

ASSIGNMENT-5.5

ROLL-NO:2306A91001

BATCH-30

TASK-1

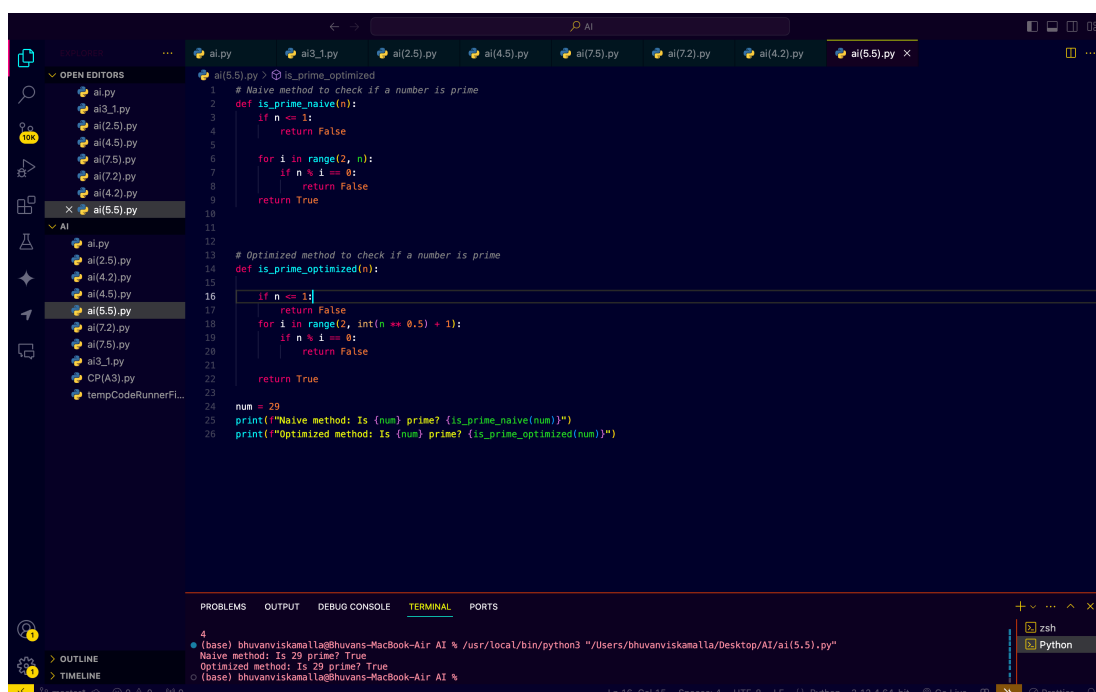
PROMPT: Generate Python code for two prime-checking methods and

explain how the optimised version improves performance

Generate Python code for two prime-checking methods:

- 1) Naive approach
- 2) Optimised approach

CODE:



```
1 # Naive method to check if a number is prime
2 def is_prime_naive(n):
3     if n <= 1:
4         return False
5
6     for i in range(2, n):
7         if n % i == 0:
8             return False
9     return True
10
11
12
13 # Optimized method to check if a number is prime
14 def is_prime_optimized(n):
15
16     if n <= 1:
17         return False
18     for i in range(2, int(n ** 0.5) + 1):
19         if n % i == 0:
20             return False
21     return True
22
23
24 num = 29
25 print(f"Naive method: Is {num} prime? {is_prime_naive(num)}")
26 print(f"Optimized method: Is {num} prime? {is_prime_optimized(num)}")
```

4
(base) bhuvanviskamilla@bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamilla/Desktop/AI/ai(5.5).py"
Naive method: Is 29 prime? True
Optimized method: Is 29 prime? True
(base) bhuvanviskamilla@bhuvans-MacBook-Air AI %

OBSERVATION:

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

The optimised 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 optimised approach is $O(\sqrt{n})$, which significantly reduces the number of operations.

