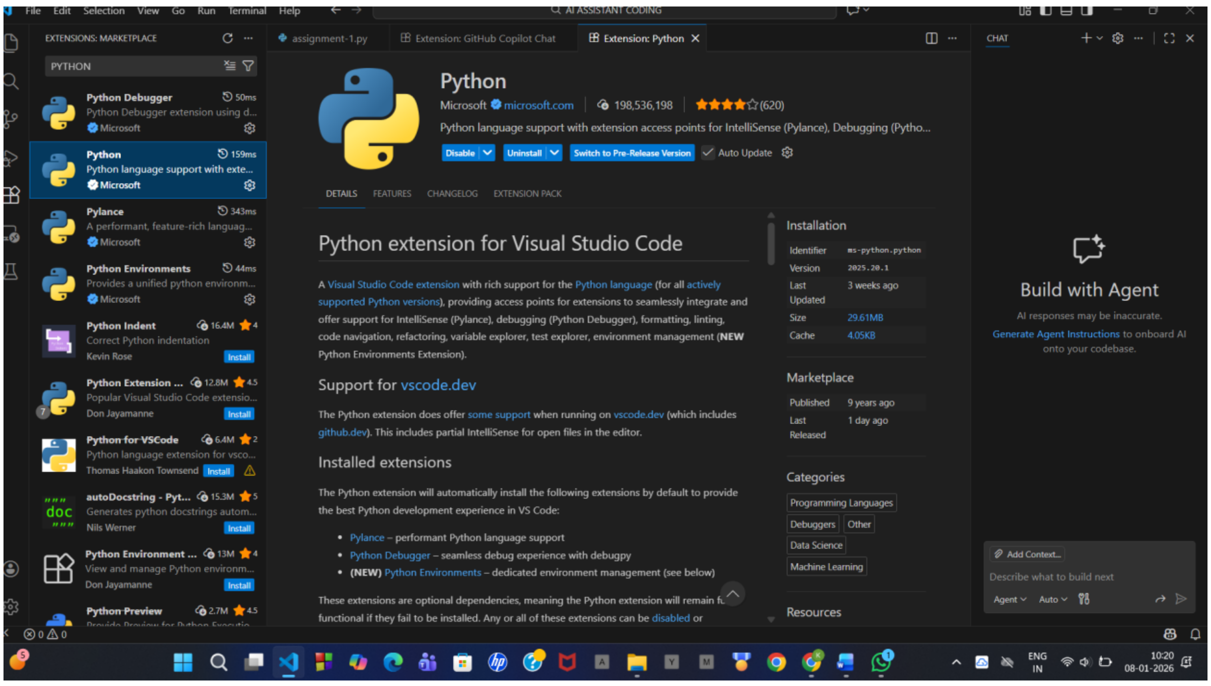
AI-ASSISTED CODING

ASSIGNMENT-1

2303A51242 BATCH-05

TASK-0:



EXTENSIONS: MARKETPLACE

PYTHON

- Python Debugger — Python Debugger extension using d... — Microsoft
- Python — Python language support with exte... — Microsoft
- Pylance — A performant, feature-rich languag... — Microsoft
- Python Environments — Provides a unified python environm... — Microsoft
- Python Indent — Correct Python indentation — Kevin Rose
- Python Extension ... — Popular Visual Studio Code extensio... — Don Jayamanne
- Python for VSCode — Python language extension for vsco... — Thomas Haakon Townsend
- autoDocstring - Pyt... — Generates python docstrings autom... — Nils Werner
- Python Environment ... — View and manage Python environm... — Don Jayamanne
- Python Preview

Python

Microsoft microsoft.com | 198,536,198 | (620)

Python language support with extension access points for IntelliSense (Pylance), Debugging (Pytho...

DETAILS FEATURES CHANGELOG EXTENSION PACK

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 files in the editor.

Installed extensions

The Python extension will automatically install the following extensions by default to provide the best Python development experience in VS Code:

- Pylance – performant Python language support
- Python Debugger – seamless debug experience with debugpy
- **(NEW)** Python Environments – dedicated environment management (see below)

These extensions are optional dependencies, meaning the Python extension will remain fu functional if they fail to be installed. Any or all of these extensions can be disabled or

Installation

Identifier	ms-python.python
Version	2025.20.1
Last Updated	3 weeks ago
Size	29.61MB
Cache	4.05KB

Marketplace

Published	9 years ago
Last Released	1 day ago

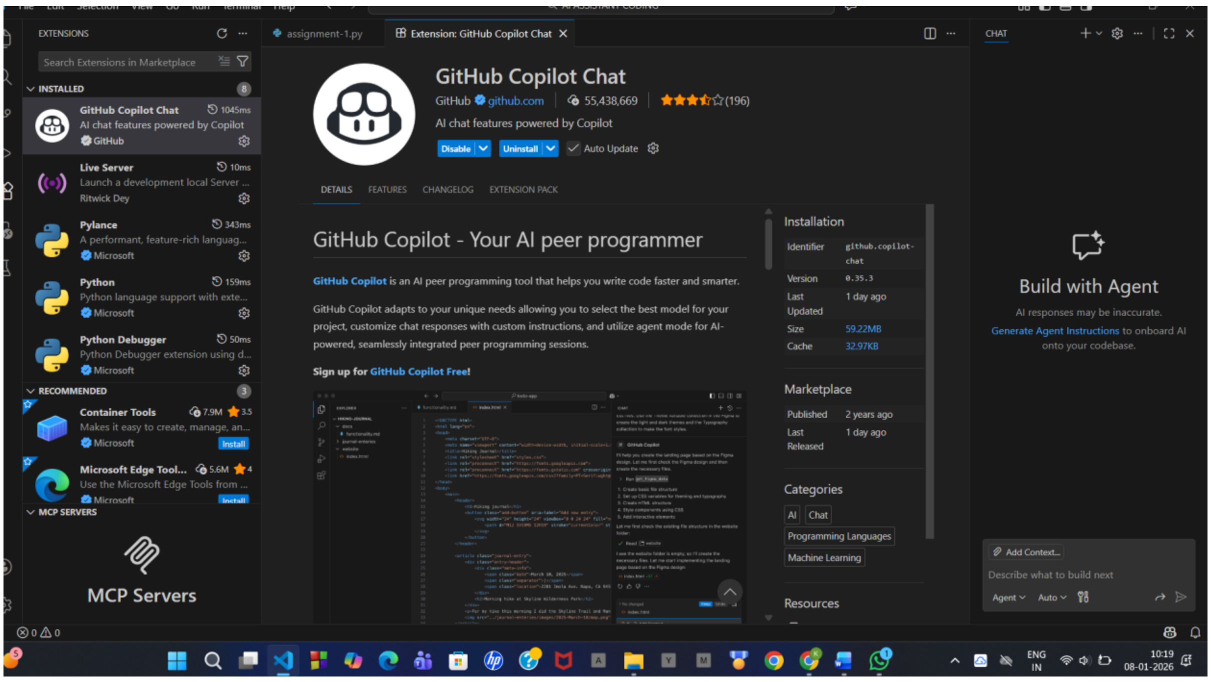
Categories: Programming Languages, Debuggers, Other, Data Science, Machine Learning

