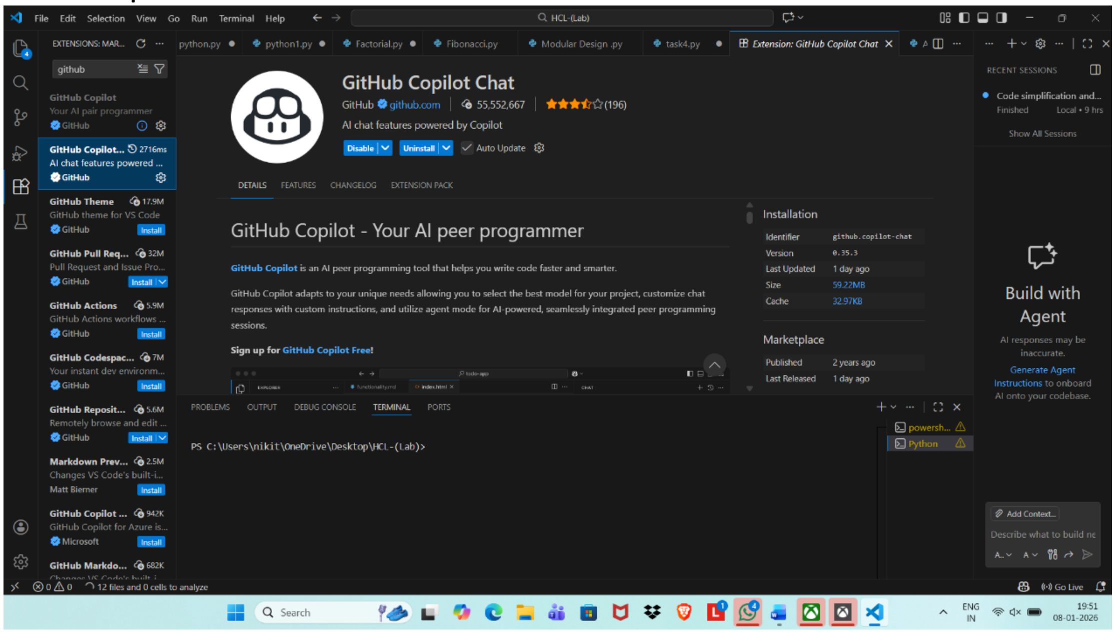
Assignment - 1

Name : K . Nikitha
H.T. NO : 2303A51405
Batch : 06

Task - 0 :

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



GitHub Copilot Chat

GitHub github.com | 55,552,667 | (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!

