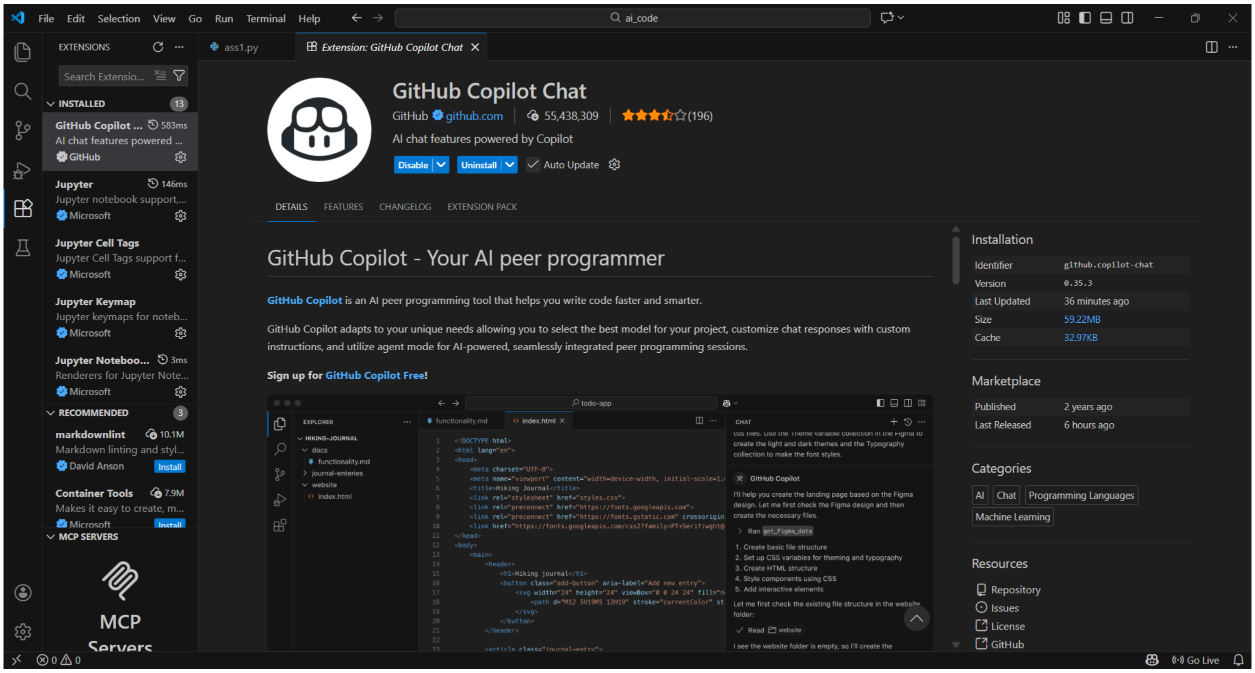
AI ASSISTED CODING ASSIGNMENT - 1

ROLL NO. 2303A510J0

Task 0: Install and configure GitHub Copilot in VS Code



GitHub Copilot Chat

GitHub github.com | 55,438,309 | ★★★★☆ (196)

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	0.35.3
Last Updated	36 minutes ago
Size	59.22MB
Cache	32.97KB