Resources

Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.



EXTENSIONS

Search Extensions in Marketplace

INSTALLED

- GitHub Copilot Chat — AI chat features powered by Copilot — GitHub
- Live Server — Launch a development local Server ... — Ritwick Dey
- Pylance — A performant, feature-rich languag... — Microsoft
- Python — Python language support with exte... — Microsoft
- Python Debugger — Python Debugger extension using d... — Microsoft

RECOMMENDED

- Container Tools — Makes it easy to create, manage, an... — Microsoft
- Microsoft Edge Tool... — Use the Microsoft Edge Tools from ... — Microsoft

MCP SERVERS

MCP Servers

GitHub Copilot Chat

GitHub github.com | 55,438,669 | (196)

AI chat features powered by Copilot

DETAILS FEATURES CHANGELOG EXTENSION PACK

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

Categories: AI, Chat, Programming Languages, Machine Learning

Resources

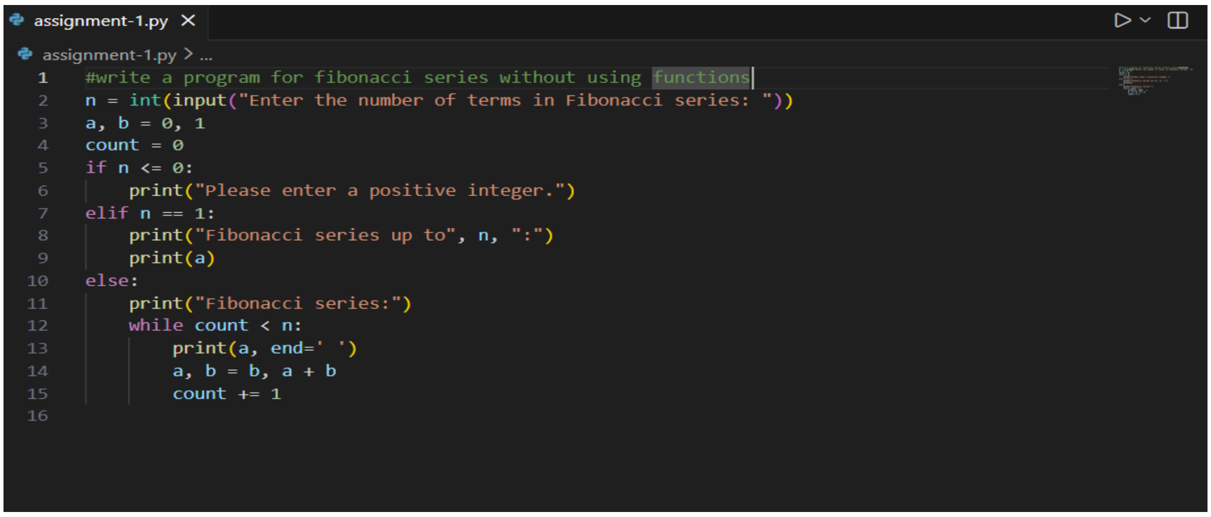
Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

