1." for zero, and computes the factorial for positive integers using the same efficient loop. The code remains in the main execution flow without user-defined functions. Validation confirms it works correctly for n=5 (120), n=0 (1), and n=-1 (error message).

assignment1.py
7
Agent Auto



```python
def factorial(num):
    if num < 0:
        return "Factorial is not defined for negative numbers."
    elif num == 0:
        return 1
    else:
        result = 1
        for i in range(1, num + 1):
            result *= i
        return result

# Main block
n = int(input("Enter a number: "))
fact = factorial(n)
if isinstance(fact, str):
    print(fact)
else:
    print(f"The factorial of {n} is {fact}")
```

```
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 5
The factorial of 5 is 120
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 7
The factorial of 7 is 5040
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 8
The factorial of 8 is 40320
PS C:\Users\LENOVO\Desktop\assised coding>
```

CHAT — PYTHON PROGRAM FOR FACTORIAL CALC...

create a user deined function clling the function from the main block for factorial code

✓ Read assignment1.py

✓ Edited assignment1.py +0 -0

Run `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

Execute Python code snippets directly in the workspace environment. Show More

Input

```
{ "codeSnippet": "def factoria
```

See more

Note that MCP servers or malicious conversation content may attempt to misuse 'Code' through tools.

What was improved?

Added proper input validation to handle negative numbers and the special case of 0.

Removed duplicate code and structured the program with clear conditional branches.

Ensured the program prints the final result in all valid cases.

Improved clarity and flow of the logic.

Why the new version is better (readability, performance, maintainability)?

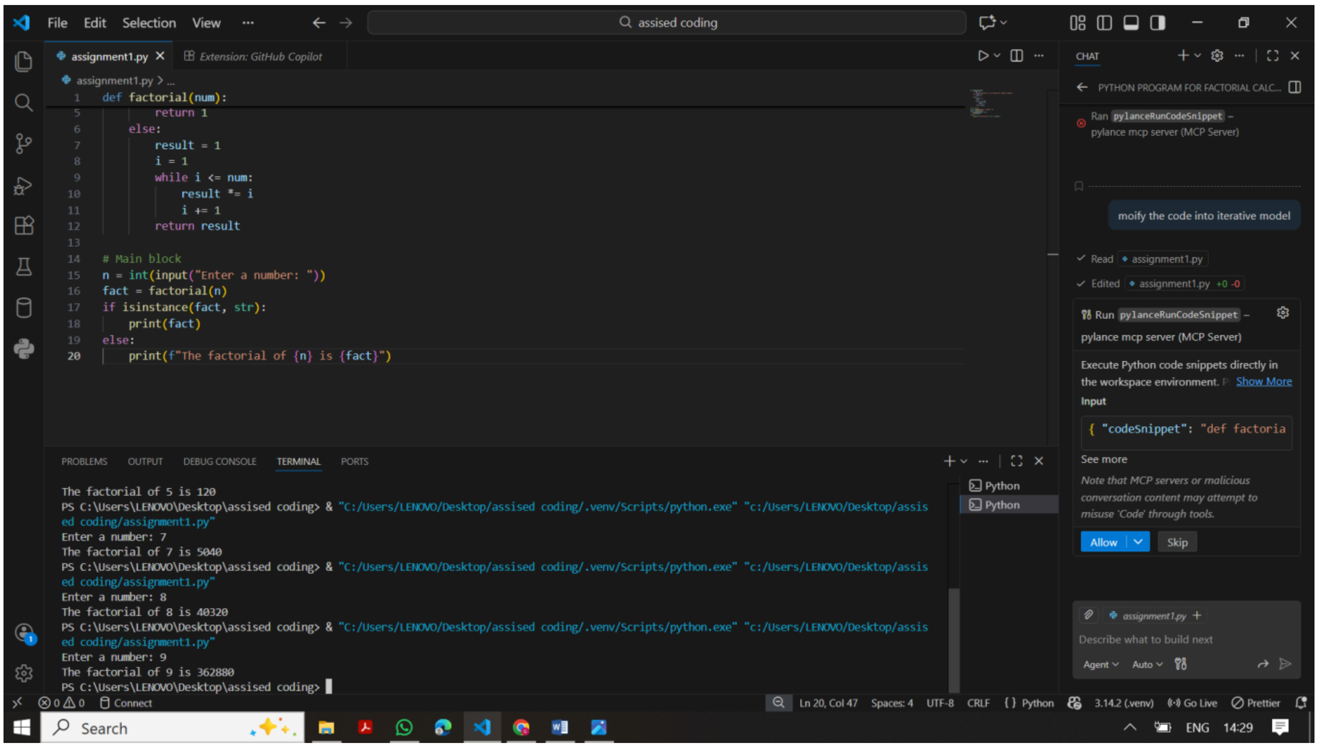
Readability: Clear if–elif–else structure makes the logic easy to understand.

