AIAssistantCoding
Assignment-1

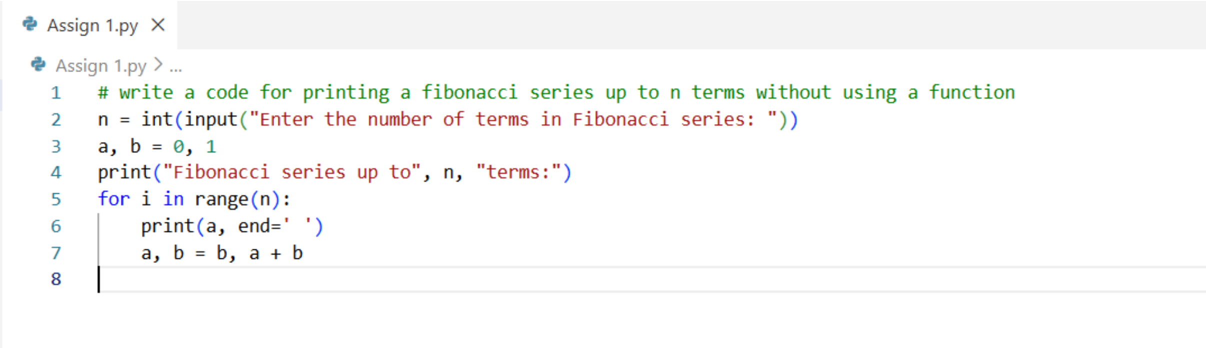
Name of Student : **K . Dinesh** **Batch:42**
Enrollment No. : **2303A52288**

Task1:AI-GeneratedLogicWithoutModularization(FibonacciSequence Without Functions)

UseGitHubCopilottogenerateaPythonprogramthat:
- ➢ **PrintstheFibonaccisequenceupto*n*terms**
- ➢ **Acceptsuserinputfor*n***
- ➢ **Implementsthelogicdirectlyinthemaincode**
- ➢ **Doesnotuseanyuser-definedfunctions**

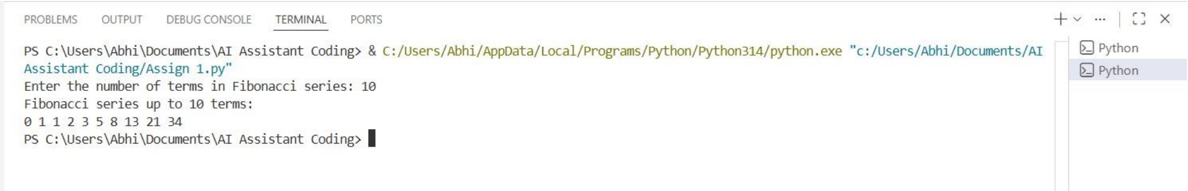
Prompt:
WriteacodeforprintingaFibonacciseriesuptontermswithoutusinga function.

Code:



```python
# write a code for printing a fibonacci series up to n terms without using a function
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for i in range(n):
    print(a, end=' ')
    a, b = b, a + b

```

Output:



```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI
Assistant Coding/Assign 1.py"
Enter the number of terms in Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Abhi\Documents\AI Assistant Coding> 
```

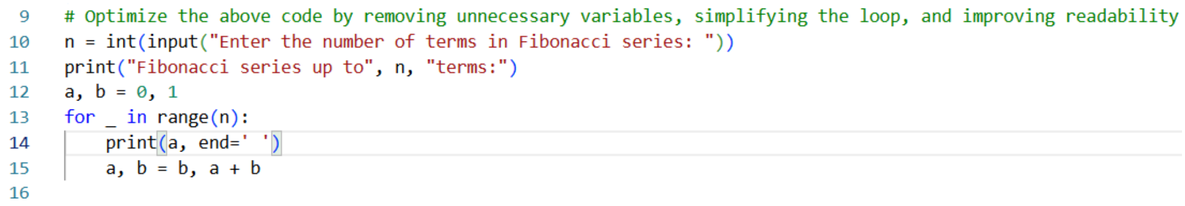
Justification:
The code was produced using an inline prompt in GitHub Copilot within VS Code to implement the logic for generating the Fibonacci sequence up to **n** terms. The program directly accepts user input for the number of terms, uses a **for loop** to iterate up to **n**, and prints the Fibonacci numbers sequentially without defining or using any function.

