ASSIGNMENT-5.5

ROLL-NO:2303A51932

BATCH-30

TASK-1

PROMPT: Generate Python code for two prime-checking methods and

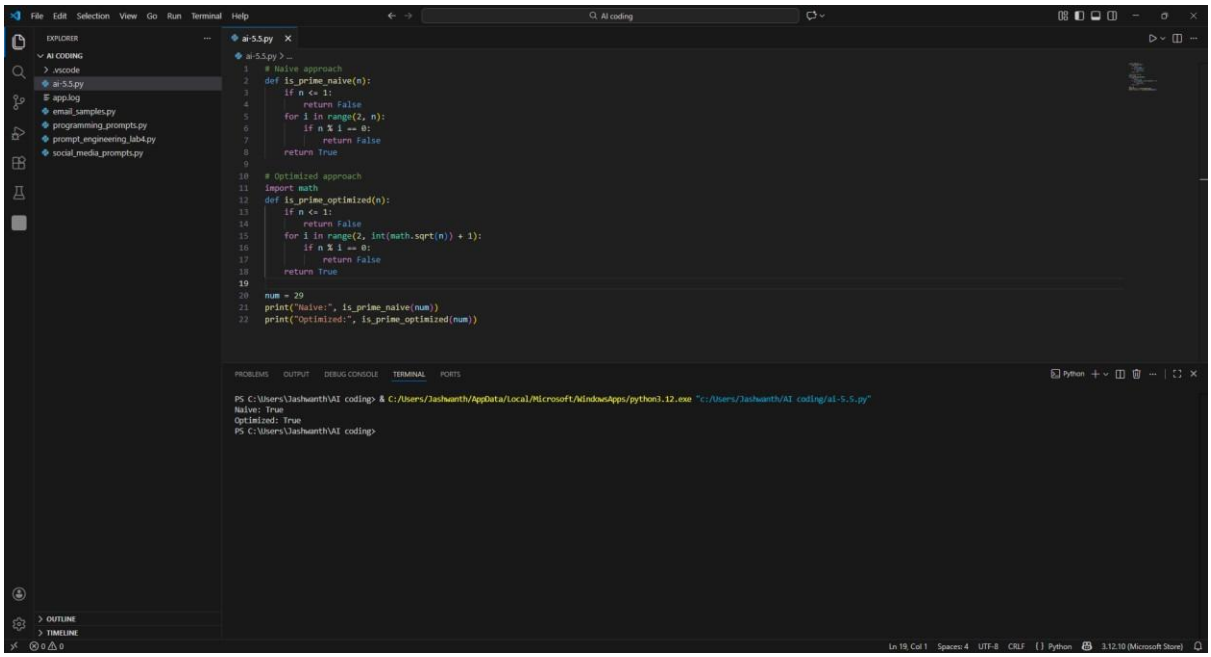
explain how the optimized version improves performance Generate

Python code for two prime-checking methods:

1) Naive approach 2)

Optimized approach

CODE:



OBSERVATION:

The naive method checks divisibility from 2 up to n−1, so it performs many unnecessary iterations for large numbers.

The optimized method only checks divisibility up to √n, because any factor larger than √n must have a corresponding smaller factor already checked.

The time complexity of the naive approach is O(n), which makes it slow when n becomes large.

The time complexity of the optimized approach is O(√n), which significantly reduces the number of operations.

Both methods produce the same correct result, but the optimized method reaches the answer much faster.

Thus, the optimized approach improves performance by reducing redundant checks while maintaining correctness.

