

AI Assisted Coding

ANNAPU VASANTHA SHOBA RANI

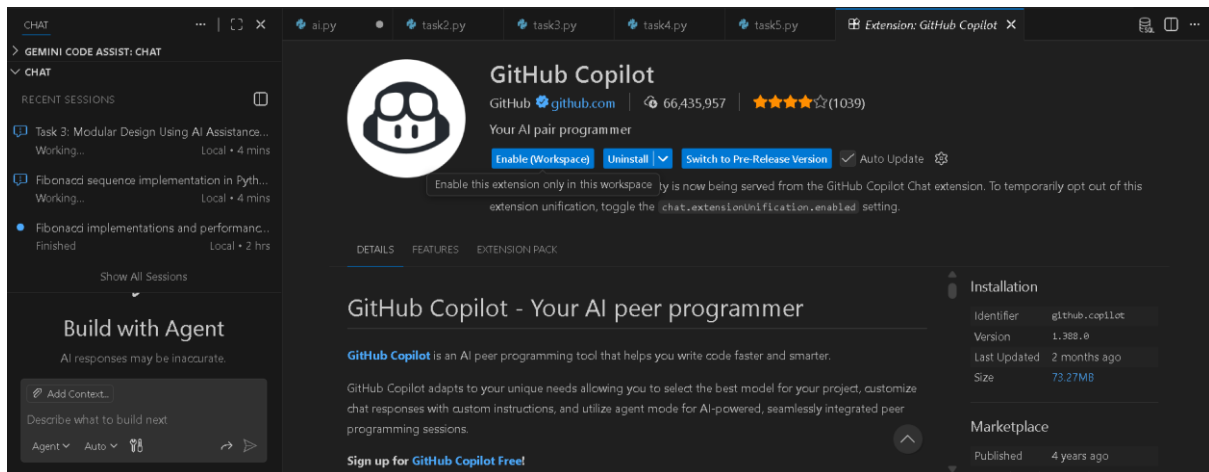
2303A51395

BATCH - 06

Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow

Lab Objectives:

Week1 -



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

PROMPT:

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

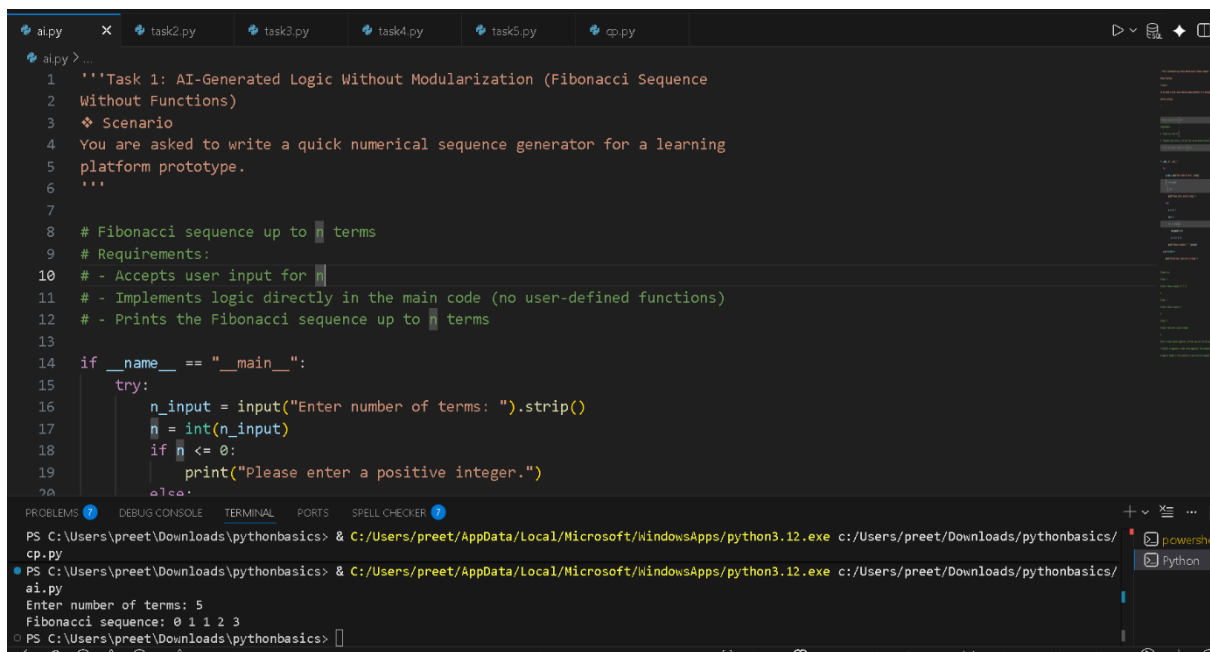
❖ Scenario

You are asked to write a quick numerical sequence generator for a learning
platform prototype.