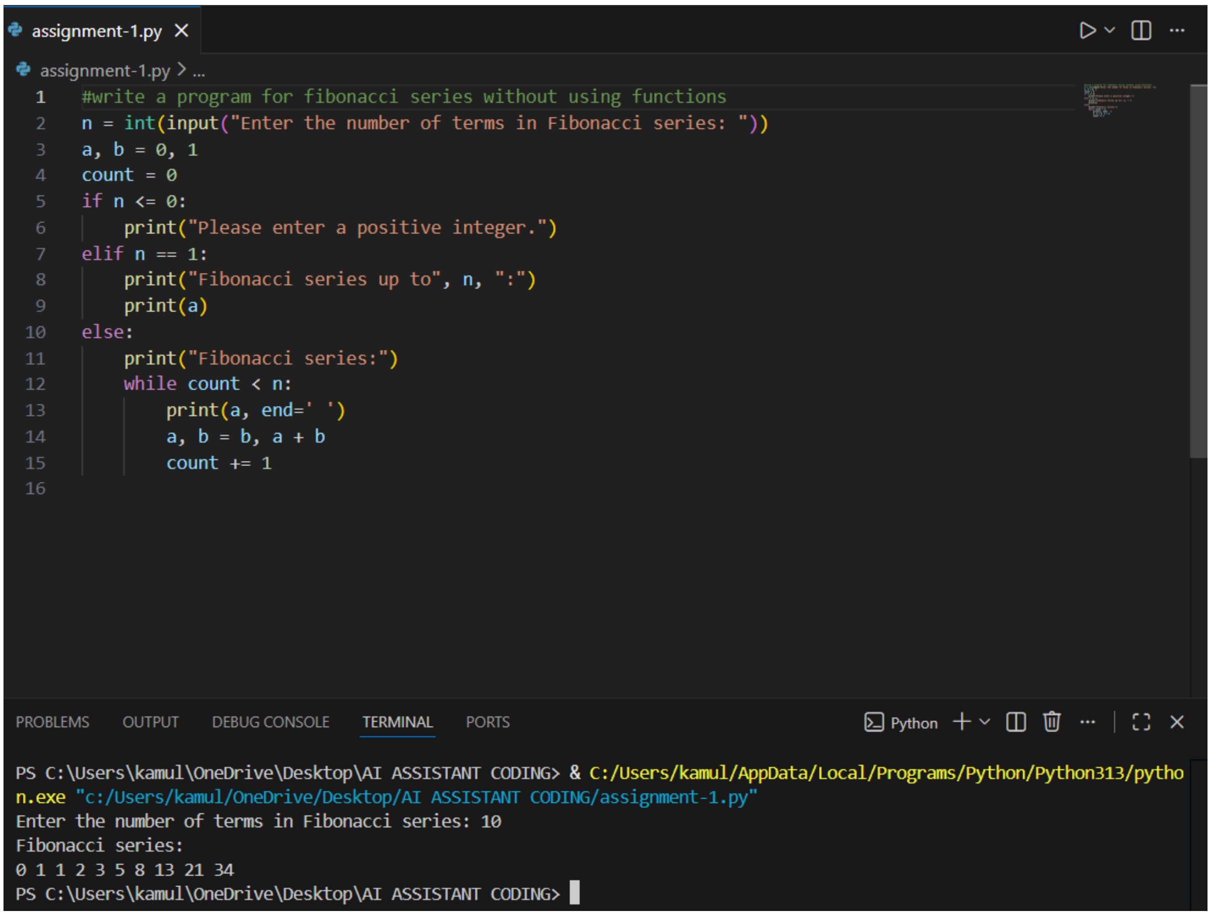
TASK-01:

#write a program for fibonacci series without using functions



assignment-1.py > ...

```python
#write a program for fibonacci series without using functions
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
    print("Please enter a positive integer.")
elif n == 1:
    print("Fibonacci series up to", n, ":")
    print(a)
else:
    print("Fibonacci series:")
    while count < n:
        print(a, end=' ')
        a, b = b, a + b
        count += 1
```



assignment-1.py > ...

```python
#write a program for fibonacci series without using functions
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
    print("Please enter a positive integer.")
elif n == 1:
    print("Fibonacci series up to", n, ":")
    print(a)
else:
    print("Fibonacci series:")
    while count < n:
        print(a, end=' ')
        a, b = b, a + b
        count += 1
```

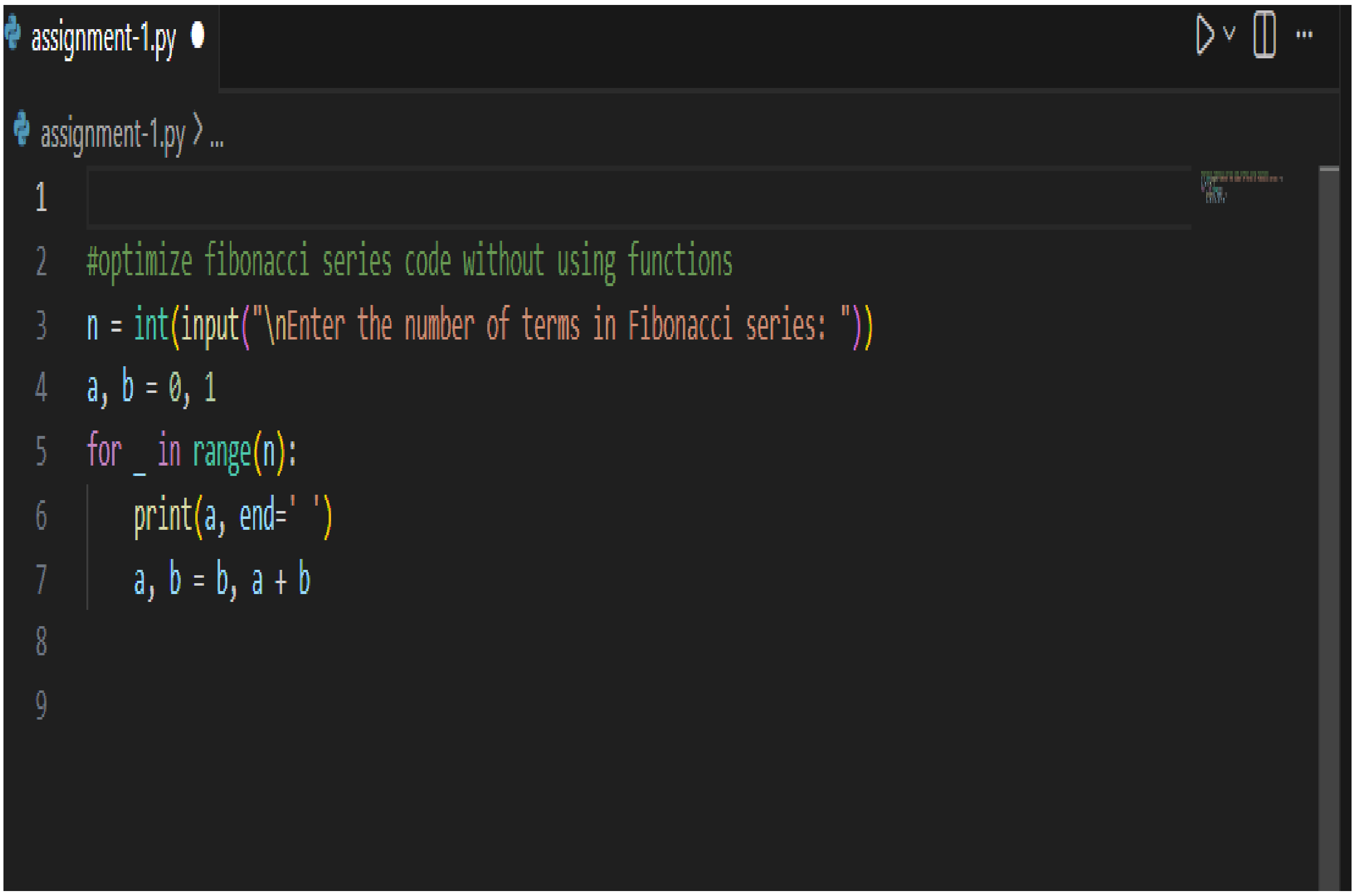
```
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/kamul/AppData/Local/Programs/Python/Python313/pytho
n.exe "c:/Users/kamul/OneDrive/Desktop/AI ASSISTANT CODING/assignment-1.py"
Enter the number of terms in Fibonacci series: 10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING>
```