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

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

TASK-2

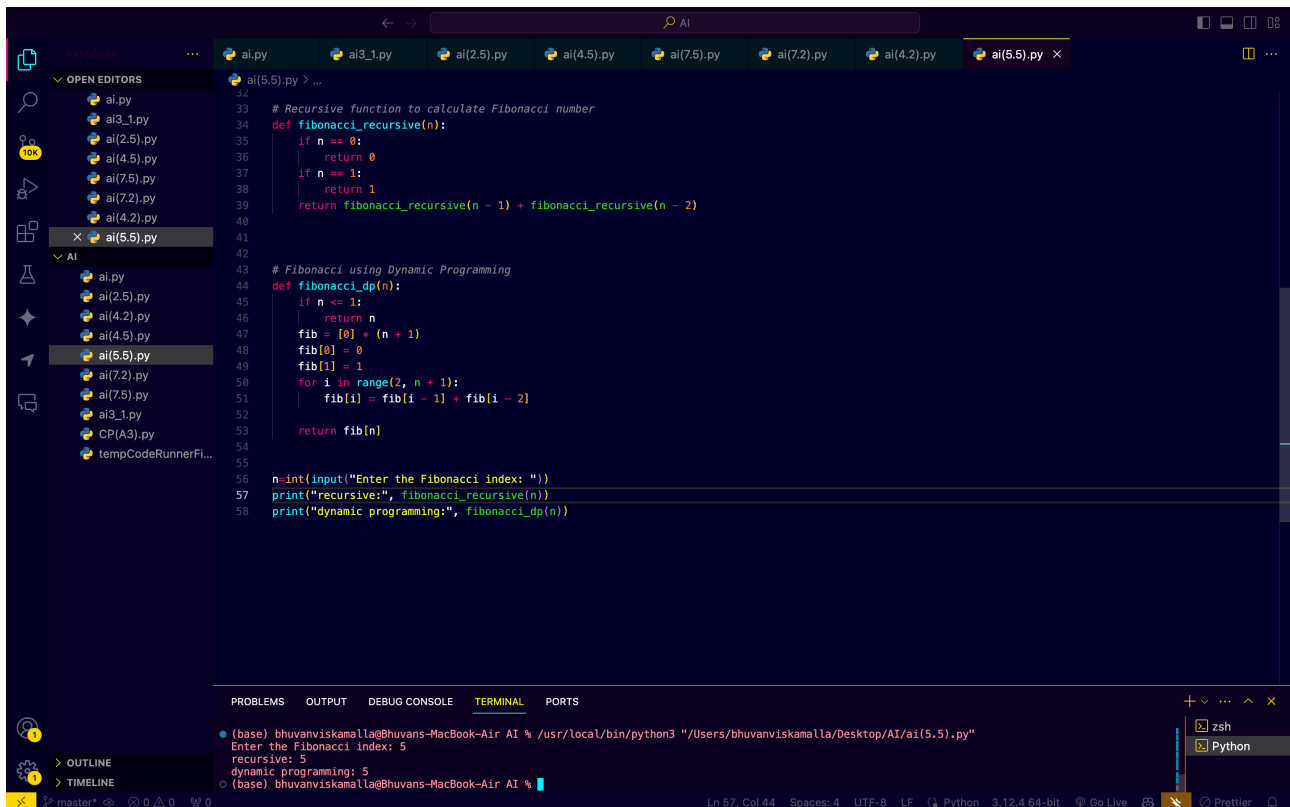
PROMPT:

Generate Python code for Fibonacci using:

- 1) Recursive method
- 2) Dynamic programming method

Explain time complexity and performance improvement.

CODE:



```
33 # Recursive function to calculate Fibonacci number
34 def fibonacci_recursive(n):
35     if n == 0:
36         return 0
37     if n == 1:
38         return 1
39     return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
40
41
42
43 # Fibonacci using Dynamic Programming
44 def fibonacci_dp(n):
45     if n <= 1:
46         return n
47     fib = [0] * (n + 1)
48     fib[0] = 0
49     fib[1] = 1
50     for i in range(2, n + 1):
51         fib[i] = fib[i - 1] + fib[i - 2]
52
53     return fib[n]
54
55
56 n=int(input("Enter the Fibonacci index: "))
57 print("recursive:", fibonacci_recursive(n))
58 print("dynamic programming:", fibonacci_dp(n))
```

