SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 2

Program	**: B. Tech (CSE)**
Specialization	**: -**
Course Title	**: AI Assisted Coding**
Course Code	**: 23CS002PC304**
Semester	**: II**
Academic Session	**: 2025-2026**
Name of Student	**: I.Sathwik Rajeshwara Chary**
Enrollment No.	**: 2403A51L03**
Batch No.	**: 51**
Date	**: 09/01/26**

Submission Starts here

Screenshots:

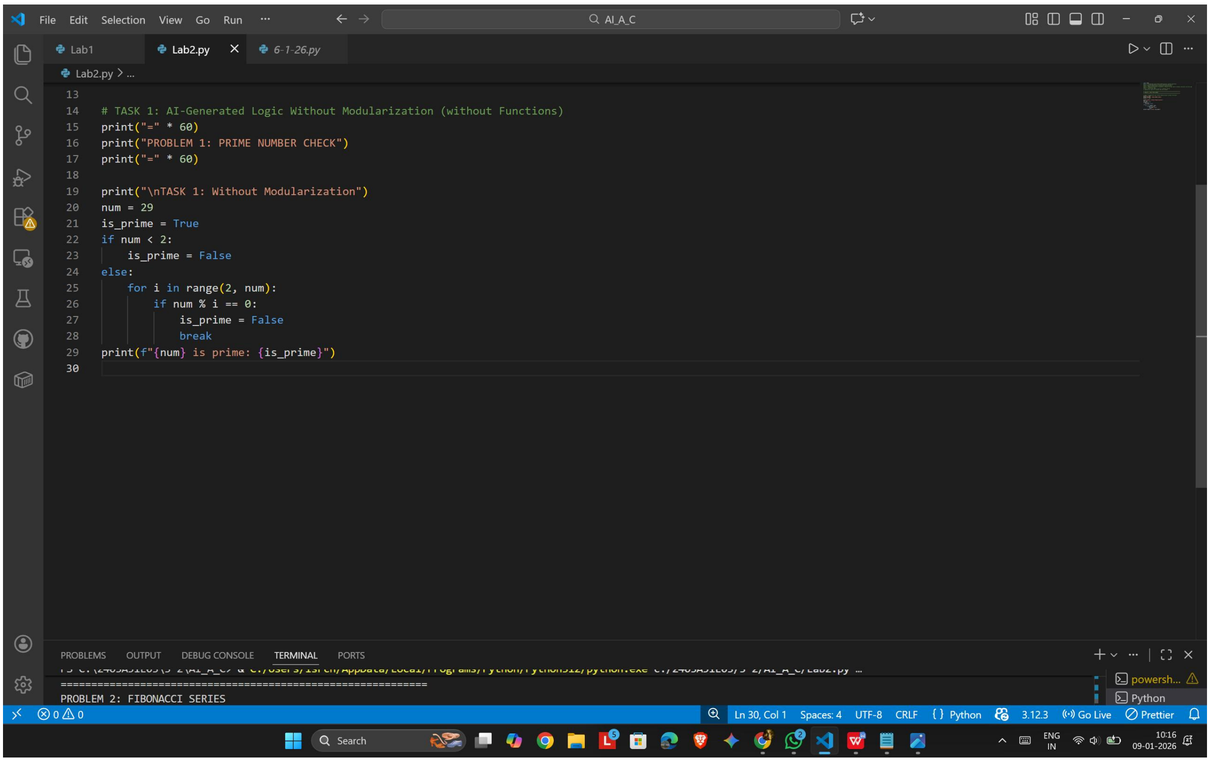
Problem 1- Check for Prime

TASK-1:

Prompt :

# TASK 1: AI-Generated Logic Without Modularization (Check for Prime without using Functions)

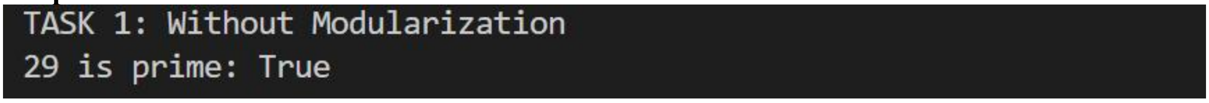
Code:



```python

# TASK 1: AI-Generated Logic Without Modularization (without Functions)
print("=" * 60)
print("PROBLEM 1: PRIME NUMBER CHECK")
print("=" * 60)

print("\nTASK 1: Without Modularization")
num = 29
is_prime = True
if num < 2:
    is_prime = False
else:
    for i in range(2, num):
        if num % i == 0:
            is_prime = False
            break
print(f"{num} is prime: {is_prime}")

```

```
============================================================
PROBLEM 2: FIBONACCI SERIES
```

Output:



```
TASK 1: Without Modularization
29 is prime: True
```

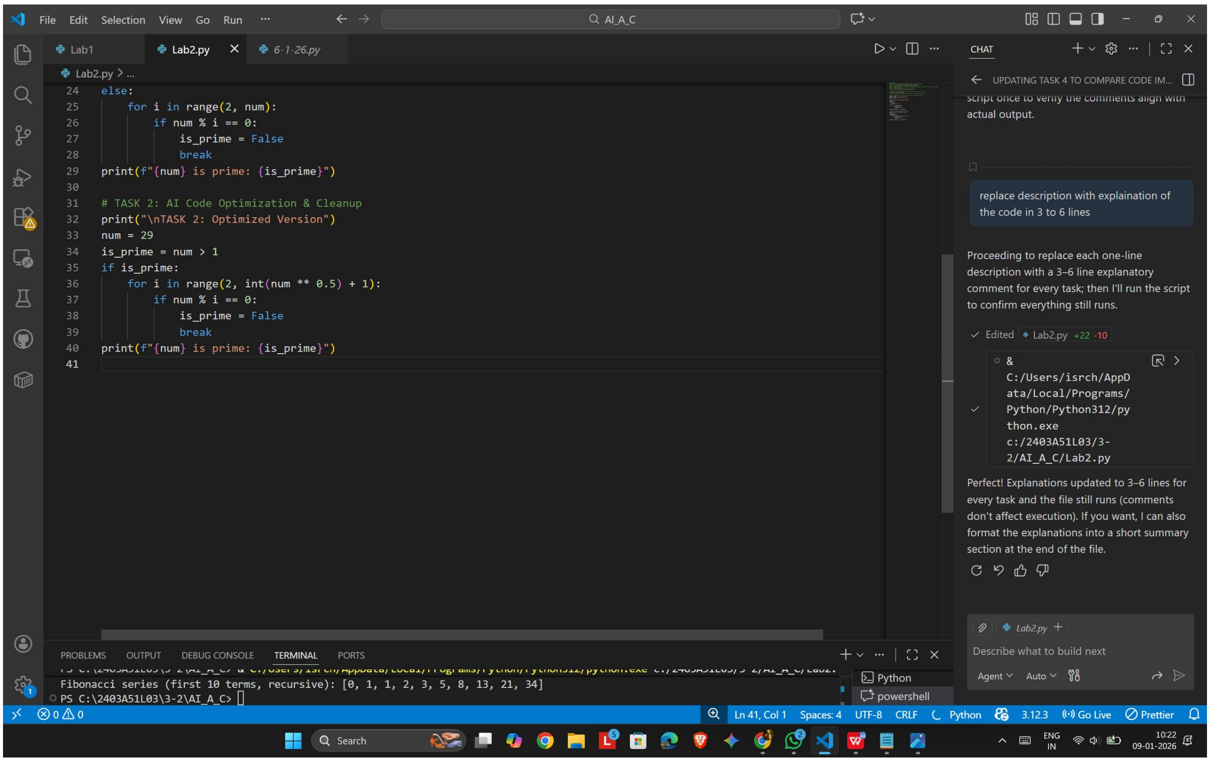
Explanation:

This procedural prime check loops from 2 to n-1 testing divisibility and breaks on the first divisor.
It uses a boolean flag to track primality and does not encapsulate logic in a function.
For 29 no divisors are found, so the printed result is True.

TASK-2:

Prompt:
# TASK 2: AI Code Optimization & Cleanup

Code:



```python
else:
    for i in range(2, num):
        if num % i == 0:
            is_prime = False
            break
print(f"{num} is prime: {is_prime}")

# TASK 2: AI Code Optimization & Cleanup
print("\nTASK 2: Optimized Version")
num = 29
is_prime = num > 1
if is_prime:
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break
print(f"{num} is prime: {is_prime}")
```

```
Fibonacci series (first 10 terms, recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\2403A51L03\3-2\AI_A_C>
```

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

✓ Edited Lab2.py +22 -10

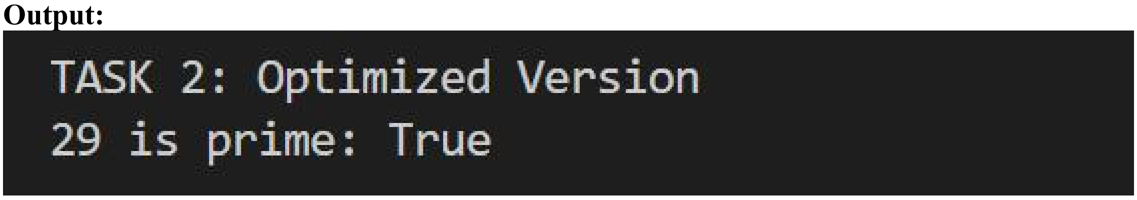
```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Describe what to build next

Agent Auto

Output:



```
TASK 2: Optimized Version
29 is prime: True
```

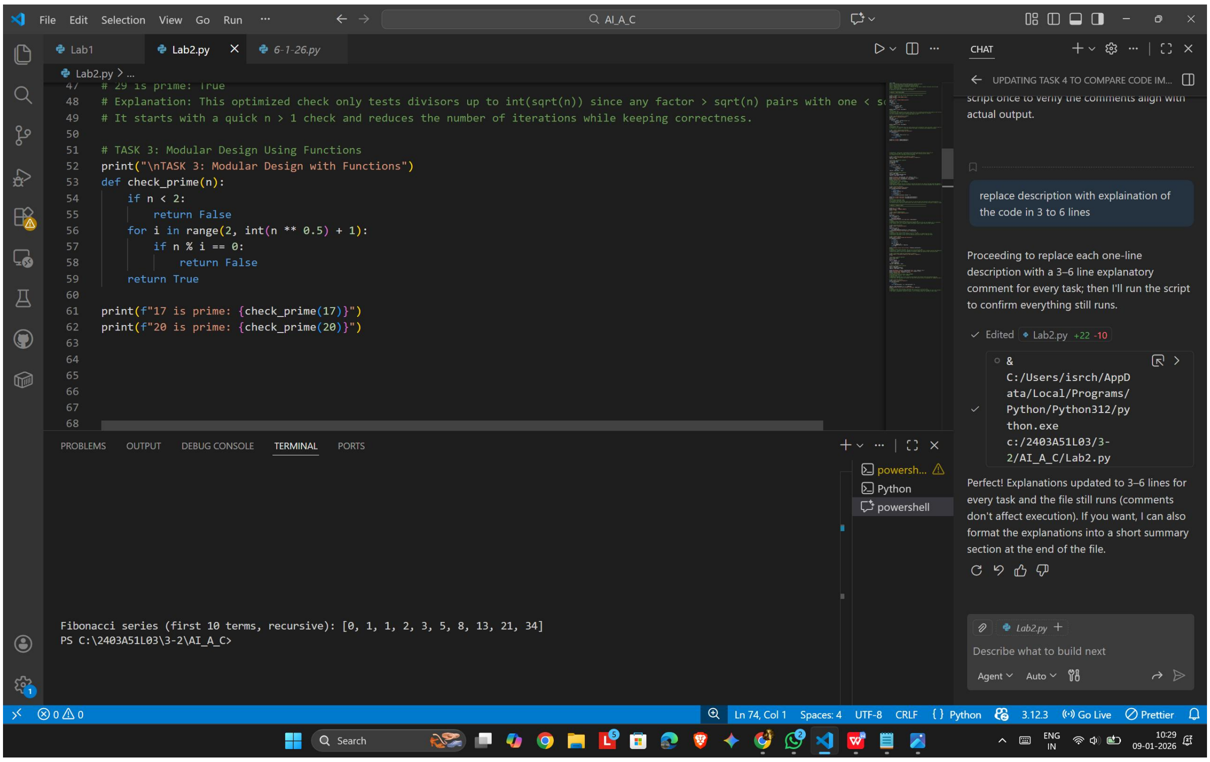
Explanation:

This optimized check only tests divisors up to int(sqrt(n)) since any factor > sqrt(n) pairs with one < sqrt(n).
It starts with a quick n > 1 check and reduces the number of iterations while keeping correctness.

TASK-3:

Prompt:
# TASK 3: Modular Design Using Functions

Code:



```
# 29 is prime: True
# Explanation: This optimized check only tests divisors up to int(sqrt(n)) since any factor > sqrt(n) pairs with one < s
# It starts with a quick n > 1 check and reduces the number of iterations while keeping correctness.

