

ASSIGNMENT-5.5

ROLL-NO:2303A51932

BATCH-30

TASK-1

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

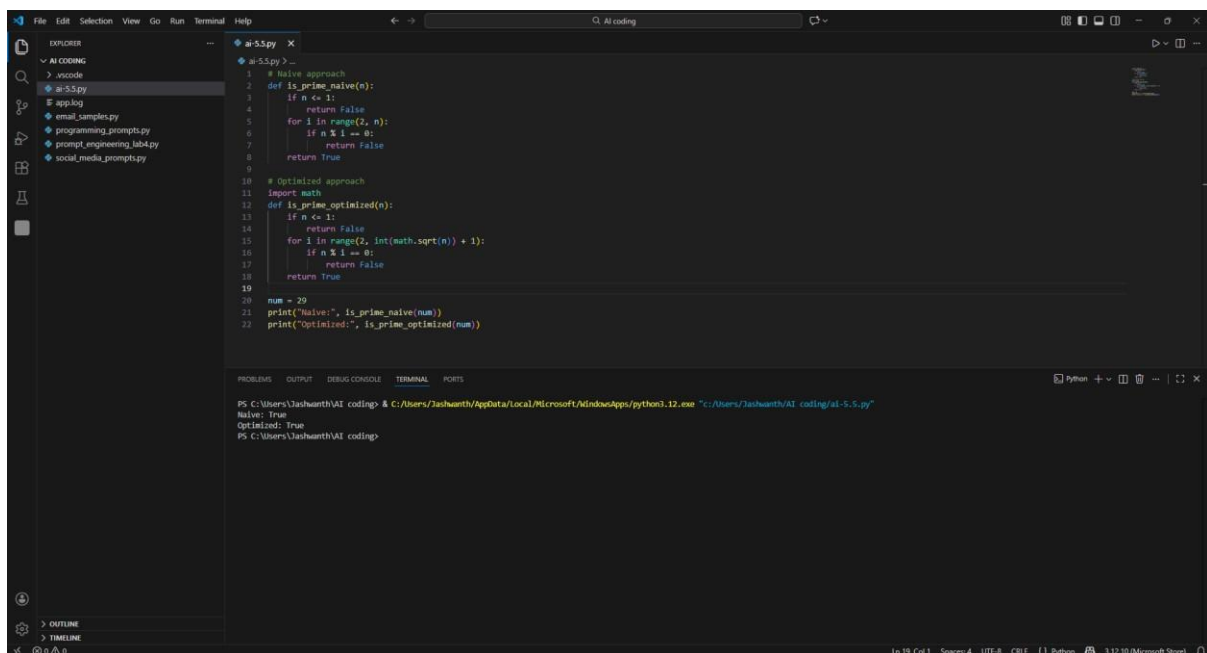
explain how the optimized version improves performance Generate

Python code for two prime-checking methods:

1) Naive approach 2)

Optimized approach

CODE:



```
1 # Naive approach
2 def is_prime_naive(n):
3     if n <= 1:
4         return False
5     for i in range(2, n):
6         if n % i == 0:
7             return False
8     return True
9
10 # Optimized approach
11 import math
12 def is_prime_optimized(n):
13     if n <= 1:
14         return False
15     for i in range(2, int(math.sqrt(n)) + 1):
16         if n % i == 0:
17             return False
18     return True
19
20 num = 29
21 print("Naive:", is_prime_naive(num))
22 print("Optimized:", is_prime_optimized(num))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Jashwanth\AI coding> & C:\Users\Jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\Jashwanth\AI coding\ai-5.5.py"
Naive: True
Optimized: True
PS C:\Users\Jashwanth\AI coding>
```

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 \sqrt{n} , because any factor larger than \sqrt{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