EXPLANATION:

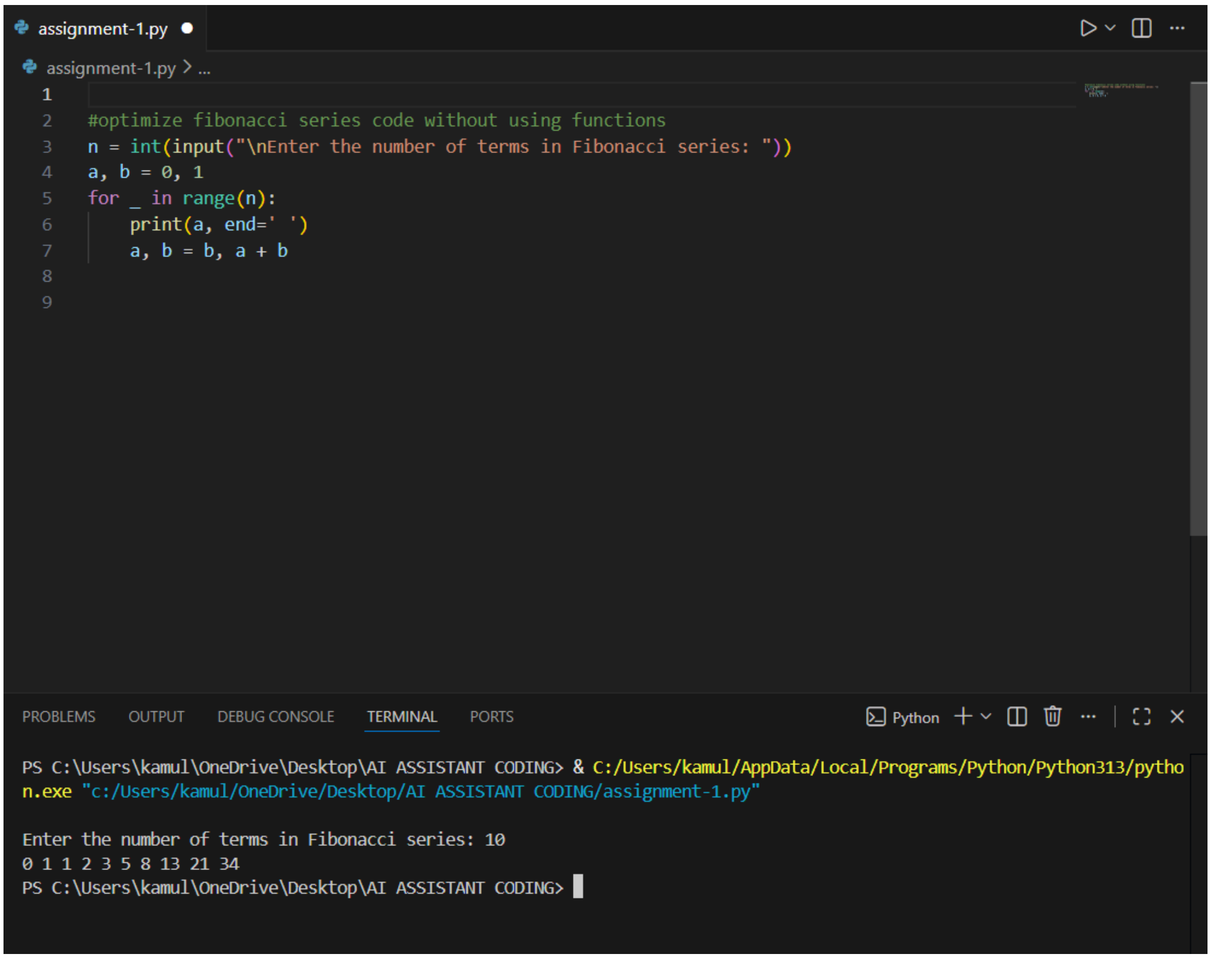
This program generates the Fibonacci series without using any functions by using variables, conditional statements, and a while loop. The user enters the number of terms to be printed. Two variables a and b are initialized with values 0 and 1 to represent the first two Fibonacci numbers. If the user enters a non-positive number, the program displays an error message. If the input is valid, the program uses a while loop to repeatedly print the current Fibonacci number and update the next value by adding the previous two numbers. This approach demonstrates how the Fibonacci series can be generated iteratively using basic programming constructs.