❖ Task Description

Use GitHub Copilot to generate a Python program that:

- Prints the Fibonacci sequence up to n terms
- Accepts user input for n
- Implements the logic directly in the main code
- Does not use any user-defined functions
- ❖ Expected Output
- Correct Fibonacci sequence for given n
- Screenshot(s) showing Copilot-generated suggestions
- Sample inputs and outputs



```
1 '''Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence
2 Without Functions)
3 ❖ Scenario
4 You are asked to write a quick numerical sequence generator for a learning
5 platform prototype.
6 '''
7
8 # Fibonacci sequence up to n terms
9 # Requirements:
10 # - Accepts user input for n
11 # - Implements logic directly in the main code (no user-defined functions)
12 # - Prints the Fibonacci sequence up to n terms
13
14 if __name__ == "__main__":
15     try:
16         n_input = input("Enter number of terms: ").strip()
17         n = int(n_input)
18         if n <= 0:
19             print("Please enter a positive integer.")
20     except:
```

PS C:\Users\preet\Downloads\pythonbasics> & C:/Users/preet/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/preet/Downloads/pythonbasics/cp.py

PS C:\Users\preet\Downloads\pythonbasics> & C:/Users/preet/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/preet/Downloads/pythonbasics/ai.py

Enter number of terms: 5

Fibonacci sequence: 0 1 1 2 3

OUTPUT: Enter number of terms: 5

Fibonacci sequence: 0 1 1 2 3

Code Explanation (Fibonacci without functions)

- input() → takes number of terms from user
- int() → converts input to integer
- if n <= 0 → checks for invalid (non-positive) input

- `a, b = 0, 1` → starting Fibonacci numbers
- `for _ in range(n)` → runs loop n times
- `seq.append(str(a))` → stores current Fibonacci number
- `a, b = b, a + b` → updates values for next term
- `" ".join(seq)` → prints numbers in one line
- `try / except` → handles invalid (non-integer) input safely

Output: Prints Fibonacci sequence up to n terms.

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

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

❖ Scenario

The prototype will be shared with other developers and needs optimization.

❖ Task Description

- Examine the Copilot-generated code from Task 1 and improve it by:
 - Removing redundant variables
 - Simplifying loop logic
 - Avoiding unnecessary computations
 - Use Copilot prompts such as:
 - “Optimize this Fibonacci code”
 - “Simplify variable usage”

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

❖ Expected Output

- Original vs improved code
- Written explanation of:
 - What was inefficient

- How the optimized version improves performance and readability

```

1 """Task 2 - Optimization & Cleanup: Fibonacci
2
3 This file contains:
4 - The original Copilot-style implementation (as 'original_fib_list') - builds string list inside loop.
5 - The improved implementation (as 'optimized_fib_generator') - yields integers and separates generation from presentation.
6 - A small benchmark that compares time and peak memory for the two approaches.
7
8 Use: python task2.py          # runs a small demo
9 | python task2.py --bench     # runs a simple benchmark (default n=10000)
10 """
11 from __future__ import annotations
12
13 import time
14 import tracemalloc
15 import sys
16 from typing import Generator, Iterable, List
17
18
19 # ----- Original (Copilot-generated) -----
20 # Inefficient generator

```

PROBLEMS | DEBUG CONSOLE | TERMINAL | PORTS | SPELL CHECKER

PS C:\Users\preet\Downloads\pythonbasics> & C:/Users/preet/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/preet/Downloads/pythonbasics/task2.py

Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34

Optimized output:

Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34

PS C:\Users\preet\Downloads\pythonbasics>

OUTPUT:

Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34

Optimized output:

Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34

CODE EXPLANATION

Optimization & Cleanup (Fibonacci)

Original version (original_fib_list)