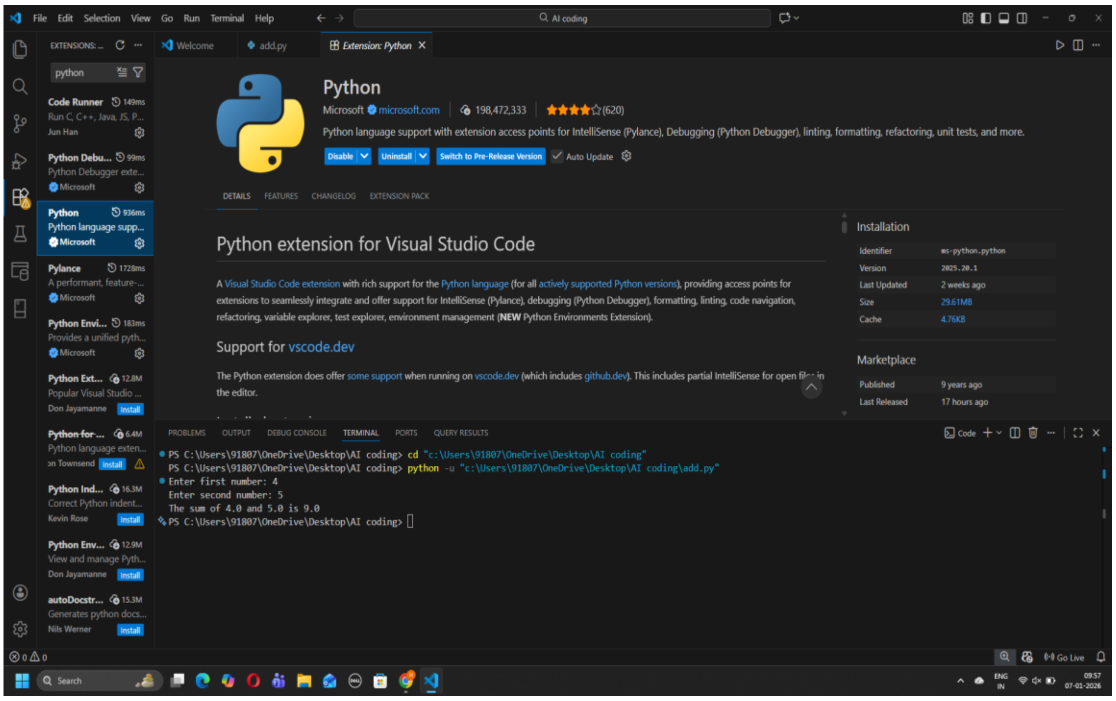
Installation

Identifier	github.copilot-chat
Version	0.35.3
Last Updated	1 day ago
Size	59.22MB
Cache	32.97KB

Marketplace

Published	2 years ago
Last Released	1 day ago

```
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)>
```



Python

Microsoft microsoft.com | 198,472,333 | (620)

Python language support with extension access points for IntelliSense (Pylance), Debugging (Python Debugger), linting, formatting, refactoring, unit tests, and more.

Python extension for Visual Studio Code

A Visual Studio Code extension with rich support for the Python language (for all actively supported Python versions), providing access points for extensions to seamlessly integrate and offer support for IntelliSense (Pylance), debugging (Python Debugger), formatting, linting, code navigation, refactoring, variable explorer, test explorer, environment management (**NEW** Python Environments Extension).

Support for vscode.dev

The Python extension does offer some support when running on vscode.dev (which includes github.dev). This includes partial IntelliSense for open fil-- in the editor.

Installation

Identifier	ms-python.python
Version	2025.20.1
Last Updated	2 weeks ago
Size	29.61MB
Cache	4.76KB

Marketplace

Published	9 years ago
Last Released	17 hours ago

```
PS C:\Users\91807\OneDrive\Desktop\AI coding> cd "c:\Users\91807\OneDrive\Desktop\AI coding"
PS C:\Users\91807\OneDrive\Desktop\AI coding> python -u "c:\Users\91807\OneDrive\Desktop\AI coding\add.py"
Enter first number: 4
Enter second number: 5
The sum of 4.0 and 5.0 is 9.0
PS C:\Users\91807\OneDrive\Desktop\AI coding> 
```

File Edit Selection View Go Run Terminal Help
AI coding
Welcome
add.py
Extension: Python
Disable
Uninstall
Switch to Pre-Release Version
Auto Update
DETAILS FEATURES CHANGELOG EXTENSION PACK
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Task - 1 :

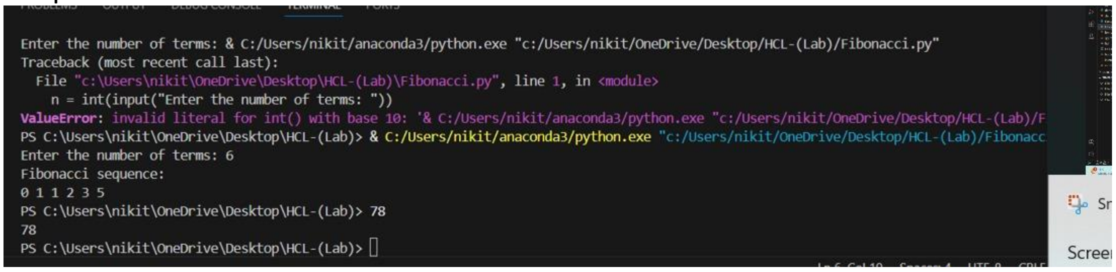
Prompt : Develop a Python program that prints the Fibonacci series up to n terms using iterative logic only, without using any user-defined functions.