# TASK 3: Modular Design Using Functions
print("\nTASK 3: Modular Design with Functions")
def check_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

print(f"17 is prime: {check_prime(17)}")
print(f"20 is prime: {check_prime(20)}")
```

```
Fibonacci series (first 10 terms, recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\2403A51L03\3-2\AI_A_C>
```

CHAT — UPDATING TASK 4 TO COMPARE CODE IM...

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

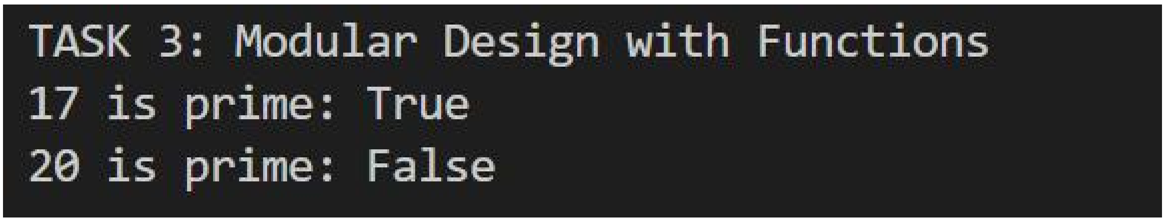
✓ Edited Lab2.py +22 -10

```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

File Edit Selection View Go Run
AI_A_C
Lab1
Lab2.py
6-1-26.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powersh...
Python
powershell
Lab2.py
Describe what to build next
Agent
Auto
Ln 74, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.12.3 Go Live Prettier
Search
ENG IN
10:29
09-01-2026

Output:



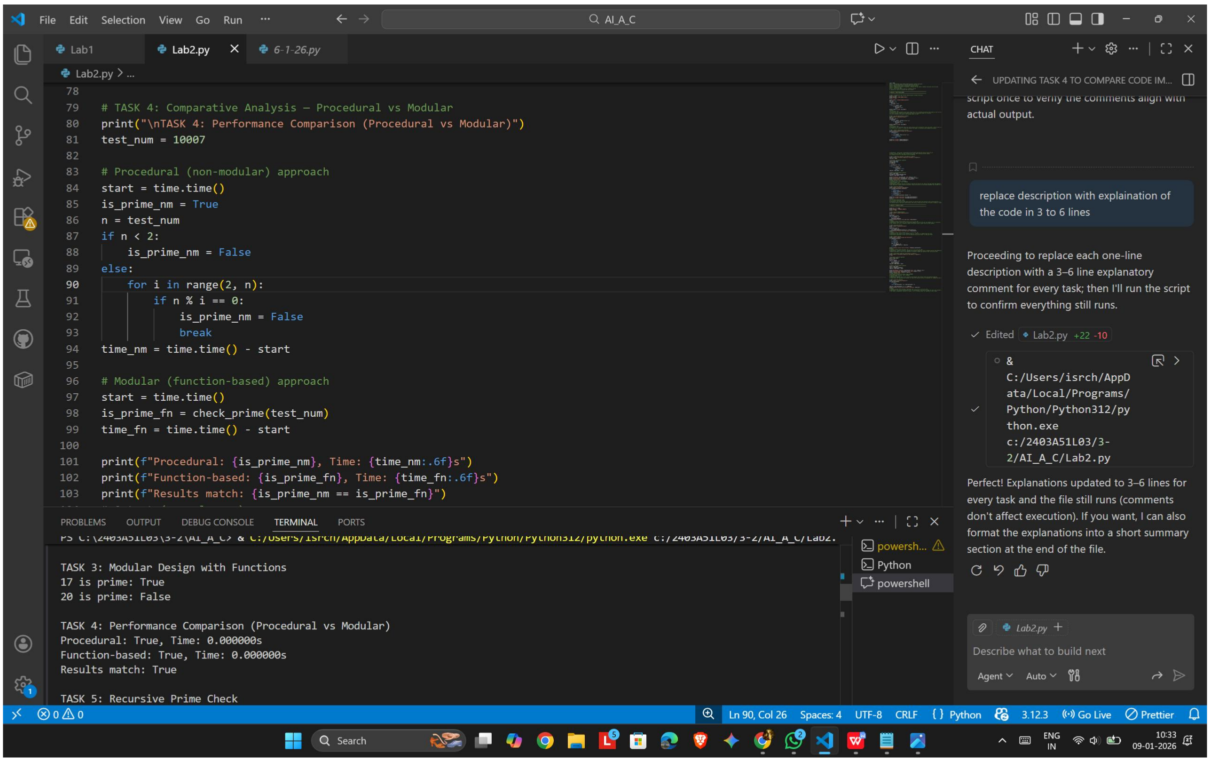
```
TASK 3: Modular Design with Functions
17 is prime: True
20 is prime: False
```

Explaination:
'check_prime' encapsulates the sqrt-based algorithm and returns False for n<2.
Using a function makes the logic reusable and clearer when checking multiple numbers.
It produces True for 17 and False for 20 as expected.

## TASK -4:

Prompt:
# TASK 4: Comparative Analysis — Procedural vs Modular

Code:


File Edit Selection View Go Run ···
AI_A_C
Lab1
Lab2.py
6-1-26.py
Lab2.py > ...
CHAT
UPDATING TASK 4 TO COMPARE CODE IM...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powersh...
Python
powershell
Ln 90, Col 26 Spaces: 4 UTF-8 CRLF { } Python 3.12.3 Go Live Prettier
Search
ENG IN
10:33
09-01-2026

```python
# TASK 4: Comparative Analysis — Procedural vs Modular
print("\nTASK 4: Performance Comparison (Procedural vs Modular)")
test_num = 10007

# Procedural (non-modular) approach
start = time.time()
is_prime_nm = True
n = test_num
if n < 2:
    is_prime_nm = False
else:
    for i in range(2, n):
        if n % i == 0:
            is_prime_nm = False
            break
time_nm = time.time() - start

# Modular (function-based) approach
start = time.time()
is_prime_fn = check_prime(test_num)
time_fn = time.time() - start

print(f"Procedural: {is_prime_nm}, Time: {time_nm:.6f}s")
print(f"Function-based: {is_prime_fn}, Time: {time_fn:.6f}s")
print(f"Results match: {is_prime_nm == is_prime_fn}")
```

```
PS C:\2403A51L03\3-2\AI_A_C> & C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.

TASK 3: Modular Design with Functions
17 is prime: True
20 is prime: False

TASK 4: Performance Comparison (Procedural vs Modular)
Procedural: True, Time: 0.000000s
Function-based: True, Time: 0.000000s
Results match: True

TASK 5: Recursive Prime Check
```