- Generates Fibonacci numbers
- Converts each number to string inside the loop
- Stores all values in a list
- ➡ **More memory usage**, mixed logic (generation + formatting)

Optimized version (optimized_fib_generator)

- Uses a **generator (yield)**
- Produces numbers one-by-one
- Conversion to string happens only when printing
- ➡ **Less memory**, cleaner design, reusable

Demo

- Prints Fibonacci using both methods
- Output is the same

Benchmark

- Measures **time** and **memory**
- Optimized version:
 - Uses less peak memory
 - Scales better for large n

Key takeaway

- Generator = better performance + cleaner code
- Separate **logic (generation)** from **presentation (printing)**

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

PROMPT:

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

❖ Scenario

The Fibonacci logic is now required in multiple modules of an application.

❖ Task Description

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

- Uses a user-defined function to generate Fibonacci numbers
- Returns or prints the sequence up to n
- Includes meaningful comments (AI-assisted)

❖ Expected Output

- Correct function-based Fibonacci implementation
- Screenshots documenting Copilot's function generation
- Sample test cases with outputs

Code Explanation:

Optimization & Cleanup (Fibonacci)

Original version (original_fib_list)

- Generates Fibonacci numbers
- Converts each number to string inside the loop
- Stores all values in a list
 - **More memory usage**, mixed logic (generation + formatting)

Optimized version (optimized_fib_generator)

- Uses a **generator (yield)**
- Produces numbers one-by-one
- Conversion to string happens only when printing
 - **Less memory**, cleaner design, reusable

Demo

- Prints Fibonacci using both methods
- Output is the same

Benchmark

- Measures **time** and **memory**
- Optimized version:
 - Uses less peak memory
 - Scales better for large n

Key takeaway

- Generator = better performance + cleaner code
- Separate **logic (generation)** from **presentation (printing)**