TASK-02:

#optimize fibonacci series code without using functions



```python

#optimize fibonacci series code without using functions
n = int(input("\nEnter the number of terms in Fibonacci series: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b


```



```python

#optimize fibonacci series code without using functions
n = int(input("\nEnter the number of terms in Fibonacci series: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b


```

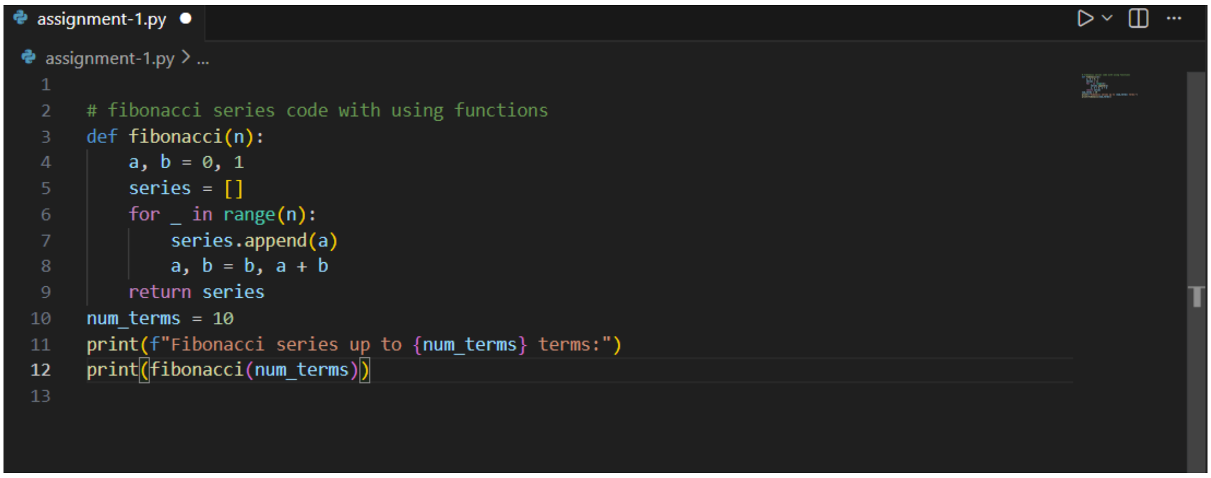
```
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/kamul/AppData/Local/Programs/Python/Python313/pytho
n.exe "c:/Users/kamul/OneDrive/Desktop/AI ASSISTANT CODING/assignment-1.py"

Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING>
```

EXPLANATION:

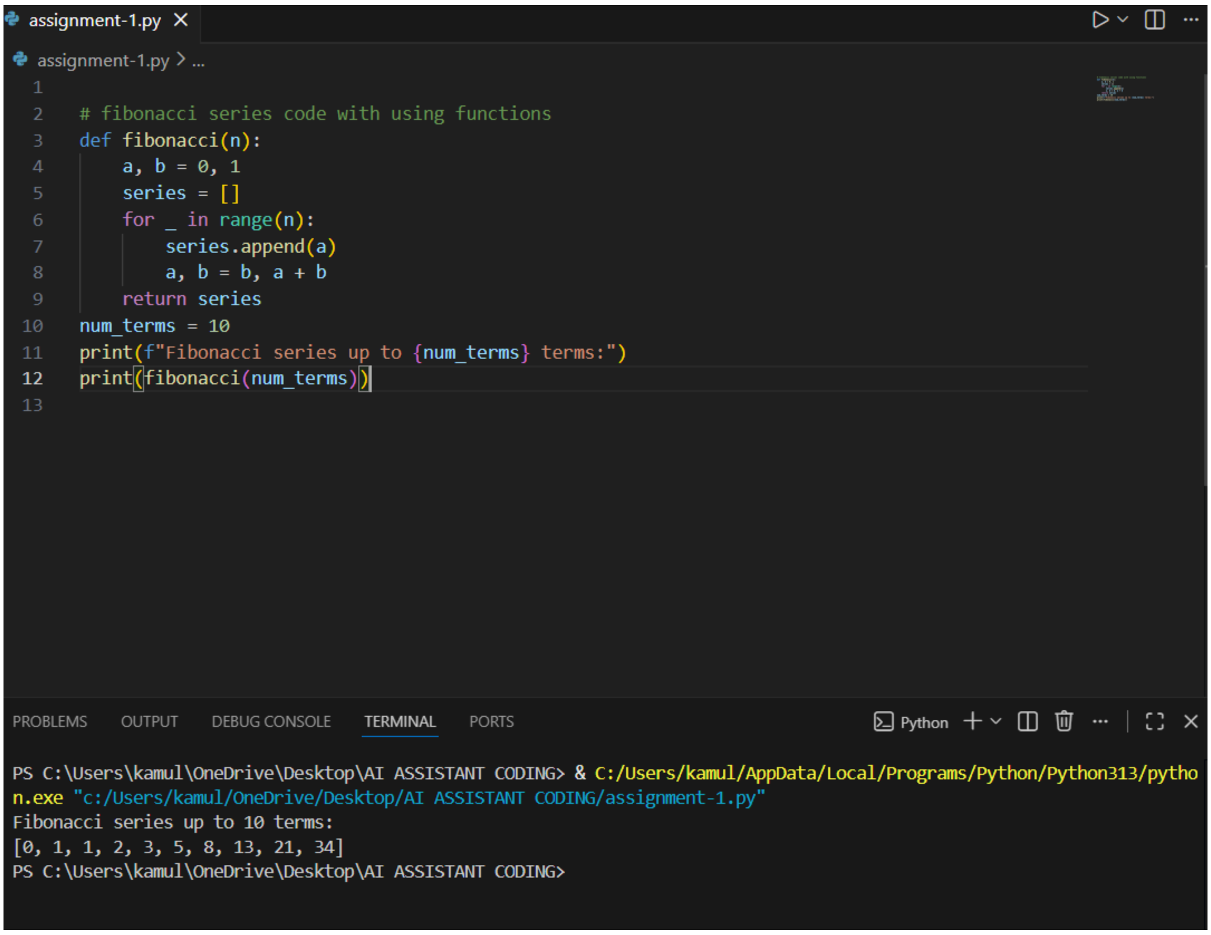
This program prints the Fibonacci series in an optimized and efficient way without using functions. The user enters the number of terms to be displayed. The variables a and b are initialized to 0 and 1, representing the first two Fibonacci numbers. A for loop runs n times, and in each iteration the current value of a is printed. Then, the next Fibonacci number is calculated by updating a and b as the sum of the previous two numbers. This approach avoids unnecessary condition checks and uses minimal variables, making the program simple and efficient.