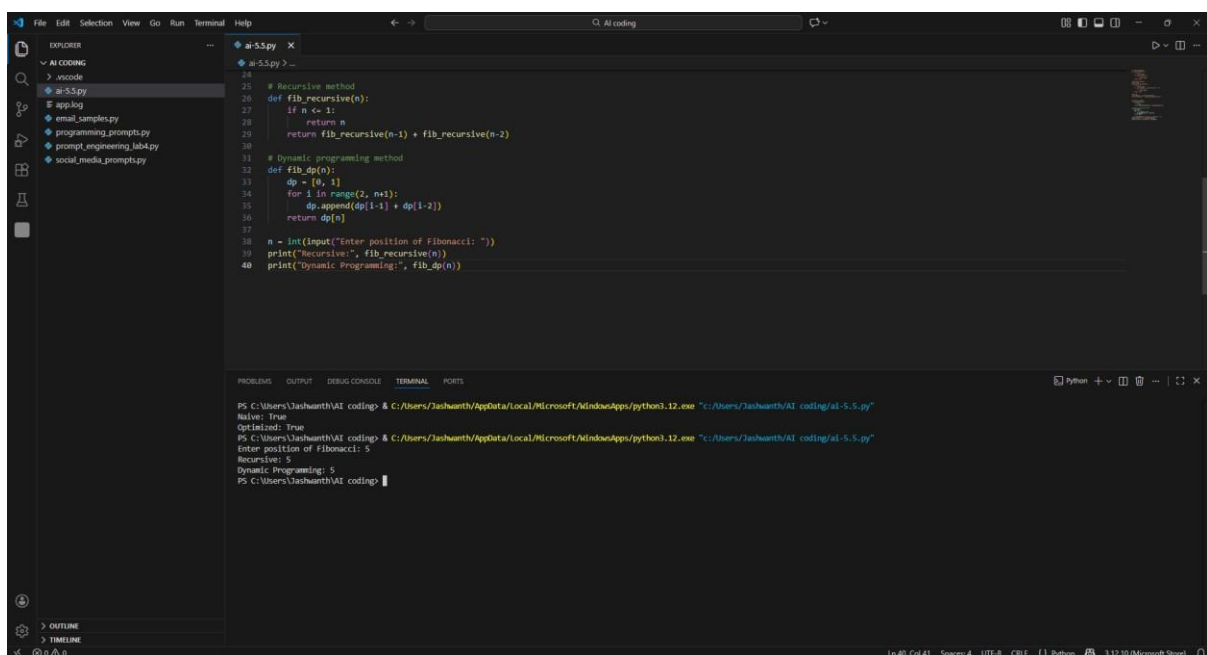
PROMPT:

Generate Python code for Fibonacci using:

- 1) Recursive method
- 2) Dynamic programming method

Explain time complexity and performance improvement.

CODE:



```
24
25 # Recursive method
26 def fib_recursive(n):
27     if n <= 1:
28         return n
29     return fib_recursive(n-1) + fib_recursive(n-2)
30
31 # Dynamic programming method
32 def fib_dp(n):
33     dp = [0, 1]
34     for i in range(2, n+1):
35         dp.append(dp[i-1] + dp[i-2])
36     return dp[n]
37
38 n = int(input("Enter position of Fibonacci: "))
39 print("Recursive:", fib_recursive(n))
40 print("Dynamic Programming:", fib_dp(n))
```

```
PS C:\Users\Jashwanth\AI coding> & C:\Users\Jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\Jashwanth\AI coding\ai-5.5.py"
Enter position of Fibonacci: 5
Recursive: 5
Dynamic Programming: 5
PS C:\Users\Jashwanth\AI coding>
```

OBSERVATION:

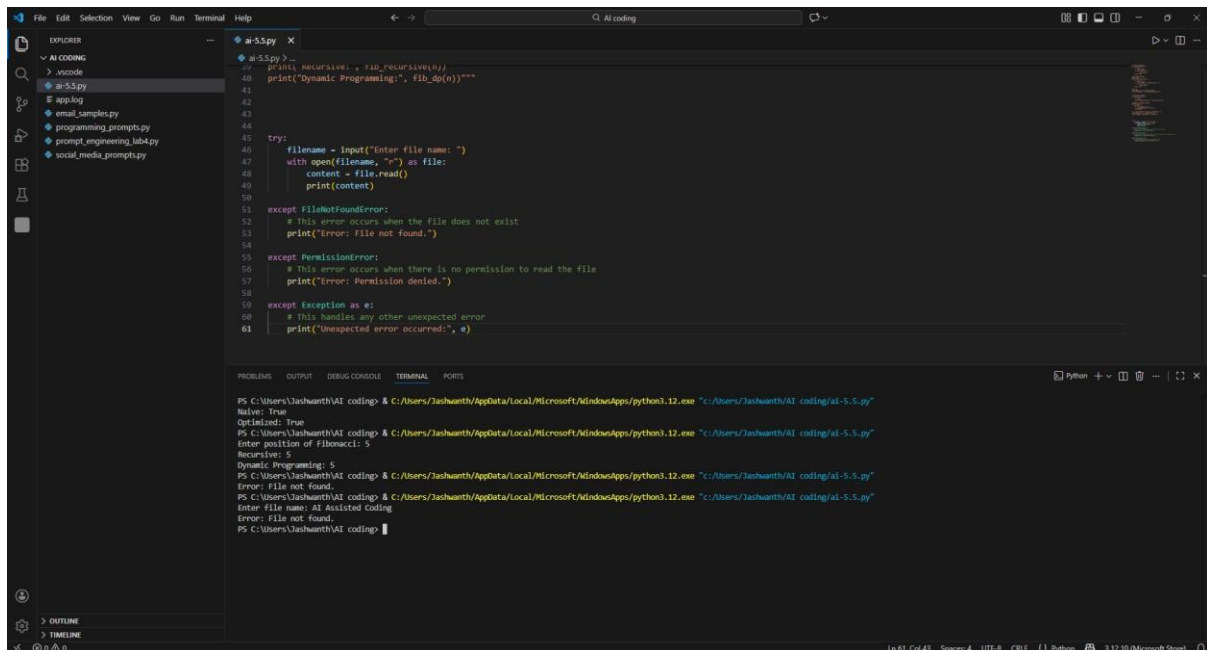
The recursive method recomputes the same values many times. The DP method stores previous results to avoid recomputation. The recursive method has exponential time complexity. The DP method has linear time complexity. Both methods produce the same Fibonacci value. The optimized method performs much faster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

PROMPT:

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

Explain each exception clearly using comments.

CODE:



```
37 def fib_recursive(n):
38     print("Dynamic Programming:", fib_dp(n))
39
40
41
42
43
44
45 try:
46     filename = input("Enter file name: ")
47     with open(filename, "r") as file:
48         content = file.read()
49         print(content)
50
51 except FileNotFoundError:
52     # This error occurs when the file does not exist
53     print("Error: File not found.")
54
55 except PermissionError:
56     # This error occurs when there is no permission to read the file
57     print("Error: Permission denied.")
58
59 except Exception as e:
60     # This handles any other unexpected error
61     print("Unexpected error occurred:", e)
```

```
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\jashwanth\AI coding\ai-5.5.py"
Naive: True
Optimized: True
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\jashwanth\AI coding\ai-5.5.py"
Enter position of Fibonacci: 5
Recursive: 5
Dynamic Programming: 5
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\jashwanth\AI coding\ai-5.5.py"
Error: File not found.
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\jashwanth\AI coding\ai-5.5.py"
Enter file name: AI Assisted Coding
Error: File not found.
PS C:\Users\jashwanth\AI coding>
```

OBSERVATION:

The program clearly separates different types of errors. Each exception is handled with a meaningful message. File Not Found error 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:

The image shows a Visual Studio Code editor window with a Python file named `ai-55.py` open. The script implements a simple login system. It defines stored credentials (username: "admin", password hash: "mypassword".encode().hexdigest()) and prompts the user for their username and password. The input password is hashed using `hashlib.sha256` and compared to the stored hash. If they match, it prints "Login successful"; otherwise, it prints "Invalid credentials".

```
68 # Stored credentials
69 stored_username = "admin"
70 stored_password_hash = hashlib.sha256("mypassword".encode()).hexdigest()
71
72 # User Input
73 username = input("Enter username: ")
74 password = input("Enter password: ")
75
76 # Hash the input password
77 hashed_input_password = hashlib.sha256(password.encode()).hexdigest()
78
79 # Login validation
80 if username == stored_username and hashed_input_password == stored_password_hash:
81     print("Login successful")
82 else:
83     print("Invalid credentials")
```

Below the editor, the terminal shows the script being executed. It prompts for a file name, then for a username and password. The user enters "Sriyani" and "1234567890", which results in "Invalid credentials" being printed.

```
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python.12.exe "C:\Users\jashwanth\AI coding\ai-5-5.py"
Halve: True
Optimized: True
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python.12.exe "C:\Users\jashwanth\AI coding\ai-5-5.py"
Enter position of Fibonacci: 5
Recursive: 5
Dynamic Programming: 5
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python.12.exe "C:\Users\jashwanth\AI coding\ai-5-5.py"
Error: file not found.
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python.12.exe "C:\Users\jashwanth\AI coding\ai-5-5.py"
Enter file name: AI Assisted Coding
Error: file not found.
PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python.12.exe "C:\Users\jashwanth\AI coding\ai-5-5.py"
Enter username: Sriyani
Enter password: 1234567890
Invalid credentials
PS C:\Users\jashwanth\AI coding>
```

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.