TASK-2

PROMPT:

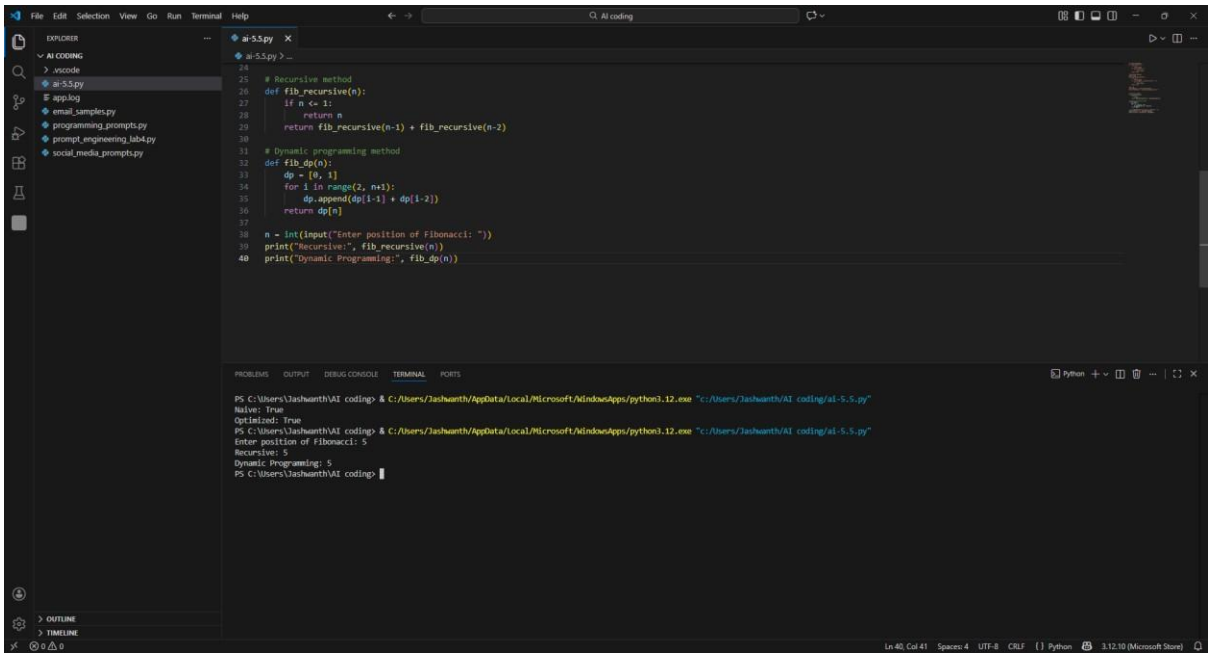
Generate Python code for Fibonacci using:

1) Recursive method

2) Dynamic programming method

Explain time complexity and performance improvement.

CODE:



OBSERVATION:

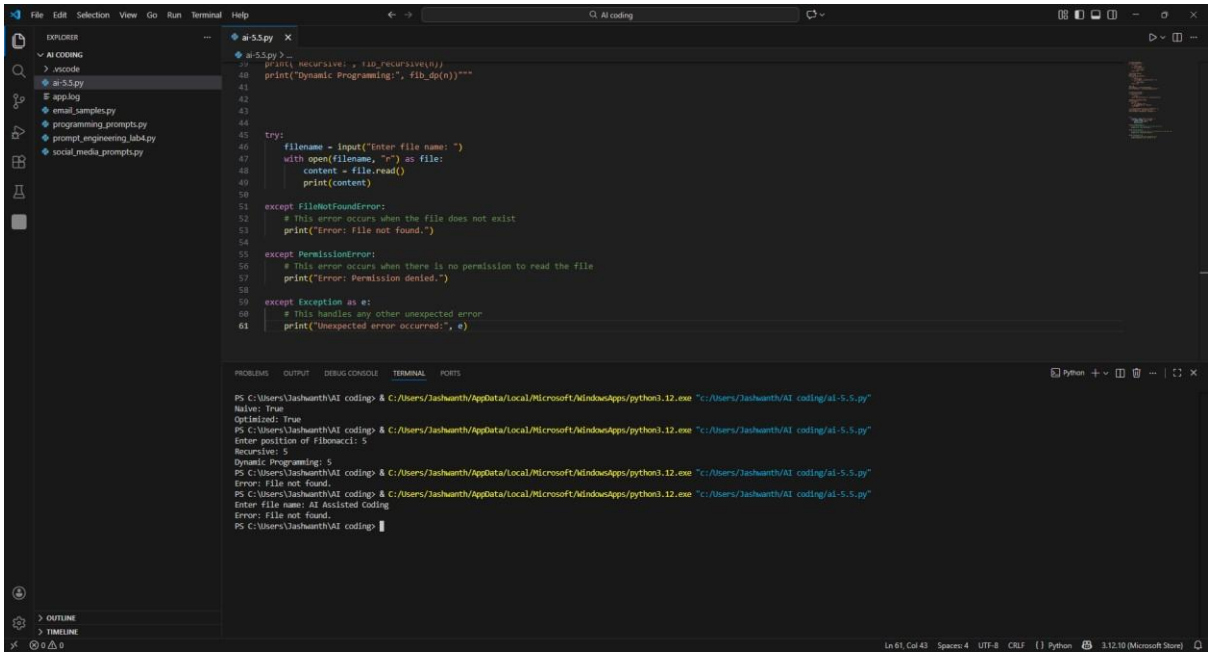
The recursive method recomputes the same values many times. The DP method stores previous results to avoid recomputation. The recursive method has exponential time complexity. The DP method has linear time complexity. Both methods produce the same Fibonacci value. The optimized method performs much faster for large n. TASK-3