Code :

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

a, b = 0, 1
```

```
print("Fibonacci series:")
for i in range(n):
    print(a, end=" ")
    c = a + b
    a = b
    b = c
```

Output:



```
Enter the number of terms: & C:/Users/nikit/anaconda3/python.exe "c:/Users/nikit/OneDrive/Desktop/HCL-(Lab)/Fibonacci.py"
Traceback (most recent call last):
  File "c:\Users\nikit\OneDrive\Desktop\HCL-(Lab)\Fibonacci.py", line 1, in <module>
    n = int(input("Enter the number of terms: "))
ValueError: invalid literal for int() with base 10: '& C:/Users/nikit/anaconda3/python.exe "c:/Users/nikit/OneDrive/Desktop/HCL-(Lab)/F
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> & C:/Users/nikit/anaconda3/python.exe "c:/Users/nikit/OneDrive/Desktop/HCL-(Lab)/Fibonacc
Enter the number of terms: 6
Fibonacci sequence:
0 1 1 2 3 5
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 78
78
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 
```

Explanation:

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

- This line asks the user to enter how many terms of the Fibonacci series we want.
- The value is stored in the variable `n`.

```
a, b = 0, 1
```

- The Fibonacci series starts with **0 and 1**.
- `a` stores the first number (0).
- `b` stores the second number (1).

```
for i in range(n):
```

- This loop runs **n times**, based on the user input.
- Each loop iteration prints one Fibonacci number.

```
print(a, end=” ” )
```

- Prints the value of a (current Fibonacci number).
- end=" " ensures all numbers are printed on the same line, separated by spaces.

```
c = a + b
a = b
b = c
```

- Adds the previous two numbers (`a` and `b`).
- Stores the result in variable `c`.
- `a` is updated to the value of `b`.
- `b` is updated to the new Fibonacci number `c`.
- This shift prepares the variables for the next loop cycle.

Task - 2 :

Prompt : Simplify this Fibonacci Python code by optimizing variable usage while keeping the same result.

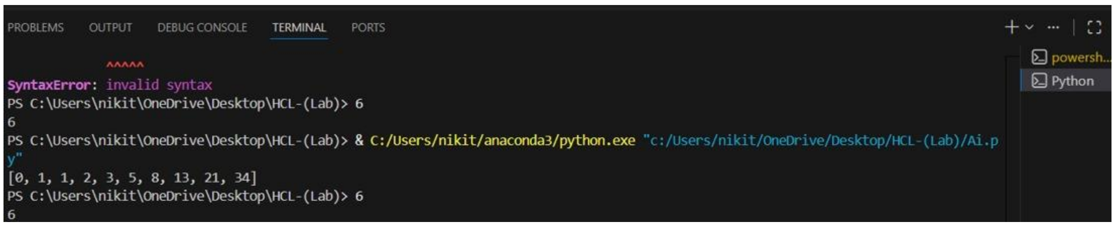
Code :

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

```python
a, b = 0, 1
```

```python
print("Fibonacci series:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
```

Output :



```
            ^^^^^
SyntaxError: invalid syntax
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 6
6
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> & C:/Users/nikit/anaconda3/python.exe "c:/Users/nikit/OneDrive/Desktop/HCL-(Lab)/Ai.p
y"
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 6
6
```

Explanation :

a, b = b, a + b

This is **tuple assignment**, which updates both variables in a single line:

a becomes the previous value of b

b becomes the sum of previous a and b

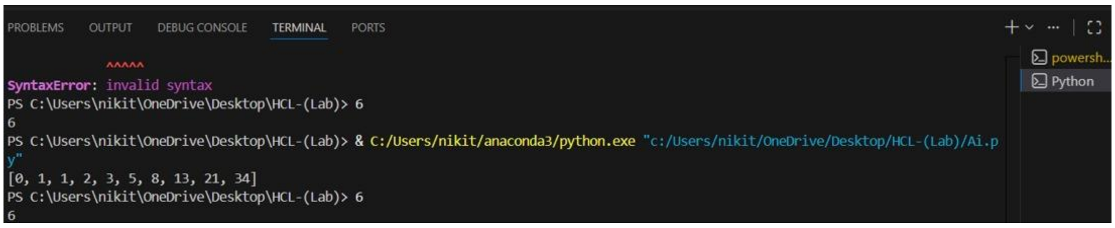
Task - 3 :

Prompt : Develop a Python program that uses a user-defined function to generate the Fibonacci series up to n terms, taking user input and including appropriate comments.

Code :