The screenshot shows a VS Code editor with a file named `task3.py` open. The code is a Python module for generating Fibonacci numbers. It includes a docstring, imports for `__future__` and `typing`, and a function `generate_fibonacci` that takes `n` and `up_to_value` as arguments. A comment mentions GitHub Copilot assistance. The terminal at the bottom shows the execution of the script, displaying the first 7 Fibonacci numbers (count=7) as `0, 1, 1, 2, 3, 5, 8`, the first 8 Fibonacci numbers up to value 20 (`up_to_value=True`) as `0, 1, 1, 2, 3, 5, 8, 13`, and edge cases for `n=0` and `n=1`.

```
task3.py > ...
1 """task3.py
2
3 Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)
4
5 This module provides a function-based implementation to generate Fibonacci numbers.
6 It supports returning the first N Fibonacci numbers (by count) or the sequence up to
7 a maximum value.
8
9 AI note: comments and docstrings were enhanced with GitHub Copilot assistance. "docstrings": Unknown word.
10 """
11 from __future__ import annotations
12 from typing import List
13
14
15 def generate_fibonacci(n: int, *, up_to_value: bool = False) -> List[int]:
16     """Generate Fibonacci numbers up to n or up to a value if up_to_value is True.
17     Returns the first n Fibonacci numbers or the sequence up to a maximum value if up_to_value is True.
18     """
19     if n < 0:
20         return []
21     if n == 0:
22         return [0]
23     if n == 1:
24         return [0, 1]
25     fibs = [0, 1]
26     for i in range(2, n):
27         fibs.append(fibs[i-1] + fibs[i-2])
28     return fibs
29
30 if __name__ == "__main__":
31     # Example: first 7 Fibonacci numbers (count=7)
32     print(generate_fibonacci(7))
33
34     # Example: Fibonacci numbers up to value 20 (up_to_value=True)
35     print(generate_fibonacci(20, up_to_value=True))
36
37     # Example: edge cases
38     print(generate_fibonacci(0))
39     print(generate_fibonacci(1))
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

PS C:\Users\preet\Downloads\pythonbasics> & C:/Users/preet/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/preet/Downloads/pythonbasics/task3.py

Example: first 7 Fibonacci numbers (count=7)

0, 1, 1, 2, 3, 5, 8

Example: Fibonacci numbers up to value 20 (up_to_value=True)

0, 1, 1, 2, 3, 5, 8, 13

Example: edge cases

n=0 -> []

n=1 -> [0]

OUTPUT:

Example: first 7 Fibonacci numbers (count=7)

0, 1, 1, 2, 3, 5, 8

Example: Fibonacci numbers up to value 20 (up_to_value=True)

0, 1, 1, 2, 3, 5, 8, 13

Example: edge cases

n=0 -> []

n=1 -> [0]

All simple tests passed.

Notes: To document Copilot's function generation, open this file in VS Code, enable GitHub Copilot, trigger an inline suggestion for `generate_fibonacci`, and take a screenshot of the suggestion popup or accepted code for your report.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

PROMPT:

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

❖ Scenario

You are participating in a code review session.

❖ Task Description

Compare the Copilot-generated Fibonacci programs:

➤ Without functions (Task 1)

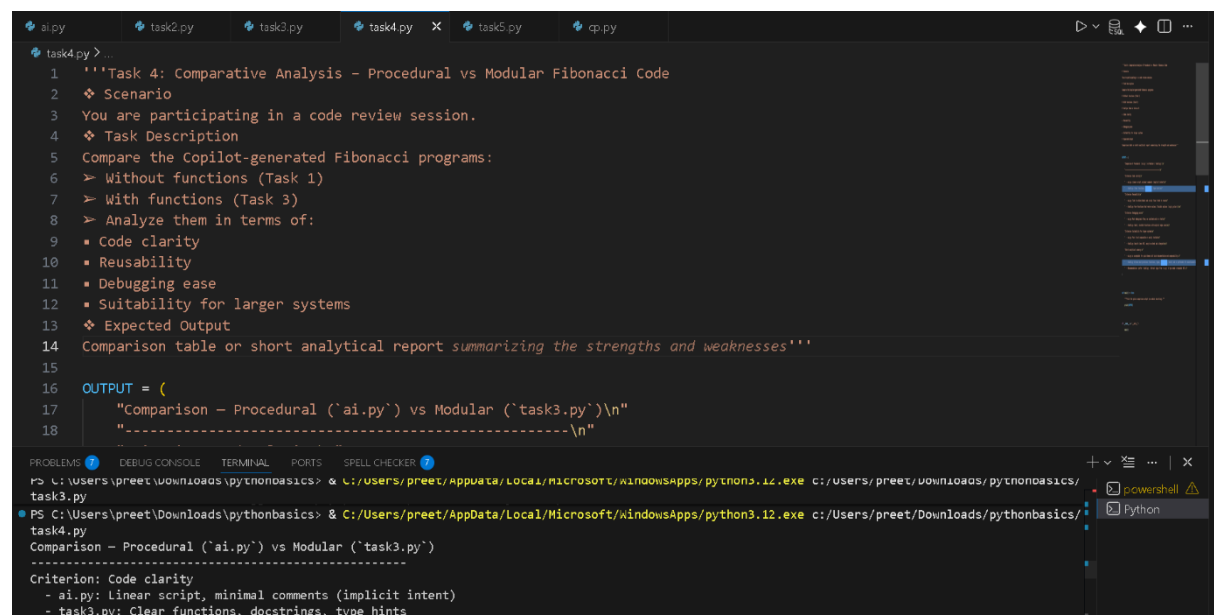
➤ With functions (Task 3)

➤ Analyze them in terms of:

- Code clarity
- Reusability
- Debugging ease
- Suitability for larger systems

❖ Expected Output

Comparison table or short analytical report