Performance: Avoids unnecessary calculations for invalid or zero inputs.

Maintainability: Well-organized conditions make future changes or enhancements easier.

Reliability: Handles edge cases correctly, reducing runtime errors.

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



```python
def factorial(num):
        return 1
    else:
        result = 1
        i = 1
        while i <= num:
            result *= i
            i += 1
        return result

# Main block
n = int(input("Enter a number: "))
fact = factorial(n)
if isinstance(fact, str):
    print(fact)
else:
    print(f"The factorial of {n} is {fact}")
```

```
The factorial of 5 is 120
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 7
The factorial of 7 is 5040
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 8
The factorial of 8 is 40320
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 9
The factorial of 9 is 362880
PS C:\Users\LENOVO\Desktop\assised coding>
```

PYTHON PROGRAM FOR FACTORIAL CALC...

Ran pylanceRunCodeSnippet – pylance mcp server (MCP Server)

moify the code into iterative model

✓ Read assignment1.py

✓ Edited assignment1.py +0 -0

Run pylanceRunCodeSnippet – pylance mcp server (MCP Server)

Execute Python code snippets directly in the workspace environment. P Show More

Input

`{ "codeSnippet": "def factoria`

See more

Note that MCP servers or malicious conversation content may attempt to misuse 'Code' through tools.

File Edit Selection View
assised coding
assignment1.py
Extension: GitHub Copilot
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
Python
CHAT
Allow
Skip
Describe what to build next
Agent
Auto
Ln 20, Col 47 Spaces: 4 UTF-8 CRLF Python 3.14.2 (.venv) Go Live Prettier
Search
ENG 14:29

How Modularity Improves Reusability
Modularity means breaking a program into smaller, independent, and reusable units (functions or modules).
How it improves reusability:
A function can be reused multiple times without rewriting code.
The same function can be imported into other programs.
Changes made in one place (the function) automatically reflect everywhere it's used.
Code becomes easier to test, debug, and maintain.
In short, modularity avoids duplication and promotes clean, reusable logic.

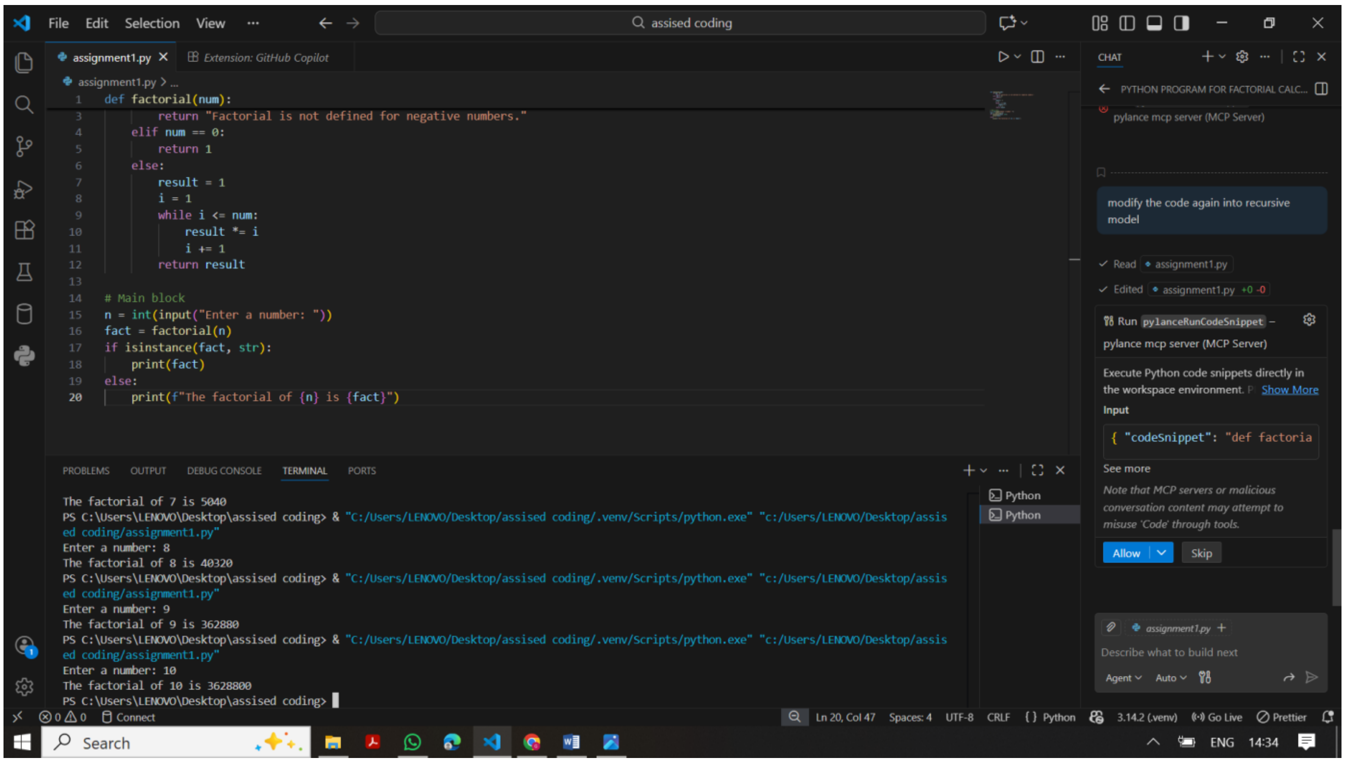
Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions.