```python
# Function to generate and display the Fibonacci series up to n terms
def fibonacci(n):
    """
    Generate and print the Fibonacci series for n terms.

    Args:
        n (int): The number of terms in the Fibonacci series to 
generate
    """
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1

    print("Fibonacci series:")
    # Iterate n times to generate n terms of the Fibonacci series
    for i in range(n):
        # Print the current Fibonacci number
        print(a, end=" ")
        # Update a and b to the next two consecutive Fibonacci numbers
        a, b = b, a + b

    # Print a newline at the end for better formatting
    print()


# Main program
if __name__ == "__main__":
    # Take the number of terms from the user
    n = int(input("Enter the number of terms: "))

    # Call the fibonacci function to generate and display the series
    fibonacci(n)
```

Output :



```
             ^^^^^
SyntaxError: invalid syntax
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 6
6
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> & C:/Users/nikit/anaconda3/python.exe "c:/Users/nikit/OneDrive/Desktop/HCL-(Lab)/Ai.p
y"
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 6
6
```

Explanation :

- The program defines a **user-defined function** `fibonacci(n)` to generate the Fibonacci series.
- It initializes the first two numbers (`0` and `1`) and uses a **loop** to produce `n` terms.
- In each iteration, the current number is printed and the next Fibonacci numbers are calculated.
- The **main program** takes input from the user and calls the function.
- This approach improves **code reusability, clarity, and modularity**.

Task - 4:

Prompt: Analyze Python Fibonacci programs implemented with functions and without functions, considering code clarity, reusability, ease of debugging, and scalability.

Aspect	Without Functions (Task 1)	With Functions (Task 3)
Code Clarity	Logic is written directly in the main program, which is easy to understand for very small scripts but becomes cluttered as code grows.	Code is well-structured with a separate fibonacci() function, making it cleaner and easier to read.
Reusability	Cannot be reused easily; the Fibonacci logic must be rewritten if needed elsewhere.	Highly reusable; the fibonacci() function can be called multiple times or used in other programs.
Debugging Ease	Debugging is harder because all logic is in one place and changes may affect the entire program.	Debugging is easier since the Fibonacci logic is isolated in a function and can be tested independently.
Scalability / Suitability for Larger Systems	Suitable only for small programs or quick scripts; not ideal for large applications.	Well-suited for large applications because functions help manage complexity.

Task - 5 :

Prompt : Develop a Python program that demonstrates Fibonacci series generation using iterative and recursive approaches, with user input and proper comments.

Take input from the user and add proper comments for each approach.

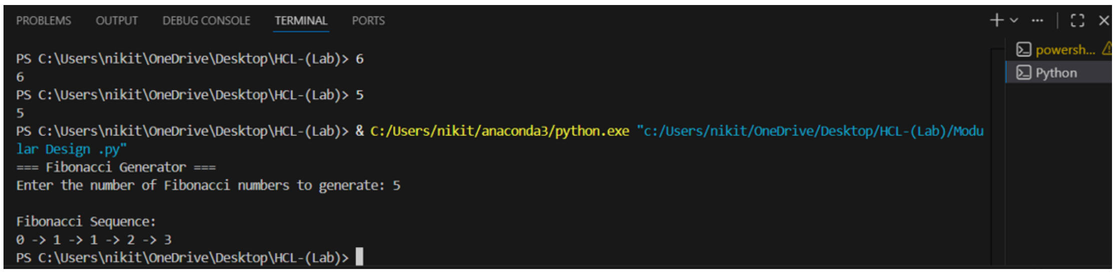
Code :

```
# ===== ITERATIVE METHOD =====
# Function to generate Fibonacci series iteratively up to n terms
def fibonacci_iterative(n):
    """
    Generate and print the Fibonacci series using iterative approach.