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

✓ Edited Lab2.py +22 -10

```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

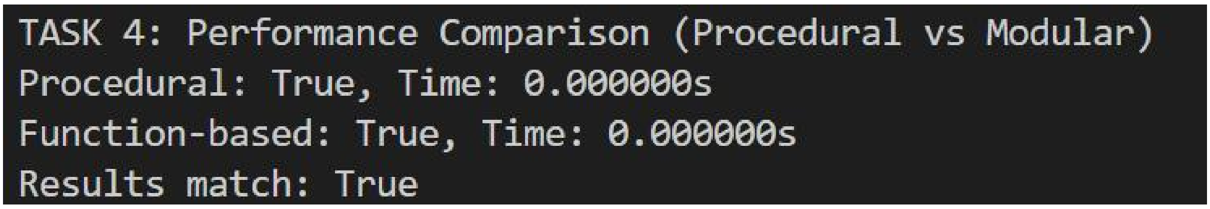
Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Lab2.py

Describe what to build next

Agent Auto

Output:



```
TASK 4: Performance Comparison (Procedural vs Modular)
Procedural: True, Time: 0.000000s
Function-based: True, Time: 0.000000s
Results match: True
```

Explanation:

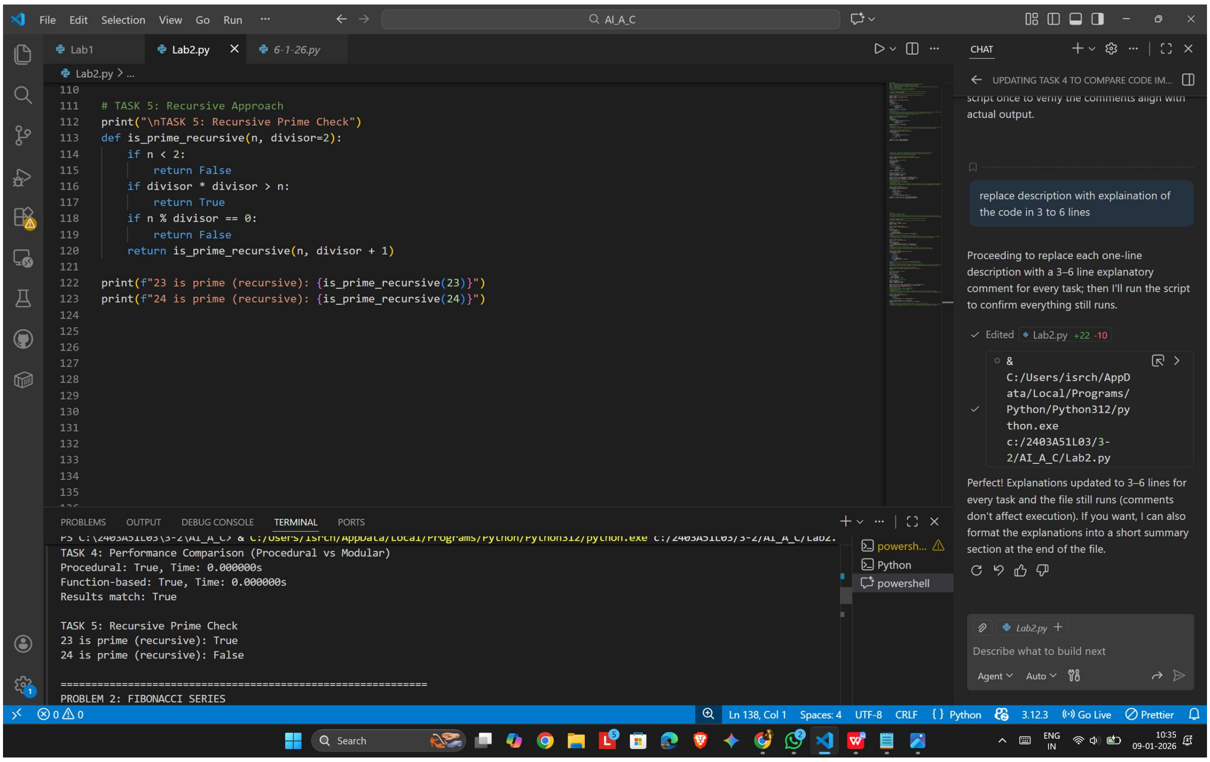
Times how long the non-modular loop and the function call take on the same input and compares results.
Both approaches compute primality and agree; timing differences are environment-dependent and intended for basic comparison.

## TASK - 5:

Prompt:

# TASK 5: Recursive Approach

Code:



```python
# TASK 5: Recursive Approach
print("\nTASK 5: Recursive Prime Check")
def is_prime_recursive(n, divisor=2):
    if n < 2:
        return False
    if divisor * divisor > n:
        return True
    if n % divisor == 0:
        return False
    return is_prime_recursive(n, divisor + 1)

print(f"23 is prime (recursive): {is_prime_recursive(23)}")
print(f"24 is prime (recursive): {is_prime_recursive(24)}")
```

```
PS C:\2403A51L03\3-2\AI_A_C> & C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.
TASK 4: Performance Comparison (Procedural vs Modular)
Procedural: True, Time: 0.000000s
Function-based: True, Time: 0.000000s
Results match: True

TASK 5: Recursive Prime Check
23 is prime (recursive): True
24 is prime (recursive): False

============================================================
PROBLEM 2: FIBONACCI SERIES
```

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

✓ Edited Lab2.py +22 -10

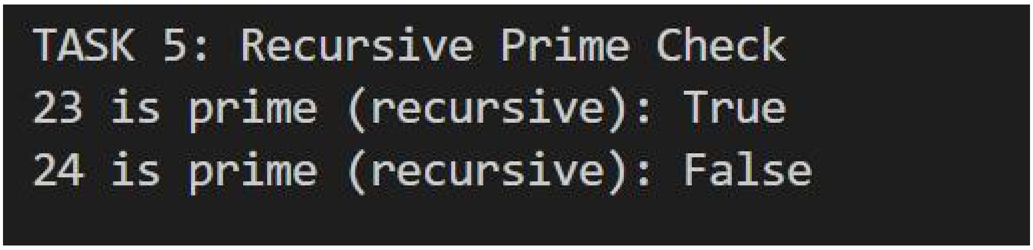
```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Lab2.py

Describe what to build next

Output:



```
TASK 5: Recursive Prime Check
23 is prime (recursive): True
24 is prime (recursive): False
```

Explanation:

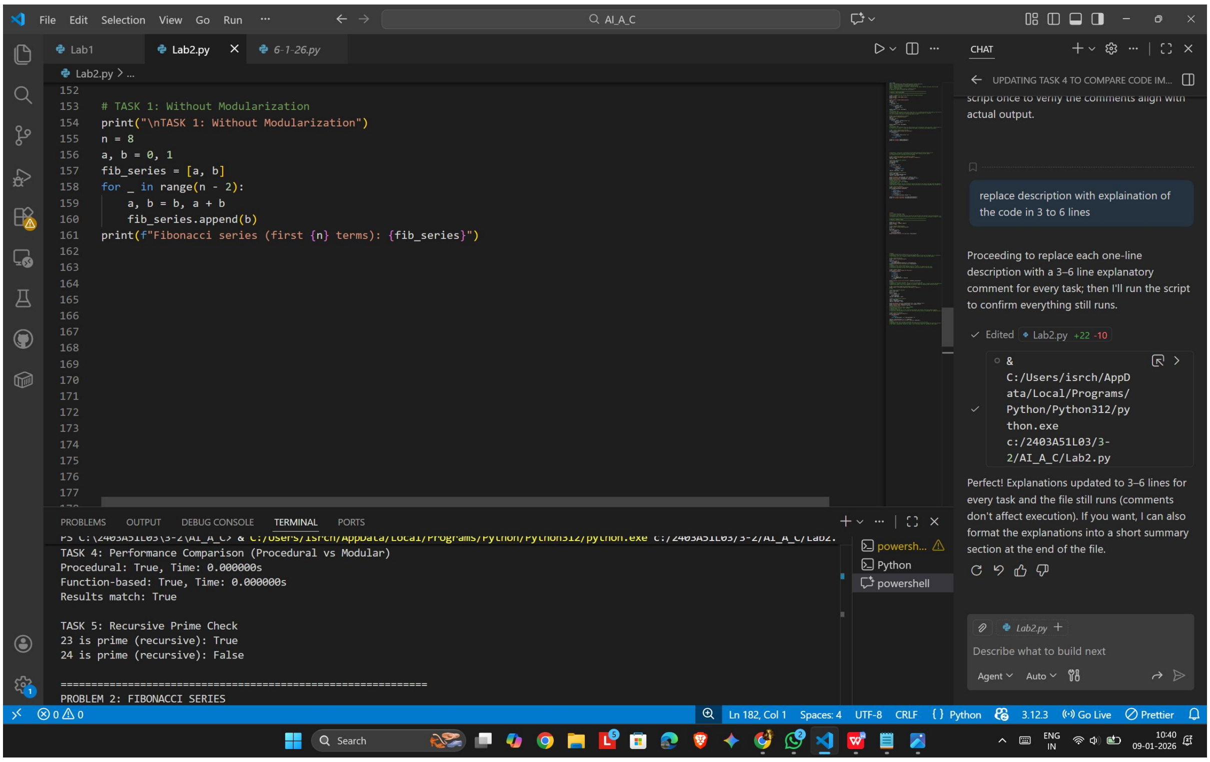
The recursive check tests divisors by calling itself with divisor+1 until divisor*divisor > n.
It returns False on the first found divisor; this form is clear but can be less efficient or deeper on large n.

Problem -2 : Fibonacci Series

TASK-1 :

Prompt:
# TASK 1: Without Modularization

Code:



File Edit Selection View Go Run ···
AI_A_C
Lab1
Lab2.py
6-1-26.py
Lab2.py > ...

```python
# TASK 1: Without Modularization
print("\nTASK 1: Without Modularization")
n = 8
a, b = 0, 1
fib_series = [a, b]
for _ in range(n - 2):
    a, b = b, a + b
    fib_series.append(b)
print(f"Fibonacci series (first {n} terms): {fib_series}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powersh...
Python
powershell

```
PS C:\2403A51L03\3-2\AI_A_C> & C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.
TASK 4: Performance Comparison (Procedural vs Modular)
Procedural: True, Time: 0.000000s
Function-based: True, Time: 0.000000s
Results match: True

TASK 5: Recursive Prime Check
23 is prime (recursive): True
24 is prime (recursive): False

============================================================
PROBLEM 2: FIBONACCI SERIES
```

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

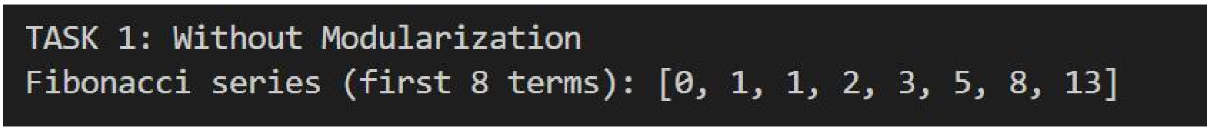
✓ Edited Lab2.py +22 -10

```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Lab2.py
Describe what to build next
Agent
Auto
Ln 182, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.12.3 Go Live Prettier
Search
ENG IN
10:40
09-01-2026

Output:



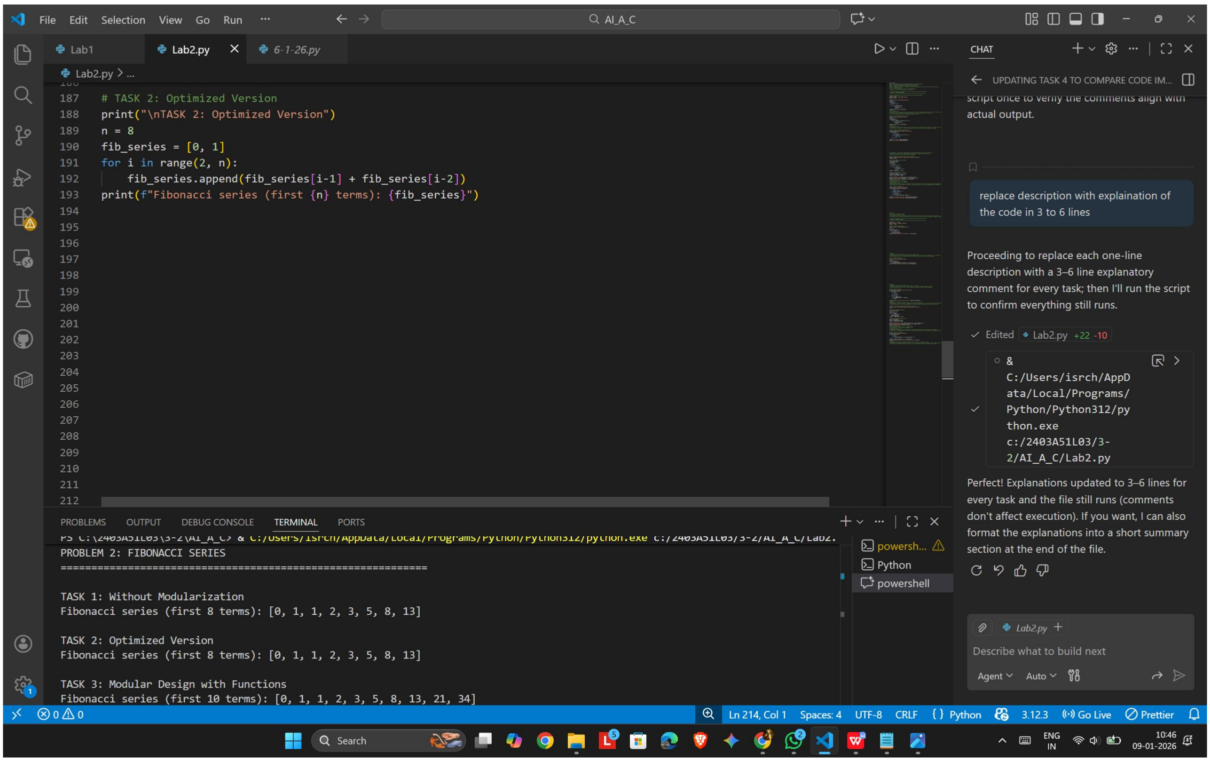
```
TASK 1: Without Modularization
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]
```