Task2:AICodeOptimization&Cleanup(ImprovingEfficiency)

- ➢ **ExaminetheCopilot-generatedcodefromTask1andimproveitby:**
- ➢ **Removingredundantvariables**
- ➢ **Simplifyinglooplogic**
- ➢ **Avoidingunnecessarycomputations**
- ➢ **UseCopilotpromptssuchas:**
    - ▪ **“OptimizethisFibonaccicode”**
    - ▪ **“Simplifyvariableusage”**

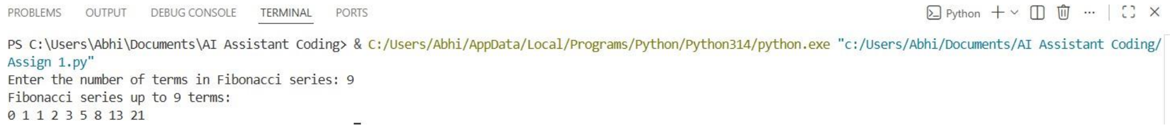
Prompt:
Optimizetheabovecodebyremovingunnecessaryvariables,simplifyingthe loop, and improving readability.

Code:



```python
# Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability
n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", n, "terms:")
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b

```

Output:



```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI Assistant Coding/
Assign 1.py"
Enter the number of terms in Fibonacci series: 9
Fibonacci series up to 9 terms:
0 1 1 2 3 5 8 13 21
```

Justification:
The program accurately produces the Fibonacci sequence using an iterative method that begins with 0 and 1. It calculates exactly the number of terms specified by the user (9), avoiding any additional or omitted values. The output is clearly formatted with space-separated numbers, making it easy to verify results and debug if needed.

Task3:ModularDesignUsingAIAssistance(FibonacciUsingFunctions) Use

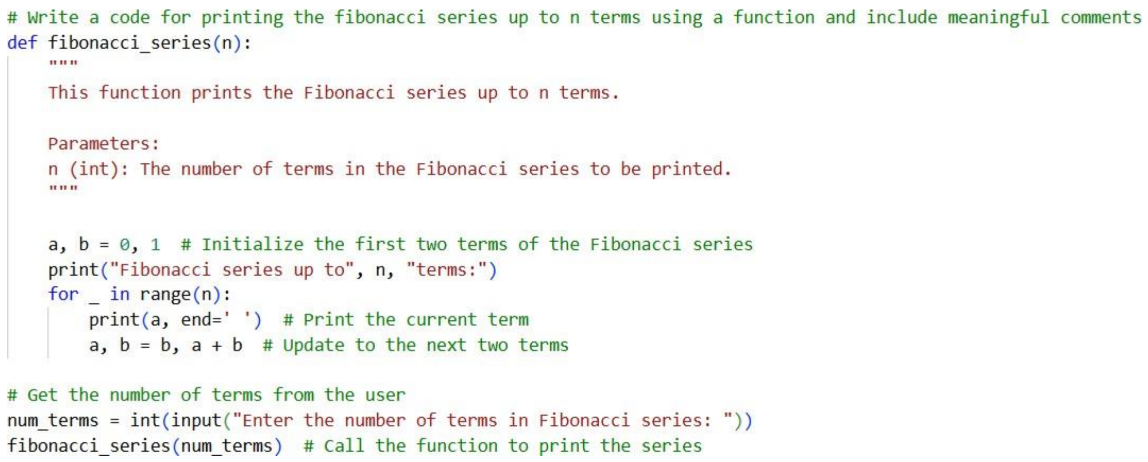
GitHub Copilot to generate a function-based Python program that:

- ➢ **Usesauser-definedfunctiontogenerateFibonaccinumbers**
- ➢ **Returnsorprintsthesequenceupton**
- ➢ **Includesmeaningfulcomments(AI-assisted)**

Prompt:

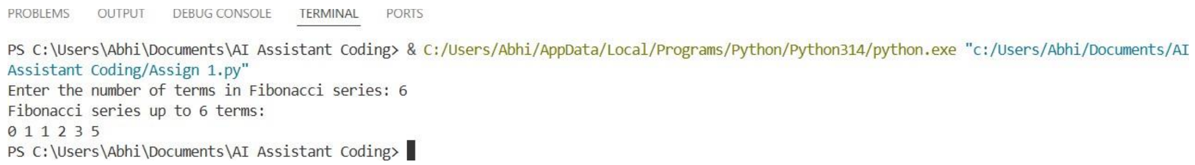
WriteacodeforprintingtheFibonacciseriesuptontermsusingafunctionand include meaningful comments.