TASK-03:

fibonacci series code with using functions



```python

# fibonacci series code with using functions
def fibonacci(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = 10
print(f"Fibonacci series up to {num_terms} terms:")
print(fibonacci(num_terms))

```



```python

# fibonacci series code with using functions
def fibonacci(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = 10
print(f"Fibonacci series up to {num_terms} terms:")
print(fibonacci(num_terms))

```

```
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/kamul/AppData/Local/Programs/Python/Python313/pytho
n.exe "c:/Users/kamul/OneDrive/Desktop/AI ASSISTANT CODING/assignment-1.py"
Fibonacci series up to 10 terms:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING>
```

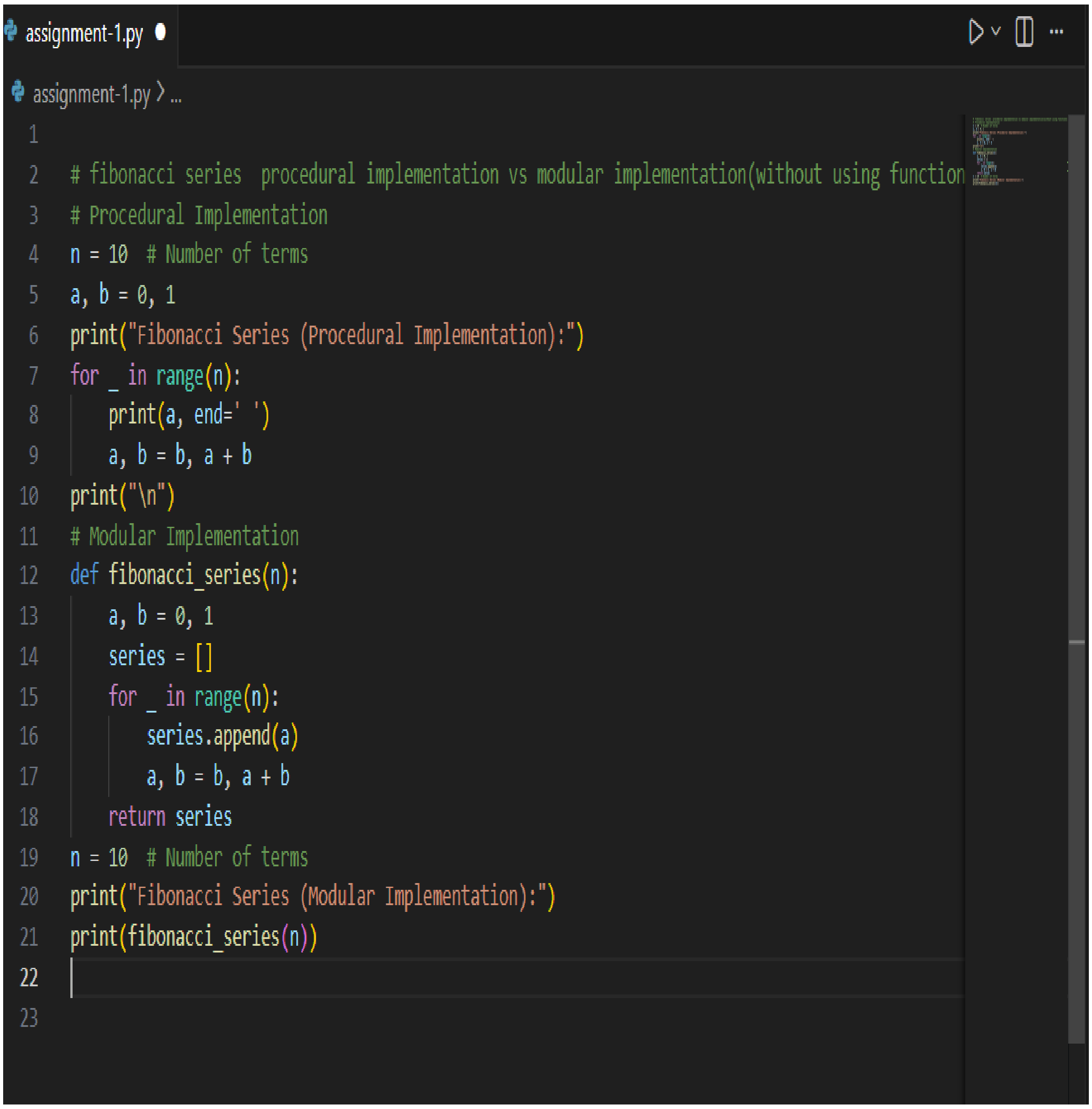
EXPLANATION:

This program generates the Fibonacci series using a function. The function fibonacci(n) initializes the first two Fibonacci numbers as 0 and 1 and uses a for loop to compute the series up to n terms. Each Fibonacci number is stored in a list called series. After calculating all terms, the function returns the list of