Explanation:
Starts with [0,1] and iteratively appends the sum of the last two elements for n-2 iterations.
This produces the first n Fibonacci numbers efficiently using a simple loop and tuple updates.

### TASK-2 :

Prompt:
# TASK 2: Optimized Version

Code:



File Edit Selection View Go Run ···
AI_A_C
Lab1
Lab2.py
6-1-26.py
Lab2.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powersh...
Python
powershell
CHAT
UPDATING TASK 4 TO COMPARE CODE IM...
Describe what to build next
Agent
Auto
Ln 214, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.12.3 Go Live Prettier
Search
ENG IN
10:46
09-01-2026

```python
# TASK 2: Optimized Version
print("\nTASK 2: Optimized Version")
n = 8
fib_series = [0, 1]
for i in range(2, n):
    fib_series.append(fib_series[i-1] + fib_series[i-2])
print(f"Fibonacci series (first {n} terms): {fib_series}")
```

```
PS C:\2403A51L03\3-2\AI_A_C> & C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.
PROBLEM 2: FIBONACCI SERIES
============================================================

TASK 1: Without Modularization
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]

TASK 2: Optimized Version
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]

TASK 3: Modular Design with Functions
Fibonacci series (first 10 terms): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

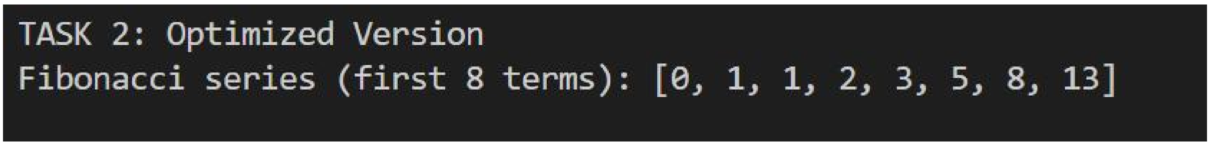
Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

✓ Edited Lab2.py +22 -10

```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Output:



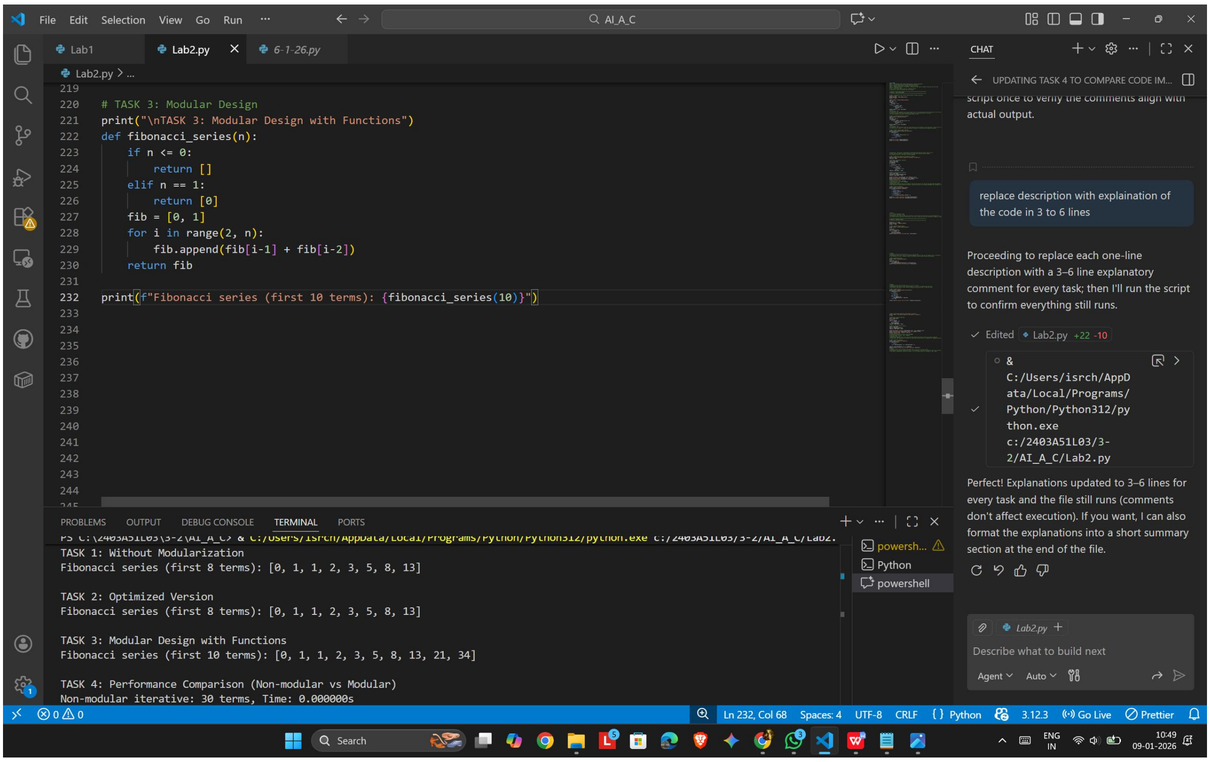
```
TASK 2: Optimized Version
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]
```

Explanation:
 Uses explicit list indexing (fib[i-1] + fib[i-2]) to compute each next term.
Functionally equivalent to Task 1 but the indexing style may be easier to read and extend.