Marketplace

Published	2 years ago
Last Released	6 hours ago

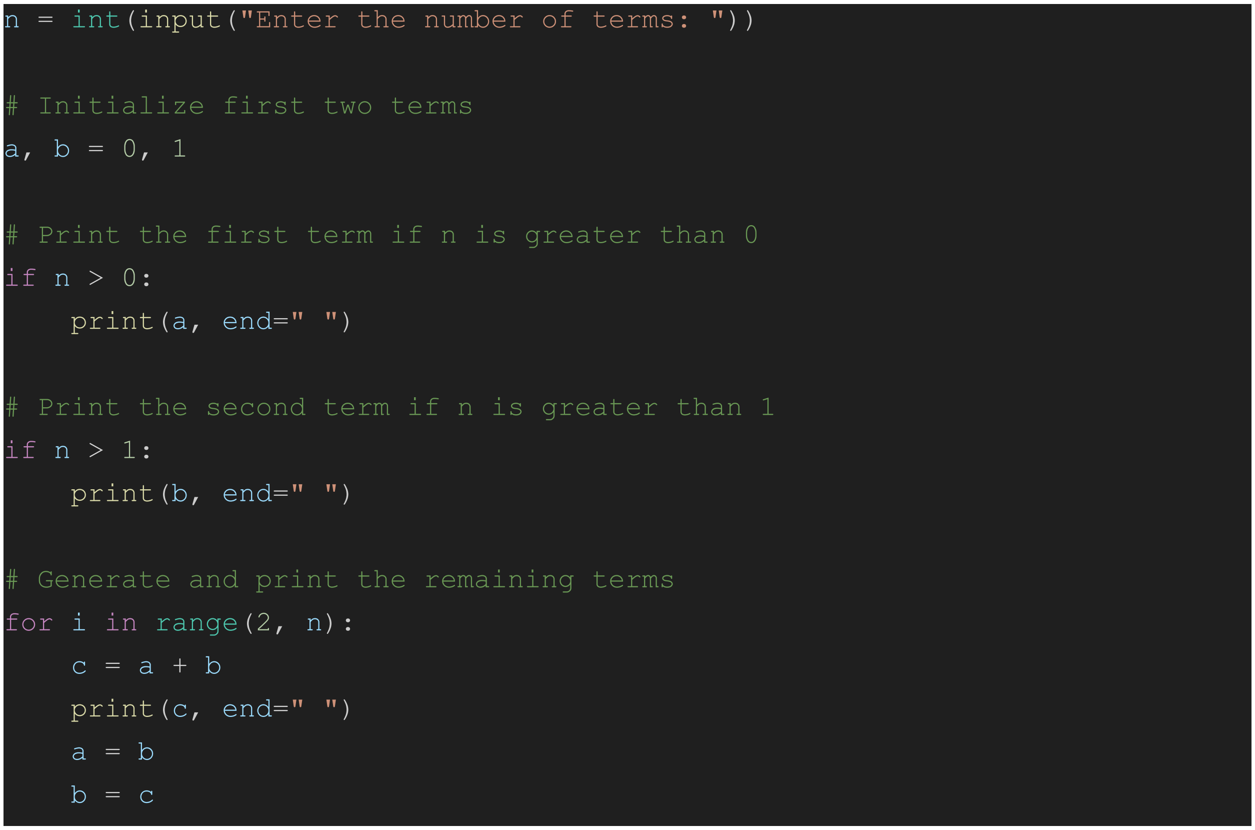
Categories

AI Chat Programming Languages Machine Learning

Resources

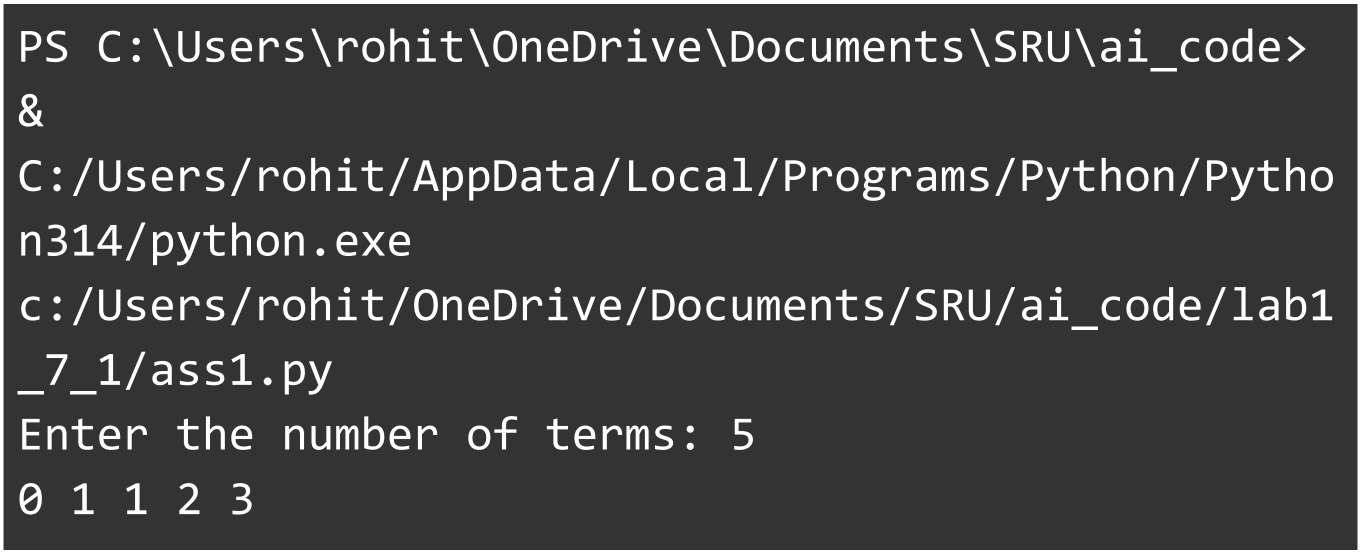
Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)

```
# Write a Python program to print Fibonacci sequence up to n terms
# Take user input
# Do not use any user-defined functions
```



```python
n = int(input("Enter the number of terms: "))

# Initialize first two terms
a, b = 0, 1

# Print the first term if n is greater than 0
if n > 0:
    print(a, end=" ")

# Print the second term if n is greater than 1
if n > 1:
    print(b, end=" ")

# Generate and print the remaining terms
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
```