Code:



```
# Write a code for printing the fibonacci series up to n terms using a function and include meaningful comments
def fibonacci_series(n):
    """
    This function prints the Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to be printed.
    """

    a, b = 0, 1  # Initialize the first two terms of the Fibonacci series
    print("Fibonacci series up to", n, "terms:")
    for _ in range(n):
        print(a, end=' ')  # Print the current term
        a, b = b, a + b  # Update to the next two terms

# Get the number of terms from the user
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
fibonacci_series(num_terms)  # Call the function to print the series
```

Output:



```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 1.py"
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 terms:
0 1 1 2 3 5
PS C:\Users\Abhi\Documents\AI Assistant Coding> 
```

Justification:

In the above code, the logic for generating the Fibonacci series is implemented using a modular approach, with clear and helpful comments explaining the purpose of each line. The `fibonacci_series` function is designed to be reusable and can be invoked multiple times by the user whenever required.

Task4:ComparativeAnalysis–ProceduralvsModularFibonacciCode

Compare the Copilot-generated Fibonacci programs:

- ➢ **Withoutfunctions(Task1)**
- ➢ **Withfunctions(Task3)**
- ➢ **Analyzethemintermsof:**
  - ▪ **Codeclarity**
  - ▪ **Reusability**
  - ▪ **Debuggingease**
  - ▪ **Suitabilityforlargersystems**

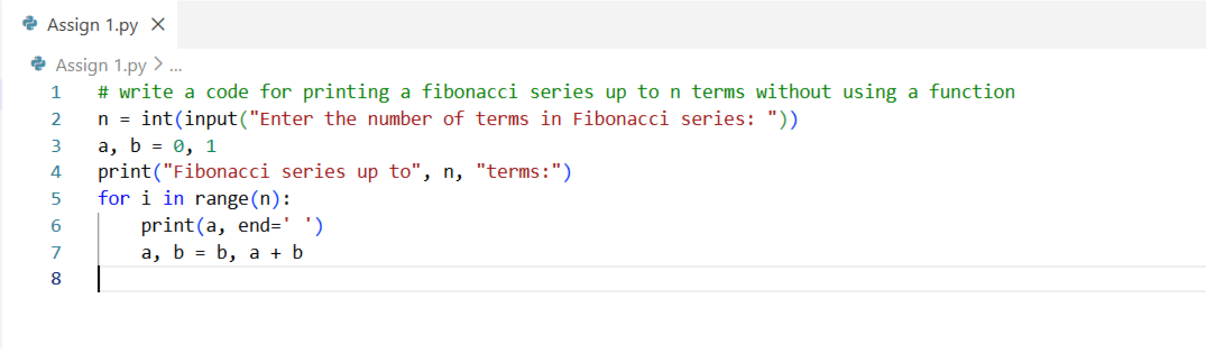
Prompt:

Non-Modular:
WriteacodeforprintingaFibonacciseriesuptontermswithoutusinga function.

Modular:
WriteacodeforprintingtheFibonacciseriesuptontermsusingafunctionand include meaningful comments.

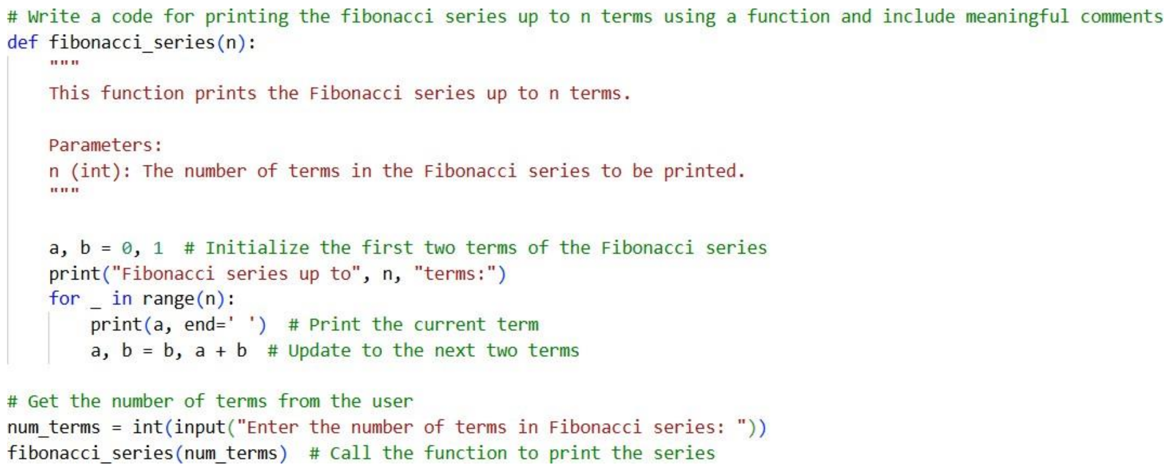
Code:

Procedural:



```python
# write a code for printing a fibonacci series up to n terms without using a function
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for i in range(n):
    print(a, end=' ')
    a, b = b, a + b

```

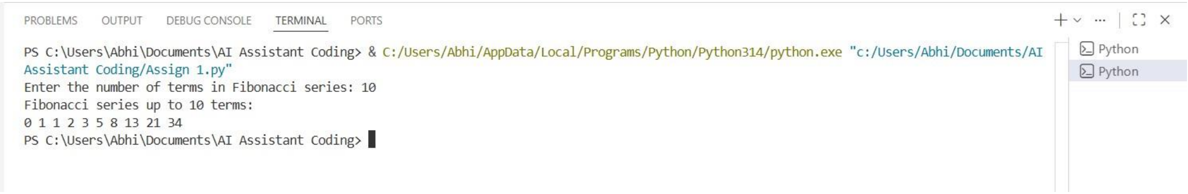
Modular:



```python
# Write a code for printing the fibonacci series up to n terms using a function and include meaningful comments
def fibonacci_series(n):
    """
    This function prints the Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to be printed.
    """

    a, b = 0, 1  # Initialize the first two terms of the Fibonacci series
    print("Fibonacci series up to", n, "terms:")
    for _ in range(n):
        print(a, end=' ')  # Print the current term
        a, b = b, a + b  # Update to the next two terms

# Get the number of terms from the user
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
fibonacci_series(num_terms)  # Call the function to print the series
```

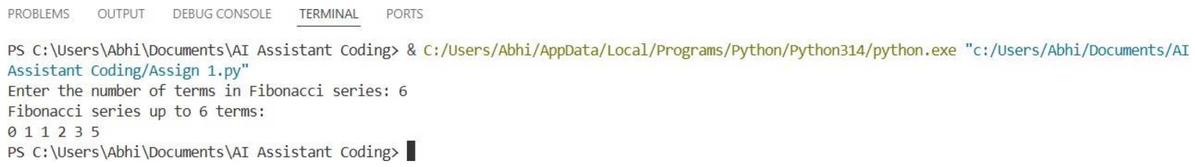
Output:

Procedural:



```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI
Assistant Coding/Assign 1.py"
Enter the number of terms in Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Abhi\Documents\AI Assistant Coding> 
```

Modular:



```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI
Assistant Coding/Assign 1.py"
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 terms:
0 1 1 2 3 5
PS C:\Users\Abhi\Documents\AI Assistant Coding> █
```

Justification:

Procedural code may be straightforward initially, but it becomes increasingly difficult to manage and maintain as the system grows. Modular programming enhances readability by separating functionality into distinct components, making the purpose of the code clearer. It also promotes reusability, allowing modules to be easily integrated into larger projects or workflows. Additionally, isolating logic in modules simplifies debugging and troubleshooting.

Task5:AI-GeneratedIterativevsRecursiveFibonacciApproaches (Different Algorithmic Approaches for Fibonacci Series)

PromptGitHubCopilottogenerate:

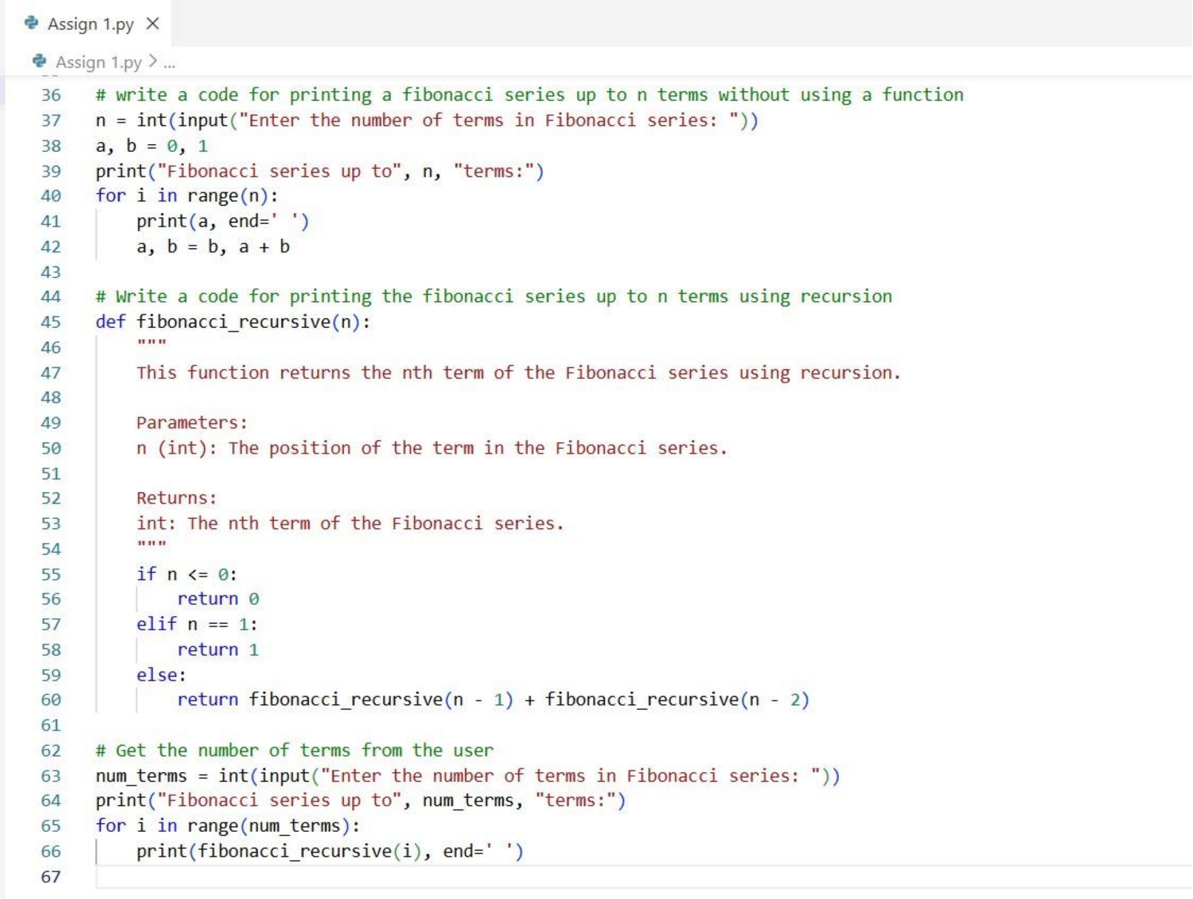
- **AniterativeFibonacciimplementation**
- **ArecursiveFibonacciimplementation**

Prompt:

WriteacodeforprintingaFibonacciseriesuptontermswithoutusinga function.

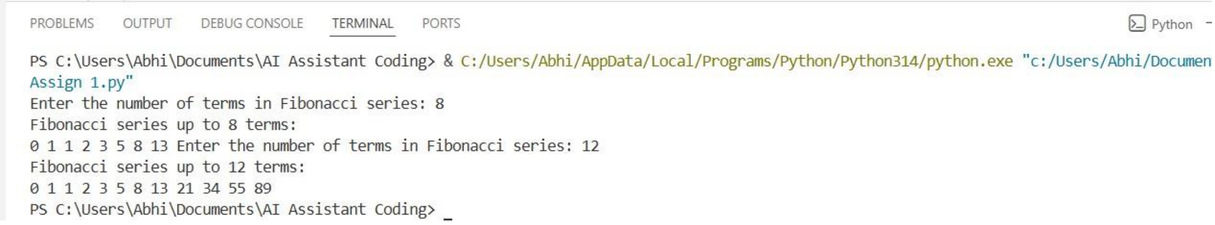
WriteacodeforprintingtheFibonacciseriesuptontermsusingrecursion.

Code:



```python
# write a code for printing a fibonacci series up to n terms without using a function
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for i in range(n):
    print(a, end=' ')
    a, b = b, a + b

# Write a code for printing the fibonacci series up to n terms using recursion
def fibonacci_recursive(n):
    """
    This function returns the nth term of the Fibonacci series using recursion.

    Parameters:
    n (int): The position of the term in the Fibonacci series.

    Returns:
    int: The nth term of the Fibonacci series.
    """
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# Get the number of terms from the user
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", num_terms, "terms:")
for i in range(num_terms):
    print(fibonacci_recursive(i), end=' ')
```

Output:



```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documen
Assign 1.py"
Enter the number of terms in Fibonacci series: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13 Enter the number of terms in Fibonacci series: 12
Fibonacci series up to 12 terms:
0 1 1 2 3 5 8 13 21 34 55 89
PS C:\Users\Abhi\Documents\AI Assistant Coding> _
```

Justification:

Because the iterative method runs efficiently and consumes a constant amount of memory, it performs well even for large values of *n*. Although the recursive approach is conceptually elegant, it suffers from exponential time complexity unless optimization techniques like memoization are applied. For large inputs, recursion can also lead to stack overflow and poor performance. Therefore, iteration is the more practical and reliable option for scalable, high-performance systems.