Technical Report: Comparison of Non-Function and Function-Based Copilot-Generated Programs
This report compares non-function-based and function-based Copilot-generated Python programs for calculating factorials, focusing on clarity, reusability, debugging, scalability, and AI dependency risk.
The non-function-based program implements the factorial logic directly in the main flow along with input and output operations. While this approach works for small scripts, it reduces logic clarity because computation, validation, and user interaction are mixed together. The code is also repeated, which makes understanding and maintaining it difficult. Any change in logic must be applied in multiple places, increasing the chance of errors.
In contrast, the function-based program separates the factorial logic into a dedicated function. This improves logic clarity by clearly defining responsibilities within the code. The function can be reused multiple times or imported into other programs, significantly improving reusability.
Debugging is easier in the function-based approach because errors can be isolated within the function. The non-function-based version makes debugging harder due to duplicated and unstructured logic.

For large projects, non-function-based code does not scale well, whereas function-based code supports modularity, testing, and teamwork.
Finally, AI dependency risk is higher in non-function-based programs, as repeated AI-generated code

may introduce redundancy and inconsistencies. Function-based code minimizes this risk by promoting standardized and reusable logic.
Overall, function-based programs are more reliable, maintainable, and suitable for long-term development.

Task 5: AI-Generated Iterative vs Recursive Thinking
Iterative model



```python
def factorial(num):
        return "Factorial is not defined for negative numbers."
    elif num == 0:
        return 1
    else:
        result = 1
        i = 1
        while i <= num:
            result *= i
            i += 1
        return result

# Main block
n = int(input("Enter a number: "))
fact = factorial(n)
if isinstance(fact, str):
    print(fact)
else:
    print(f"The factorial of {n} is {fact}")
```

```
The factorial of 7 is 5040
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 8
The factorial of 8 is 40320
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 9
The factorial of 9 is 362880
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 10
The factorial of 10 is 3628800
PS C:\Users\LENOVO\Desktop\assised coding>
```

pylance mcp server (MCP Server)