Output:



```
PS C:\Users\rohit\OneDrive\Documents\SRU\ai_code>
&
C:/Users/rohit/AppData/Local/Programs/Python/Pytho
n314/python.exe
c:/Users/rohit/OneDrive/Documents/SRU/ai_code/lab1
_7_1/ass1.py
Enter the number of terms: 5
0 1 1 2 3
```

Explanation:

The program directly implements Fibonacci logic in the main block.

It uses variables a and b to store previous values.

No functions are used, making it procedural.

GitHub Copilot generated the logic based on comments.

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

```
# Optimize this Fibonacci code
# Simplify variable usage
# Improve readability
n = int(input("Enter number of terms: "))

prev, curr = 0, 1

for _ in range(n):
    print(prev, end=" ")
    prev, curr = curr, prev + curr
print()
```

Output:

```
PS
C:\Users\rohit\OneDrive\Documents\SRU\ai_code>
&
C:/Users/rohit/AppData/Local/Programs/Python/P
ython314/python.exe
c:/Users/rohit/OneDrive/Documents/SRU/ai_code/
lab1_7_1/ass1.py
Enter number of terms: 5
0 1 1 2 3
```

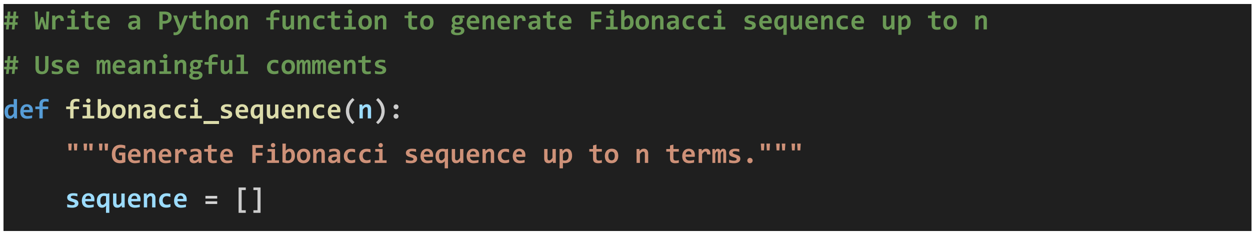
Explanation:
Reduced number of variables.

Cleaner tuple assignment.

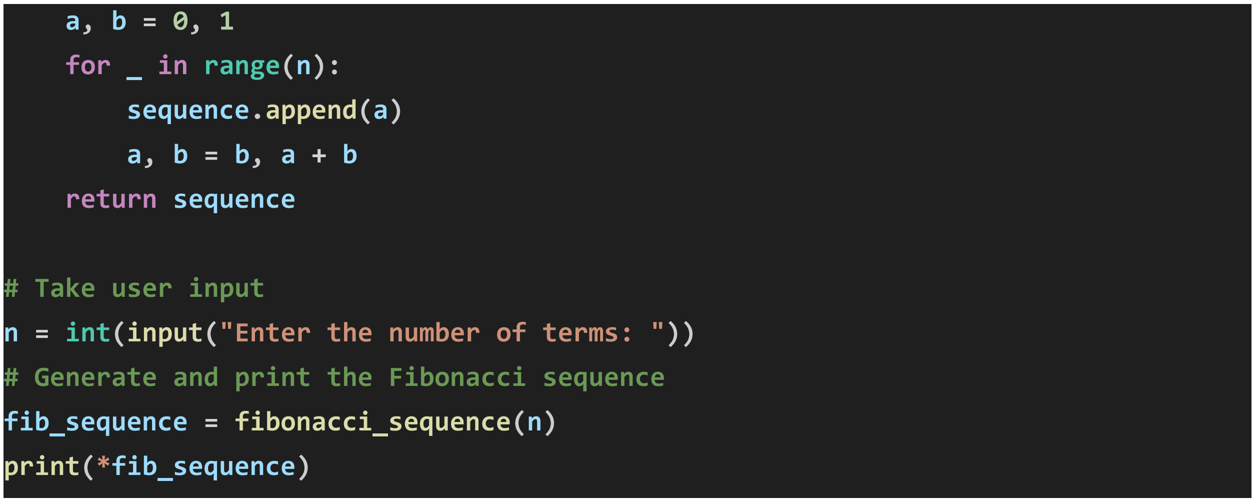
More readable and concise.