TASK - 3:

Prompt:
# TASK 3: Modular Design

Code:



```
# TASK 3: Modular Design
print("\nTASK 3: Modular Design with Functions")
def fibonacci_series(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    fib = [0, 1]
    for i in range(2, n):
        fib.append(fib[i-1] + fib[i-2])
    return fib

print(f"Fibonacci series (first 10 terms): {fibonacci_series(10)}")
```

```
PS C:\2403A51L03\3-2\AI_A_C> & C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.
TASK 1: Without Modularization
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]

TASK 2: Optimized Version
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]

TASK 3: Modular Design with Functions
Fibonacci series (first 10 terms): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

TASK 4: Performance Comparison (Non-modular vs Modular)
Non-modular iterative: 30 terms, Time: 0.000000s
```

CHAT

UPDATING TASK 4 TO COMPARE CODE IM...

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

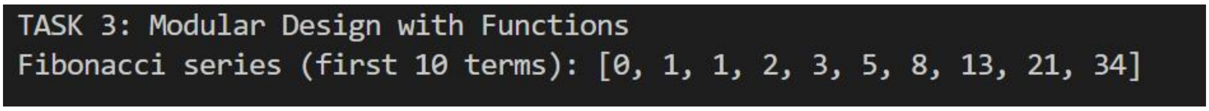
Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

✓ Edited Lab2.py +22 -10

```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Output:



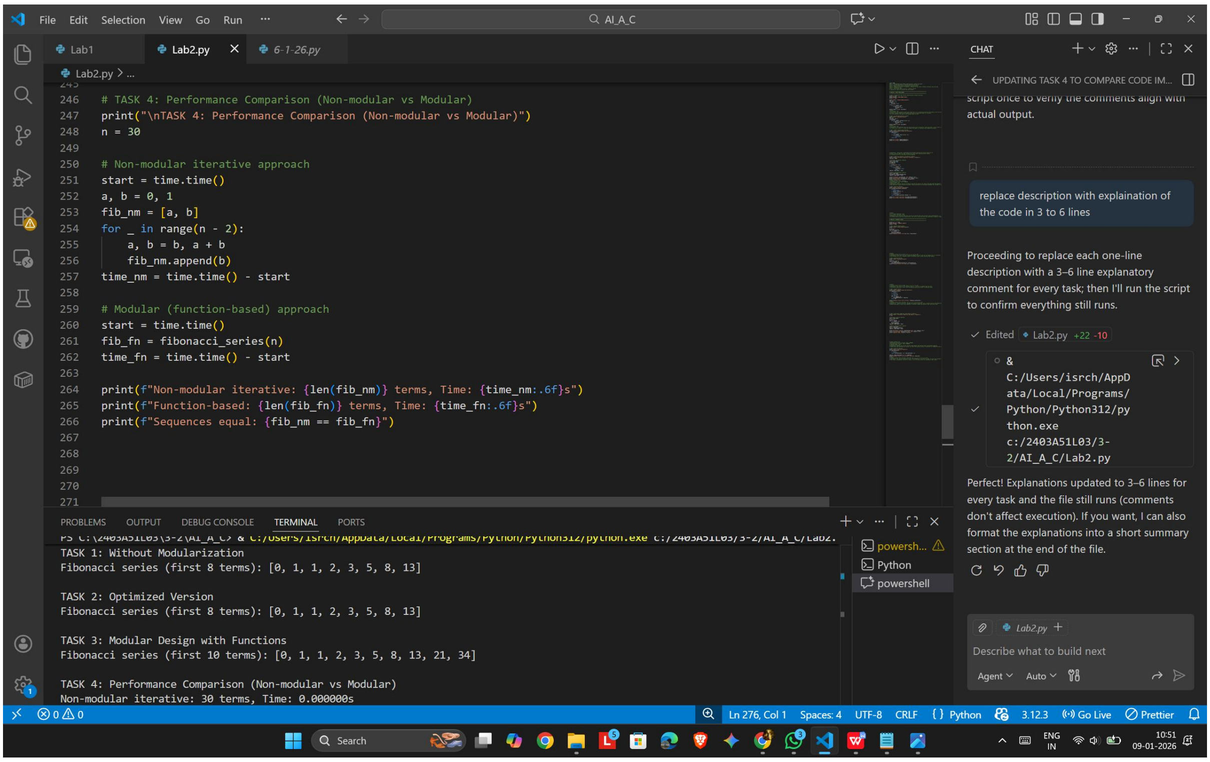
```
TASK 3: Modular Design with Functions
Fibonacci series (first 10 terms): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Explanation:
`fibonacci_series(n)` packages the iterative generation into a reusable function and handles edge cases.
Modularizing makes it easy to get sequences of different lengths and improves code clarity and reuse.

TASK-4:

Prompt:
# TASK 4: Performance Comparison (Non-modular vs Modular)

Code:



```python
# TASK 4: Performance Comparison (Non-modular vs Modular)
print("\nTASK 4: Performance Comparison (Non-modular vs Modular)")
n = 30

# Non-modular iterative approach
start = time.time()
a, b = 0, 1
fib_nm = [a, b]
for _ in range(n - 2):
    a, b = b, a + b
    fib_nm.append(b)
time_nm = time.time() - start

# Modular (function-based) approach
start = time.time()
fib_fn = fibonacci_series(n)
time_fn = time.time() - start

print(f"Non-modular iterative: {len(fib_nm)} terms, Time: {time_nm:.6f}s")
print(f"Function-based: {len(fib_fn)} terms, Time: {time_fn:.6f}s")
print(f"Sequences equal: {fib_nm == fib_fn}")
```

```
PS C:\2403A51L03\3-2\AI_A_C> & C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.
TASK 1: Without Modularization
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]

TASK 2: Optimized Version
Fibonacci series (first 8 terms): [0, 1, 1, 2, 3, 5, 8, 13]

TASK 3: Modular Design with Functions
Fibonacci series (first 10 terms): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

TASK 4: Performance Comparison (Non-modular vs Modular)
Non-modular iterative: 30 terms, Time: 0.000000s
```

CHAT

UPDATING TASK 4 TO COMPARE CODE IM...

script once to verify the comments align with actual output.

replace description with explaination of the code in 3 to 6 lines

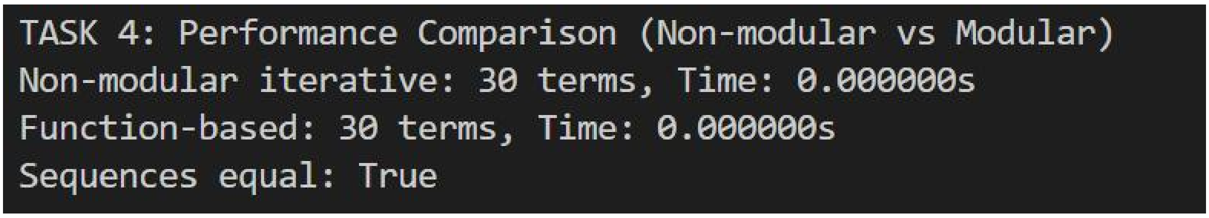
Proceeding to replace each one-line description with a 3–6 line explanatory comment for every task; then I'll run the script to confirm everything still runs.