```
1  """Task 4: Comparative Analysis - Procedural vs Modular Fibonacci Code
2  ❖ Scenario
3  You are participating in a code review session.
4  ❖ Task Description
5  Compare the Copilot-generated Fibonacci programs:
6  ➤ Without functions (Task 1)
7  ➤ With functions (Task 3)
8  ➤ Analyze them in terms of:
9  ▪ Code clarity
10 ▪ Reusability
11 ▪ Debugging ease
12 ▪ Suitability for larger systems
13 ❖ Expected Output
14 Comparison table or short analytical report summarizing the strengths and weaknesses'''
15
16 OUTPUT = (
17     "Comparison - Procedural (`ai.py`) vs Modular (`task3.py`)\\n"
18     "-----\\n"
19
20     "Criterion: Code clarity\\n"
21     "- ai.py: Linear script, minimal comments (implicit intent)\\n"
22     "- task3.py: Clear functions, docstrings, type hints\\n"
23 )
```

OUTPUT:

Comparison — Procedural (`ai.py`) vs Modular (`task3.py`)

Criterion: Code clarity

- ai.py: Linear script, minimal comments (implicit intent)

- task3.py: Clear functions, docstrings, type hints

Criterion: Reusability

- ai.py: Tied to stdin/stdout and script flow — hard to reuse
- task3.py: Pure functions that return values, flexible options (`up_to_value`)

Criterion: Debugging ease

- ai.py: Must debug main flow; no isolated units or tests
- task3.py: Small, testable functions with explicit edge cases

Criterion: Suitability for larger systems

- ai.py: Poor — not composable or easily testable
- task3.py: Good — clear API, easy to extend and integrate

Short analytical summary:

- ai.py is acceptable for quick demos but lacks documentation and composability.
- task3.py follows best practices (functions, types, docstrings, tests) and is preferable for maintainability and production.
- Recommendation: prefer `task3.py`. Extract logic from `ai.py` if you need a reusable API.

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

PROMPT :

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

❖ Scenario

Your mentor wants to assess AI's understanding of different algorithmic

paradigms.

❖ Task Description

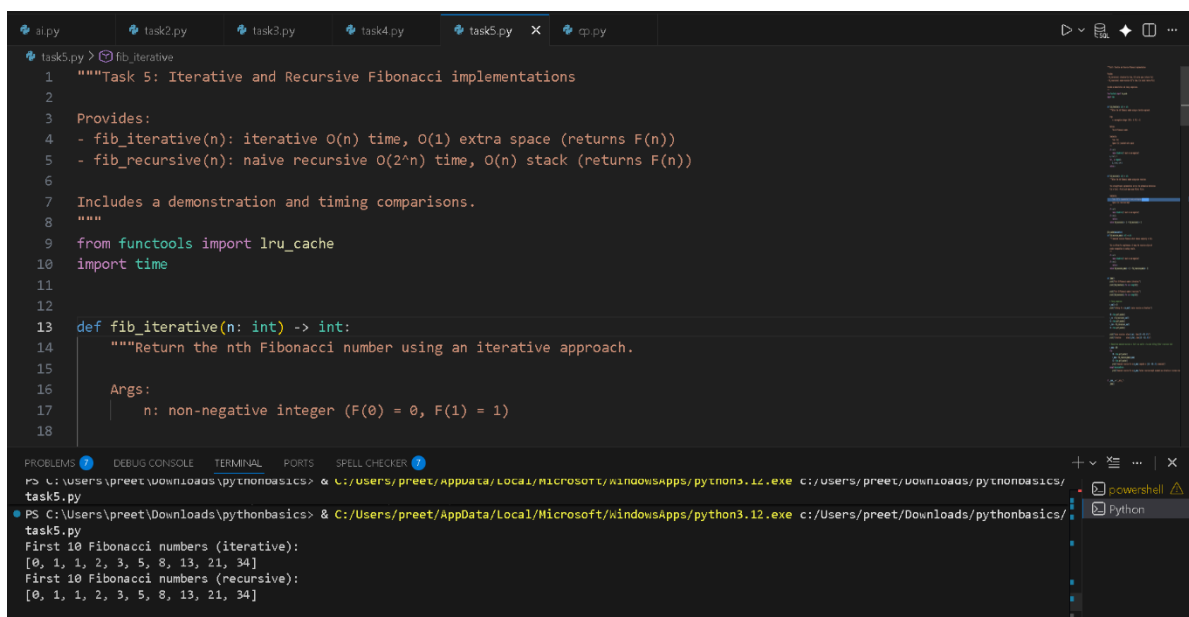
Prompt GitHub Copilot to generate:

An iterative Fibonacci implementation

A recursive Fibonacci implementation

❖ Expected Output

- Two correct implementations
- Explanation of execution flow for both
- Comparison covering:
 - Time and space complexity
 - Performance for large n
 - When recursion should be avoided



The screenshot shows a VS Code editor with a file named `task5.py` open. The code defines two functions: `fib_iterative` and `fib_recursive`. The `fib_iterative` function uses a loop to calculate the nth Fibonacci number with O(n) time and O(1) space complexity. The `fib_recursive` function uses a naive recursive approach with O(2^n) time and O(n) stack space complexity. The code also includes a demonstration and timing comparisons. The terminal output shows the first 10 Fibonacci numbers for both implementations, which are identical: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34].