modify the code again into recursive model

✓ Read assignment1.py
✓ Edited assignment1.py +0 -0

Run `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

Execute Python code snippets directly in the workspace environment. P Show More

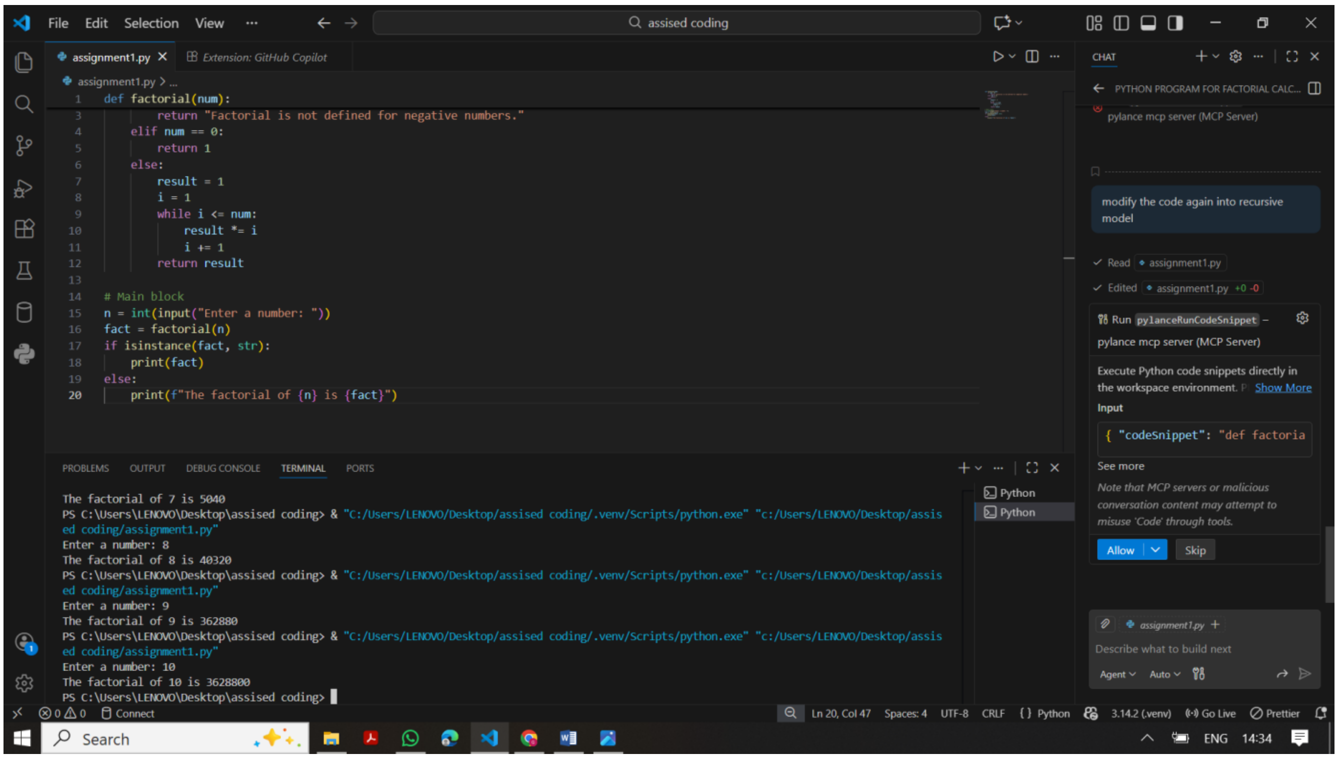
Input

```
{ "codeSnippet": "def factoria
```

See more

Note that MCP servers or malicious conversation content may attempt to misuse 'Code' through tools.

Recursive
Model



```python
def factorial(num):
        return "Factorial is not defined for negative numbers."
    elif num == 0:
        return 1
    else:
        result = 1
        i = 1
        while i <= num:
            result *= i
            i += 1
        return result

# Main block
n = int(input("Enter a number: "))
fact = factorial(n)
if isinstance(fact, str):
    print(fact)
else:
    print(f"The factorial of {n} is {fact}")
```

```
The factorial of 7 is 5040
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 8
The factorial of 8 is 40320
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 9
The factorial of 9 is 362880
PS C:\Users\LENOVO\Desktop\assised coding> & "C:/Users/LENOVO/Desktop/assised coding/.venv/Scripts/python.exe" "c:/Users/LENOVO/Desktop/assis
ed coding/assignment1.py"
Enter a number: 10
The factorial of 10 is 3628800
PS C:\Users\LENOVO\Desktop\assised coding>
```

PYTHON PROGRAM FOR FACTORIAL CALC...

pylance mcp server (MCP Server)

modify the code again into recursive model

✓ Read assignment1.py

✓ Edited assignment1.py +0 -0

Run `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

Execute Python code snippets directly in the workspace environment. P Show More

Input

`{ "codeSnippet": "def factoria`

See more

Note that MCP servers or malicious conversation content may attempt to misuse 'Code' through tools.

Allow | Skip

Below is a clear execution flow explanation and comparison of the two AI-generated implementations (Iterative and Recursive factorial).

Execution Flow Explanation

Iterative Model

The program takes an integer input from the user.

The calculate_factorial(num) function checks:

If the number is negative → returns an error message.

If the number is 0 → returns 1.

For positive numbers, a loop runs from 1 to num.

The factorial value is updated step by step using multiplication.

The final result is returned and printed.

Flow: Input → Loop-based multiplication → Outpu

Recursive Model

The program takes an integer input from the user.

The calculate_factorial_recursive(num) function checks:

If the number is negative → returns an error message.

If the number is 0 → returns 1 (base case).

For positive numbers, the function calls itself with num - 1.

Recursive calls continue until the base case is reached.

Results are multiplied while returning back through the call stack.

Flow: Input → Repeated function calls → Base case → Backtracking → Output

Comparison

Readability

Iterative: More straightforward and easier for beginners to understand.

Recursive: Shorter and mathematically elegant, but harder to trace.

Stack Usage

Iterative: Uses constant memory (no call stack growth).
Recursive: Uses stack memory for each function call.
Performance Implications
Iterative: Faster and more memory-efficient.
Recursive: Slightly slower due to function call overhead.
When Recursion Is Not Recommended
When input size is large (risk of stack overflow).
When performance and memory efficiency are critical.
In environments with limited stack memory.