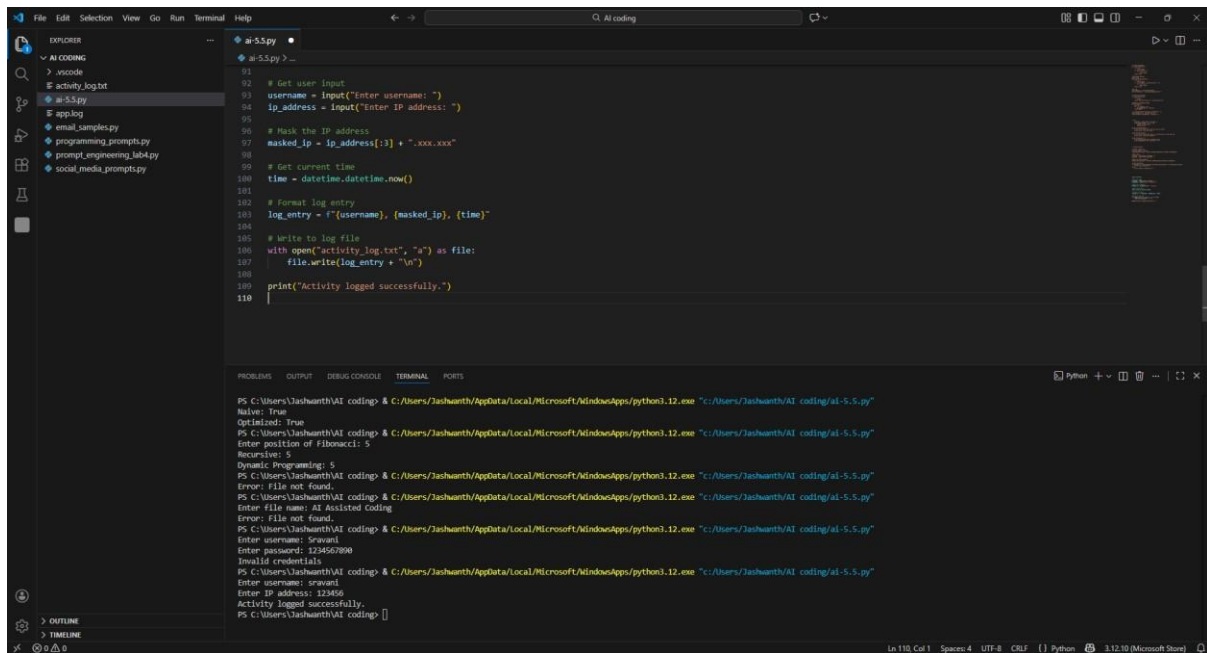
TASK-5

PROMPT:

Generate a Python script that logs user activity.

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

CODE:



```
91
92 # Get user input
93 username = input("Enter username: ")
94 ip_address = input("Enter IP address: ")
95
96 # Mask the IP address
97 masked_ip = ip_address[:3] + ".xxx.xxx"
98
99 # Get current time
100 time = datetime.datetime.now()
101
102 # Format log entry
103 log_entry = f"{username}, {masked_ip}, {time}"
104
105 # Write to log file
106 with open("activity_log.txt", "a") as file:
107     file.write(log_entry + "\n")
108
109 print("Activity logged successfully.")
110
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + Python 3.12.0 (Microsoft Store)

PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/jashwanth/ai coding/ai-5.5.py"

Halve: True

Optimized: True

PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/jashwanth/ai coding/ai-5.5.py"

Enter position of Fibonacci: 5

Recursive: 5

Dynamic Programming: 5

PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/jashwanth/ai coding/ai-5.5.py"

Error: file not found.

PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/jashwanth/ai coding/ai-5.5.py"

Enter file name: AI Assisted Coding

Error: file not found.

PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/jashwanth/ai coding/ai-5.5.py"

Enter username: Swami

Enter password: 1234567890

Invalid credentials

PS C:\Users\jashwanth\AI coding> & C:\Users\jashwanth\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/jashwanth/ai coding/ai-5.5.py"

Enter username: Swami

Enter IP address: 123456

Activity logged successfully.

PS C:\Users\jashwanth\AI coding>

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.