```
1 """Task 5: Iterative and Recursive Fibonacci implementations
2
3 Provides:
4 - fib_iterative(n): iterative O(n) time, O(1) extra space (returns F(n))
5 - fib_recursive(n): naive recursive O(2^n) time, O(n) stack (returns F(n))
6
7 Includes a demonstration and timing comparisons.
8 """
9 from functools import lru_cache
10 import time
11
12
13 def fib_iterative(n: int) -> int:
14     """Return the nth Fibonacci number using an iterative approach.
15
16     Args:
17         n: non-negative integer (F(0) = 0, F(1) = 1)
18
19     """
20     if n < 0:
21         raise ValueError("n must be a non-negative integer")
22     if n == 0:
23         return 0
24     if n == 1:
25         return 1
26     a, b = 0, 1
27     for _ in range(2, n + 1):
28         a, b = b, a + b
29     return b
30
31 def fib_recursive(n: int) -> int:
32     """Return the nth Fibonacci number using a recursive approach.
33
34     Args:
35         n: non-negative integer (F(0) = 0, F(1) = 1)
36
37     """
38     if n < 0:
39         raise ValueError("n must be a non-negative integer")
40     if n == 0:
41         return 0
42     if n == 1:
43         return 1
44     return fib_recursive(n - 1) + fib_recursive(n - 2)
45
46 if __name__ == "__main__":
47     n = 10
48     print(f"First {n} Fibonacci numbers (iterative):")
49     print(fib_iterative_list(n))
50     print(f"First {n} Fibonacci numbers (recursive):")
51     print(fib_recursive_list(n))
52
53 def fib_iterative_list(n: int) -> list[int]:
54     """Return the first n Fibonacci numbers using an iterative approach.
55
56     Args:
57         n: non-negative integer
58
59     """
60     if n < 0:
61         raise ValueError("n must be a non-negative integer")
62     fibs = [0] * n
63     if n > 0:
64         fibs[0] = 0
65     if n > 1:
66         fibs[1] = 1
67     for i in range(2, n):
68         fibs[i] = fibs[i - 1] + fibs[i - 2]
69     return fibs
70
71 def fib_recursive_list(n: int) -> list[int]:
72     """Return the first n Fibonacci numbers using a recursive approach.
73
74     Args:
75         n: non-negative integer
76
77     """
78     if n < 0:
79         raise ValueError("n must be a non-negative integer")
80     fibs = [0] * n
81     if n > 0:
82         fibs[0] = 0
83     if n > 1:
84         fibs[1] = 1
85     for i in range(2, n):
86         fibs[i] = fib_recursive(i)
87     return fibs
```

Terminal Output:

```
PS C:\Users\preet\Downloads\pythonbasics> & C:/Users/preet/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/preet/Downloads/pythonbasics/task5.py
First 10 Fibonacci numbers (iterative):
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
First 10 Fibonacci numbers (recursive):
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

OUTPUT:

First 10 Fibonacci numbers (iterative):

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

First 10 Fibonacci numbers (recursive):

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

Timing for n=30 (naive recursive vs iterative):

naive recursive: value=832040, time=0.185535s

iterative: value=832040, time=0.000009s

memoized recursive for n=500 computed in 0.001015s (memoized)

CODE EXPLANATION

Task 5: Iterative vs Recursive Fibonacci

fib_iterative(n)

- Uses a loop to calculate Fibonacci
- Updates values using $a, b = b, a + b$
- Time: $O(n)$
- Space: $O(1)$
 - ✓ Fast and memory-efficient

fib_recursive(n)

- Follows the formula $F(n) = F(n-1) + F(n-2)$
- Repeats many calculations
- Time: $O(2^n)$
- Space: $O(n)$ (call stack)
 - ✗ Very slow for large n

fib_recursive_memo(n)

- Uses @lru_cache to store previous results
- Avoids repeated work
- Time: $O(n)$
- Space: $O(n)$
 - ✓ Fast but still uses recursion

Demo & Timing

- Prints first 10 Fibonacci numbers
- Compares execution time of iterative vs recursive
- Shows memoized recursion is much faster than naive recursion

Conclusion

- Iterative → best for performance

- Naive recursive → only for learning
- Memoized recursive → good balance, but iterative is safest for large n