PROMPT:

Generate Python code that reads a file and processes data with proper error handling.

Explain each exception clearly using comments.

CODE:



OBSERVATION:

The program clearly separates different types of errors. Each exception is handled with a meaningful message. File NotFoundError explains missing file issues. PermissionError explains access-related problems. A general exception block handles unknown runtime errors. The explanations match the behaviour seen during execution.

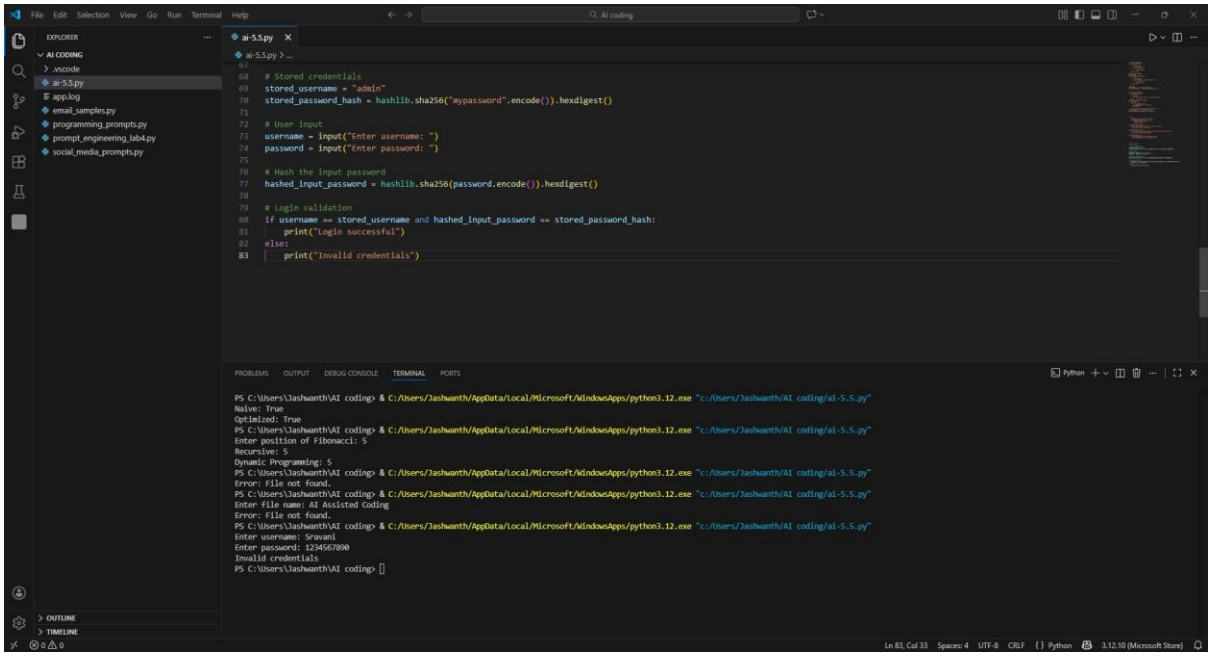
TASK-4

PROMPT:

Generate a Python-based login system.