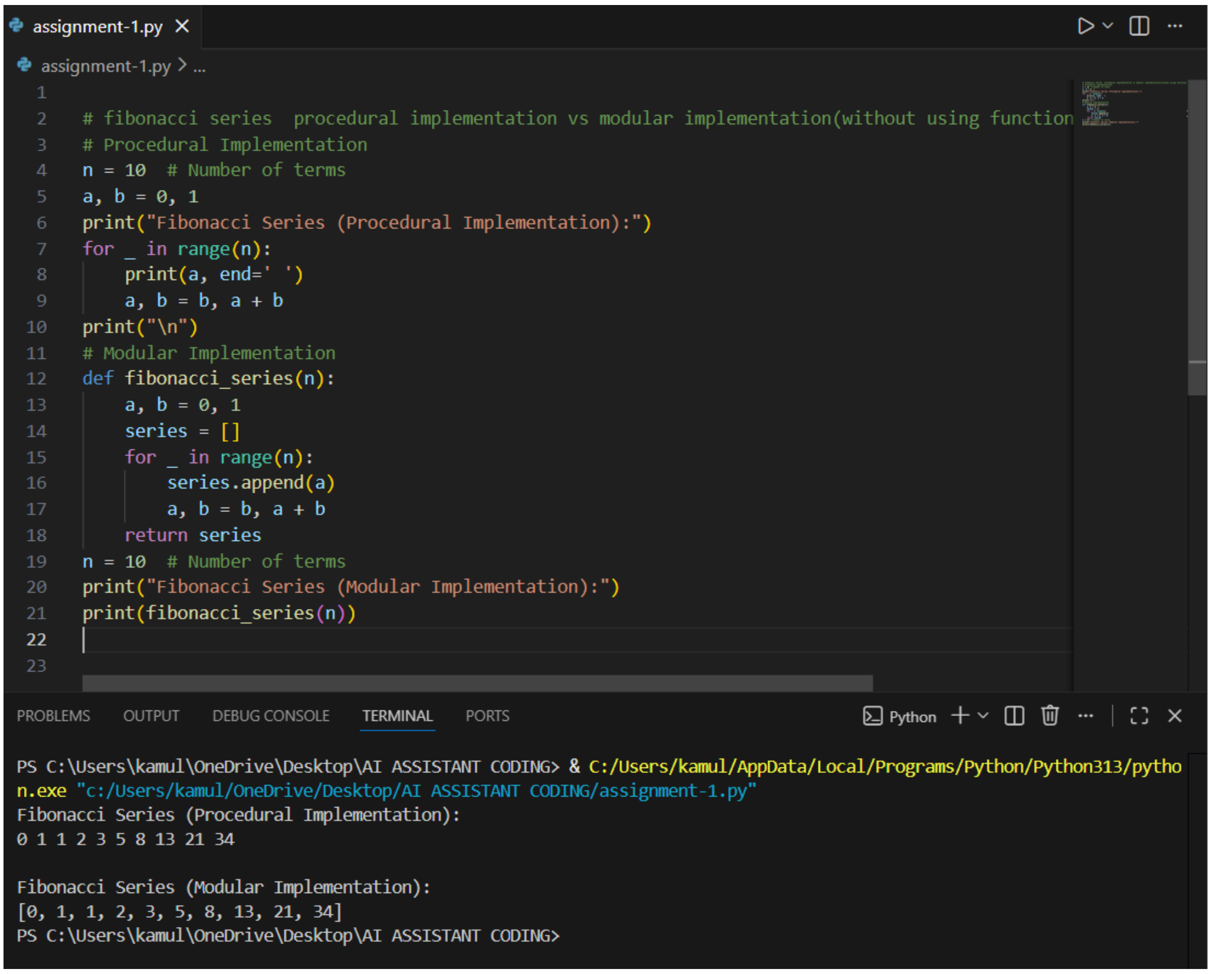
Fibonacci numbers. In the main part of the program, the number of terms is set to 10, and the function is called to display the Fibonacci series. This approach improves code readability, reusability, and organization by using functions.

TASK-04:

# fibonacci series procedural implementation vs modular implementation(without using functions vs with functions)



```python

# fibonacci series  procedural implementation vs modular implementation(without using function
# Procedural Implementation
n = 10  # Number of terms
a, b = 0, 1
print("Fibonacci Series (Procedural Implementation):")
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
print("\n")
# Modular Implementation
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
n = 10  # Number of terms
print("Fibonacci Series (Modular Implementation):")
print(fibonacci_series(n))

```



```python

# fibonacci series  procedural implementation vs modular implementation(without using function
# Procedural Implementation
n = 10  # Number of terms
a, b = 0, 1
print("Fibonacci Series (Procedural Implementation):")
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
print("\n")
# Modular Implementation
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
n = 10  # Number of terms
print("Fibonacci Series (Modular Implementation):")
print(fibonacci_series(n))

```

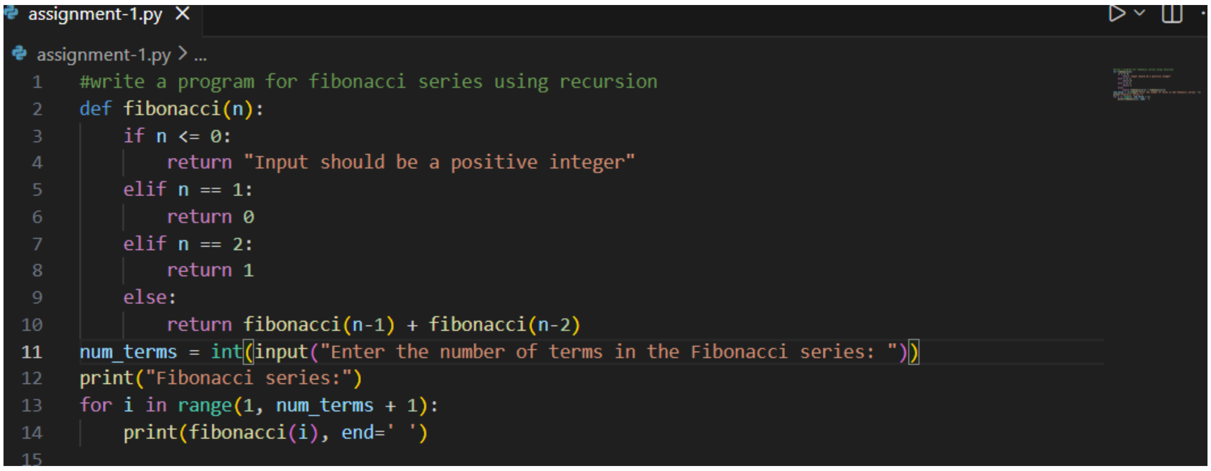
```
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/kamul/AppData/Local/Programs/Python/Python313/pytho
n.exe "c:/Users/kamul/OneDrive/Desktop/AI ASSISTANT CODING/assignment-1.py"
Fibonacci Series (Procedural Implementation):
0 1 1 2 3 5 8 13 21 34

Fibonacci Series (Modular Implementation):
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING>
```