    Args:
        n (int): The number of terms in the Fibonacci series to
generate
    """
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1

    print("Fibonacci series (Iterative):")

    # Iterate n times to generate n terms of the Fibonacci series
    for i in range(n):
        # Print the current Fibonacci number
        print(a, end=" ")
        # Update a and b to the next two consecutive Fibonacci numbers
        a, b = b, a + b

    # Print a newline at the end for better formatting
    print()


# ===== RECURSIVE METHOD =====
# Function to generate Fibonacci series recursively
def fibonacci_recursive(n):
    """
    Generate and print the Fibonacci series using recursive approach.

    Args:
        n (int): The number of terms in the Fibonacci series to
generate
    """
    # Base case: if n is 0 or negative, return empty list
    if n <= 0:
        return []

    # Helper function to calculate the nth Fibonacci number
    def fib_helper(num):
        # Base cases for recursion
        if num == 0:
            return 0
        elif num == 1:
            return 1
        # Recursive case: sum of previous two Fibonacci numbers
        else:
            return fib_helper(num - 1) + fib_helper(num - 2)

    print("Fibonacci series (Recursive):")

    # Generate and print n Fibonacci numbers using recursion
    for i in range(n):
        # Call the helper function to get the ith Fibonacci number and
print it
        print(fib_helper(i), end=" ")

    # Print a newline at the end for better formatting
    print()
```

```
# Main program
if __name__ == "__main__":
    # Take the number of terms from the user
    n = int(input("Enter the number of terms: "))
```

```python
    # Display both iterative and recursive approaches
    print()

    # Call the iterative Fibonacci function
    fibonacci_iterative(n)

    print()

    # Call the recursive Fibonacci function
    fibonacci_recursive(n)
```

Output :



```
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 6
6
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> 5
5
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)> & C:/Users/nikit/anaconda3/python.exe "c:/Users/nikit/OneDrive/Desktop/HCL-(Lab)/Modu
lar Design .py"
=== Fibonacci Generator ===
Enter the number of Fibonacci numbers to generate: 5

Fibonacci Sequence:
0 -> 1 -> 1 -> 2 -> 3
PS C:\Users\nikit\OneDrive\Desktop\HCL-(Lab)>
```

Explanation :

Iterative Method Execution Flow

The program starts from the main block and takes input n.

fibonacci_iterative(n) is called.

Variables a = 0 and b = 1 are initialized.

A for loop runs n times:

Prints the current value of a.

Updates values using a, b = b, a + b.

The loop ends after printing n Fibonacci numbers.

Control returns to the main program.

Recursive Method Execution Flow

The program starts from the main block and takes input n.

fibonacci_recursive(n) is called.

A nested helper function fib_helper(num) is defined.

For each value i from 0 to n-1:

  fib_helper(i) is called.

fib_helper() works as follows:

  If num is 0 or 1, return base values.

  Otherwise, recursively calls itself twice:

  fib_helper(n-1) + fib_helper(n-2)

This creates many repeated function calls.

The result is printed and control returns to the main program.

Time and space complexity :

Aspect	**Iterative Approach**	**Recursive Approach**
Time Complexity	O(n)	**O(2^n)** (Exponential)
Space Complexity	O(1)	O(n)
Performance for Large n	Very efficient, fast execution	Very slow and inefficient

When recursion should be avoided :

Recursion **should be avoided** when:

  n is large (e.g., n > 30)

  Performance and speed are important

Memory usage must be minimal

The problem has **overlapping subproblems** without memoization

There is a risk of **stack overflow**