Analyze security flaws and provide a revised secure version using password hashing and input validation.

CODE:



OBSERVATION:

storing passwords in plain text is a serious security risk. Hashing ensures passwords are not stored in readable form. User input is validated before authentication. The system compares hashed values instead of raw passwords. This reduces the risk of password leakage. Secure authentication improves protection against attacks.

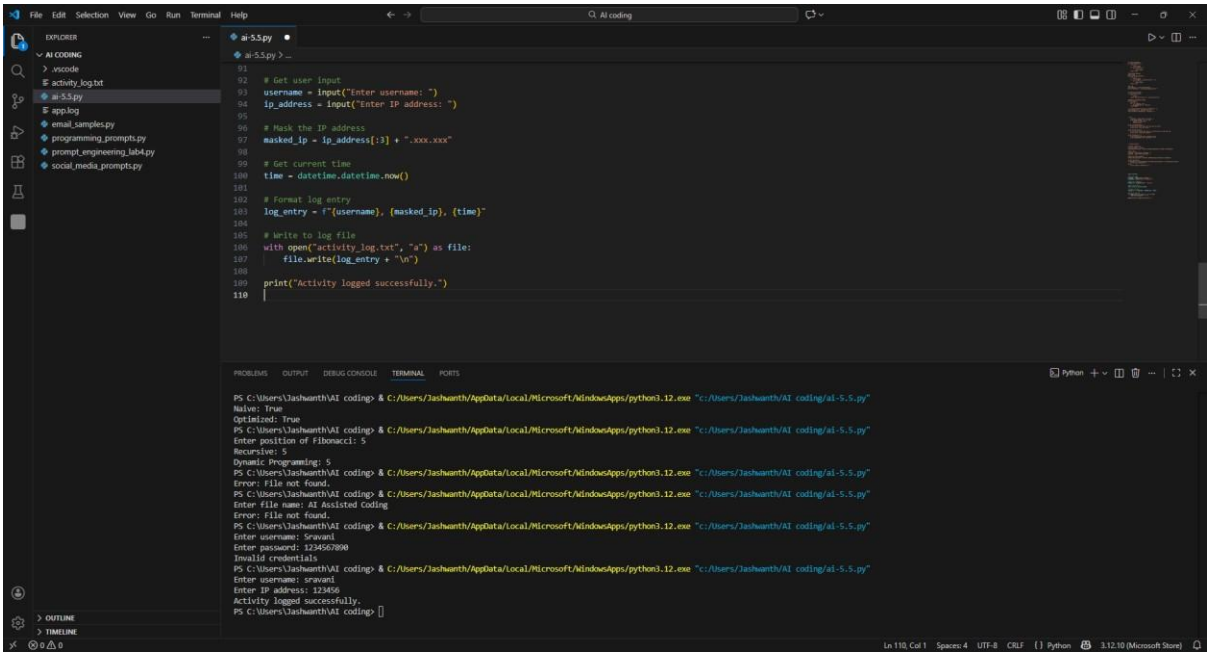
TASK-5

PROMPT:

Generate a Python script that logs user activity.

Analyze privacy risks and provide an improved version using masked or minimal logging.

CODE:



OBSERVATION:

Logging full IP addresses can expose user identity. Masking the IP reduces the risk of tracking users. Only necessary information is stored in logs. Sensitive data is not written in raw form. Minimal logging supports user privacy. Privacy-aware logging prevents misuse of stored data.