EXPLANATION:

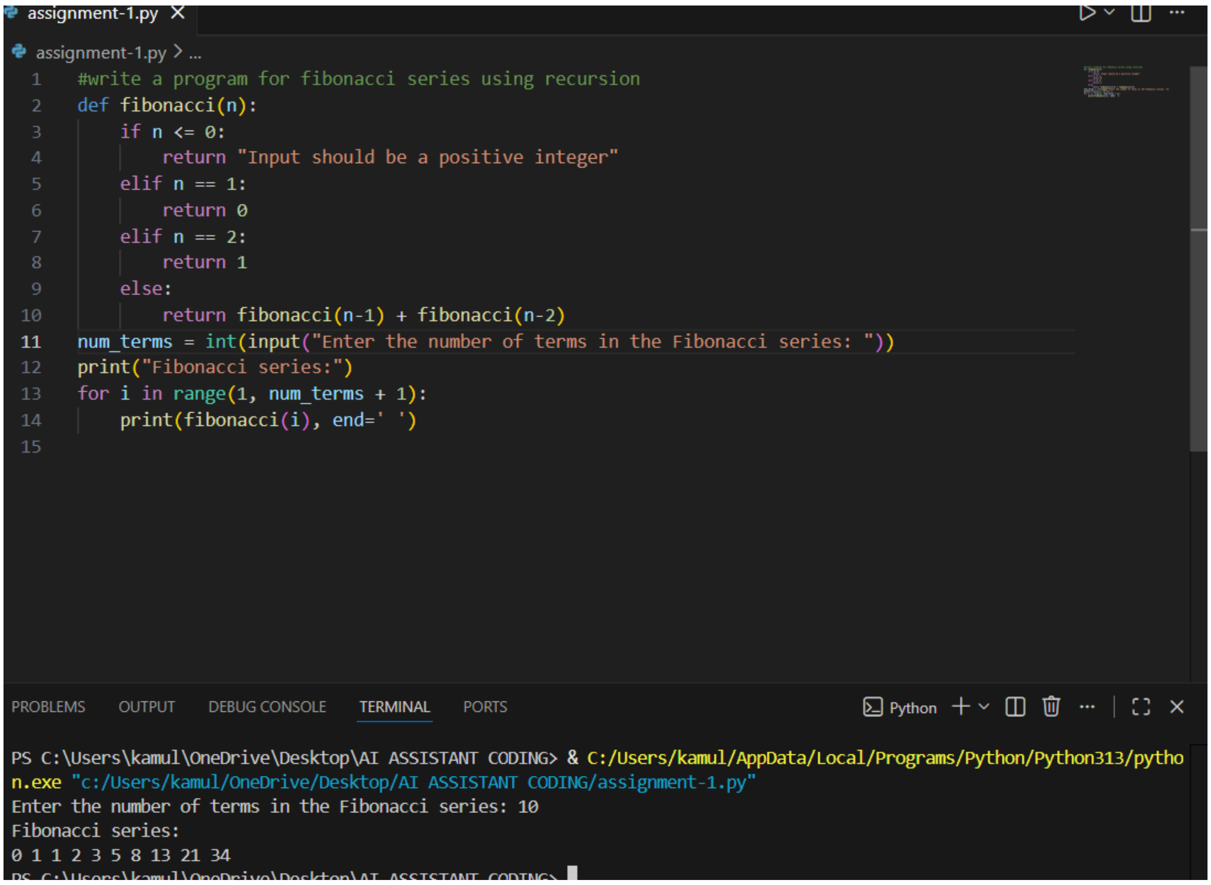
This program demonstrates two different approaches to generating the Fibonacci series: procedural implementation and modular implementation.In the procedural approach, the logic for generating the Fibonacci series is written directly in the main program without using any functions. The variables a and b are initialized to 0 and 1, and a for loop is used to print the Fibonacci numbers one by one. This method is simple but less reusable.In the modular approach, the Fibonacci logic is placed inside a function called fibonacci_series(n). The function computes the Fibonacci numbers using a loop, stores them in a list, and returns the list. The main program then calls this function to display the series. This approach improves code organization, readability, and reusability.Overall, the program highlights the difference between procedural and modular programming by showing how the same problem can be solved in two structured ways.

TASK-05:

#write a program for fibonacci series using recursion



```python
#write a program for fibonacci series using recursion
def fibonacci(n):
    if n <= 0:
        return "Input should be a positive integer"
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
print("Fibonacci series:")
for i in range(1, num_terms + 1):
    print(fibonacci(i), end=' ')
```



```python
#write a program for fibonacci series using recursion
def fibonacci(n):
    if n <= 0:
        return "Input should be a positive integer"
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
print("Fibonacci series:")
for i in range(1, num_terms + 1):
    print(fibonacci(i), end=' ')
```

```
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/kamul/AppData/Local/Programs/Python/Python313/pytho
n.exe "c:/Users/kamul/OneDrive/Desktop/AI ASSISTANT CODING/assignment-1.py"
Enter the number of terms in the Fibonacci series: 10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\kamul\OneDrive\Desktop\AI ASSISTANT CODING>
```

EXPLANATION:

This program generates the Fibonacci series using a recursive function. The function fibonacci(n) returns the *n-th* Fibonacci number. Base conditions are used to return 0 when n = 1 and 1 when n = 2, ensuring the recursion stops. For values greater than 2, the function calls itself to calculate the sum of the previous two Fibonacci numbers. The user enters the number of terms, and a loop prints the Fibonacci series up to that number. This program demonstrates the concept of recursion by solving a problem through repeated function calls.