✓ Edited Lab2.py +22 -10

```
& C:/Users/isrch/AppData/Local/Programs/Python/Python312/python.exe c:/2403A51L03/3-2/AI_A_C/Lab2.py
```

Perfect! Explanations updated to 3–6 lines for every task and the file still runs (comments don't affect execution). If you want, I can also format the explanations into a short summary section at the end of the file.

Output:



```
TASK 4: Performance Comparison (Non-modular vs Modular)
Non-modular iterative: 30 terms, Time: 0.000000s
Function-based: 30 terms, Time: 0.000000s
Sequences equal: True
```

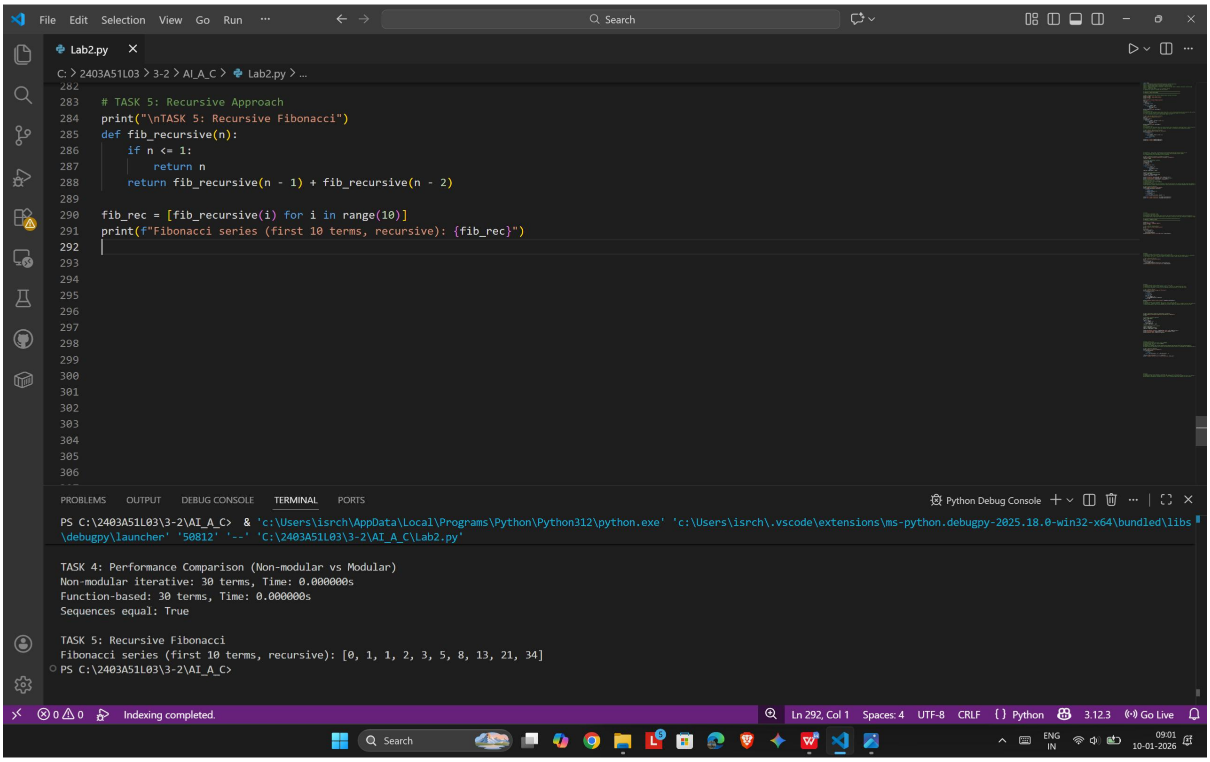
Explanation: Compares the in-line iterative build against the function result and confirms equality.
Timing shows both approaches are similar for n=30; the function version is preferable for readability and reuse.

## TASK -5:

Prompt:

# TASK 5: Recursive Approach

Code:

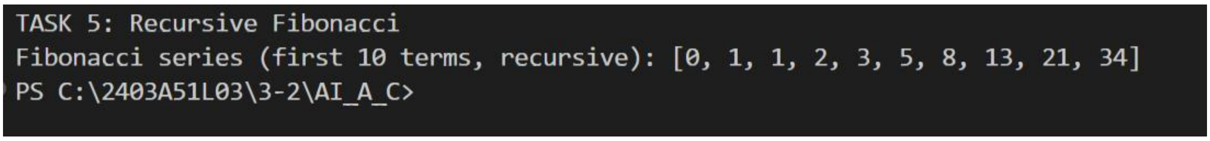


File Edit Selection View Go Run ···
Search
Lab2.py
C: > 2403A51L03 > 3-2 > AI_A_C > Lab2.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console
Indexing completed.
Ln 292, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.12.3 Go Live

```python
# TASK 5: Recursive Approach
print("\nTASK 5: Recursive Fibonacci")
def fib_recursive(n):
    if n <= 1:
        return n
    return fib_recursive(n - 1) + fib_recursive(n - 2)

fib_rec = [fib_recursive(i) for i in range(10)]
print(f"Fibonacci series (first 10 terms, recursive): {fib_rec}")
```

```
PS C:\2403A51L03\3-2\AI_A_C>  & 'c:\Users\isrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrch\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50812' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab2.py'

TASK 4: Performance Comparison (Non-modular vs Modular)
Non-modular iterative: 30 terms, Time: 0.000000s
Function-based: 30 terms, Time: 0.000000s
Sequences equal: True

TASK 5: Recursive Fibonacci
Fibonacci series (first 10 terms, recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\2403A51L03\3-2\AI_A_C>
```

Output:



```
TASK 5: Recursive Fibonacci
Fibonacci series (first 10 terms, recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\2403A51L03\3-2\AI_A_C>
```

Explanation:

The recursive definition mirrors the mathematical recurrence but makes two calls per non-base case.

This leads to exponential runtime for larger n, so it is mainly useful for teaching or small inputs.