Better performance due to fewer operations.

Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)



```python
# Write a Python function to generate Fibonacci sequence up to n
# Use meaningful comments
def fibonacci_sequence(n):
    """Generate Fibonacci sequence up to n terms."""
    sequence = []
```



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

# Take user input
n = int(input("Enter the number of terms: "))
# Generate and print the Fibonacci sequence
fib_sequence = fibonacci_sequence(n)
print(*fib_sequence)
```

Output:

```
PS
C:\Users\rohit\OneDrive\Documents\SRU\ai_code>
&
C:/Users/rohit/AppData/Local/Programs/Python/P
ython314/python.exe
c:/Users/rohit/OneDrive/Documents/SRU/ai_code/
lab1_7_1/ass1.py
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
```

Explanation:

Fibonacci logic is encapsulated in a function.

Improves reusability and clarity.

Easy to maintain and test.

Copilot generated comments and clean structure.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Feature	Without Functions	With Functions
Code clarity	Moderate	High
Reusability	No	Yes
Debugging	Difficult	Easy
Readability	Average	Better
Large systems	Not suitable	Suitable

Analysis:

- Modular code is preferred in real-world applications.
- Functions allow reuse and easier debugging.
- Procedural code is suitable only for small scripts.

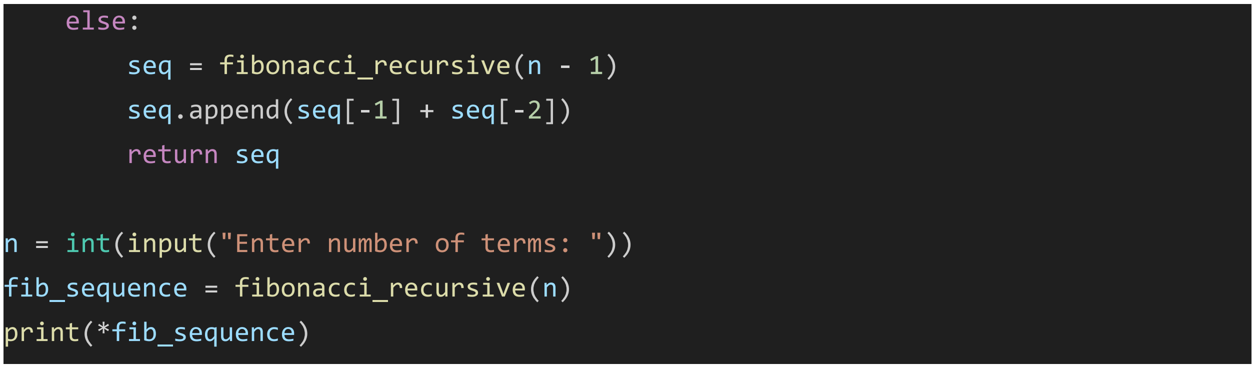
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

```
# Write an iterative Fibonacci program in Python
def fibonacci_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b
n = int(input("Enter number of terms: "))
fibonacci_iterative(n)
```

Output:

```
PS
C:\Users\rohit\OneDrive\Documents\SRU\ai_code>
&
C:/Users/rohit/AppData/Local/Programs/Python/P
ython314/python.exe
c:/Users/rohit/OneDrive/Documents/SRU/ai_code/
lab1_7_1/ass1.py
Enter number of terms: 10
0 1 1 2 3 5 8 13 21 34
```

```
# Write a recursive Fibonacci program in Python
def fibonacci_recursive(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
```



```python
    else:
        seq = fibonacci_recursive(n - 1)
        seq.append(seq[-1] + seq[-2])
        return seq

n = int(input("Enter number of terms: "))
fib_sequence = fibonacci_recursive(n)
print(*fib_sequence)
```

Output:

```
PS
C:\Users\rohit\OneDrive\Documents\SRU\ai_code>
&
C:/Users/rohit/AppData/Local/Programs/Python/P
ython314/python.exe
c:/Users/rohit/OneDrive/Documents/SRU/ai_code/
lab1_7_1/ass1.py
Enter number of terms: 6
0 1 1 2 3 5
```

Execution Flow Explanation

- **Iterative:** Uses a loop and updates values step-by-step.
- **Recursive:** Calls itself repeatedly until base condition is met.

Comparison

Aspect	Iterative	Recursive
Time Complexity	O(n)	$O(2^n)$
Space Complexity	O(1)	O(n)
Performance	Fast	Slow
Large n	Recommended	Avoided

Why recursion should be avoided

- High memory usage due to stack calls.
- Poor performance for large inputs.