Terminal Output:

```
(base) bhuvanviskamalla@Bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"
Enter the Fibonacci index: 5
recursive: 5
dynamic programming: 5
(base) bhuvanviskamalla@Bhuvans-MacBook-Air AI %
```

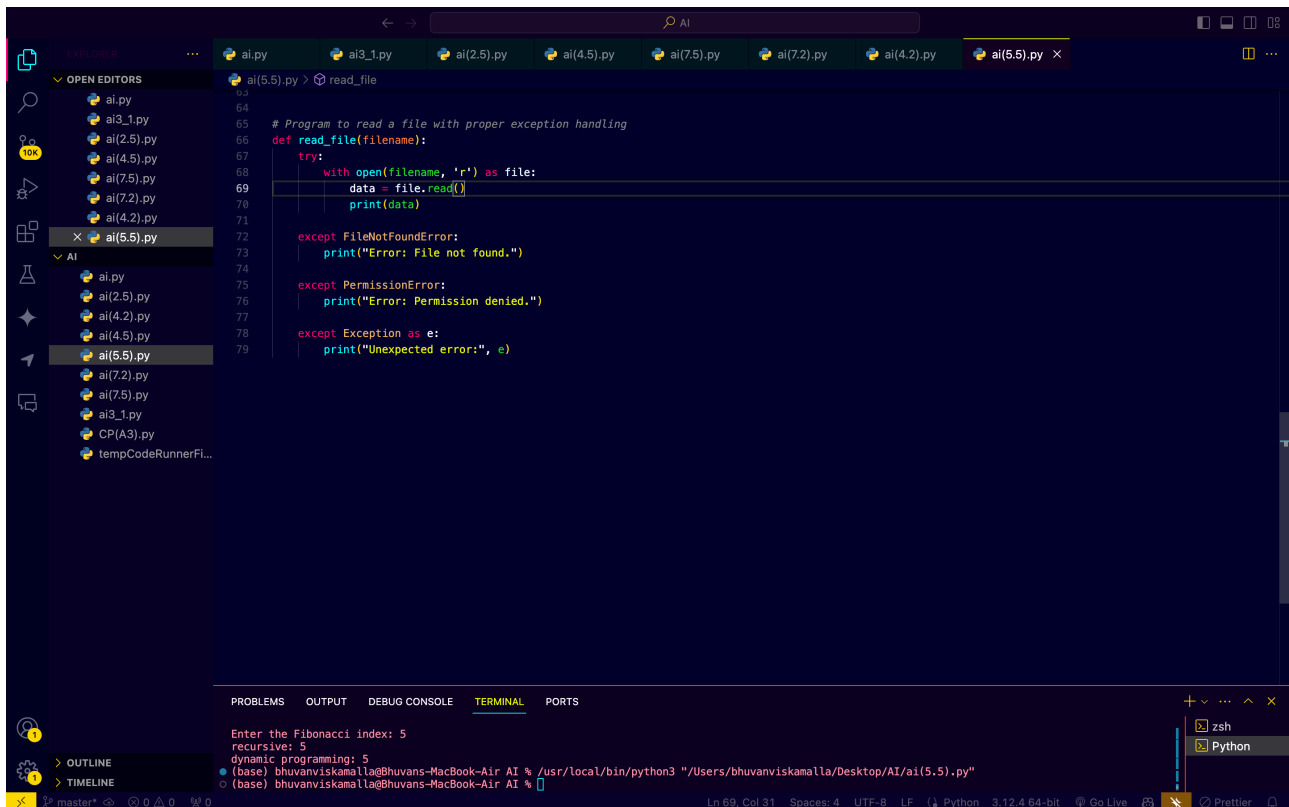
OBSERVATION:

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 optimised 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:



```
64
65 # Program to read a file with proper exception handling
66 def read_file(filename):
67     try:
68         with open(filename, 'r') as file:
69             data = file.read()
70             print(data)
71
72     except FileNotFoundError:
73         print("Error: File not found.")
74
75     except PermissionError:
76         print("Error: Permission denied.")
77
78     except Exception as e:
79         print("Unexpected error:", e)
```

Enter the Fibonacci index: 5
recursive: 5
dynamic programming: 5
(base) bhuvanviskamalla@Bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"

OBSERVATION:

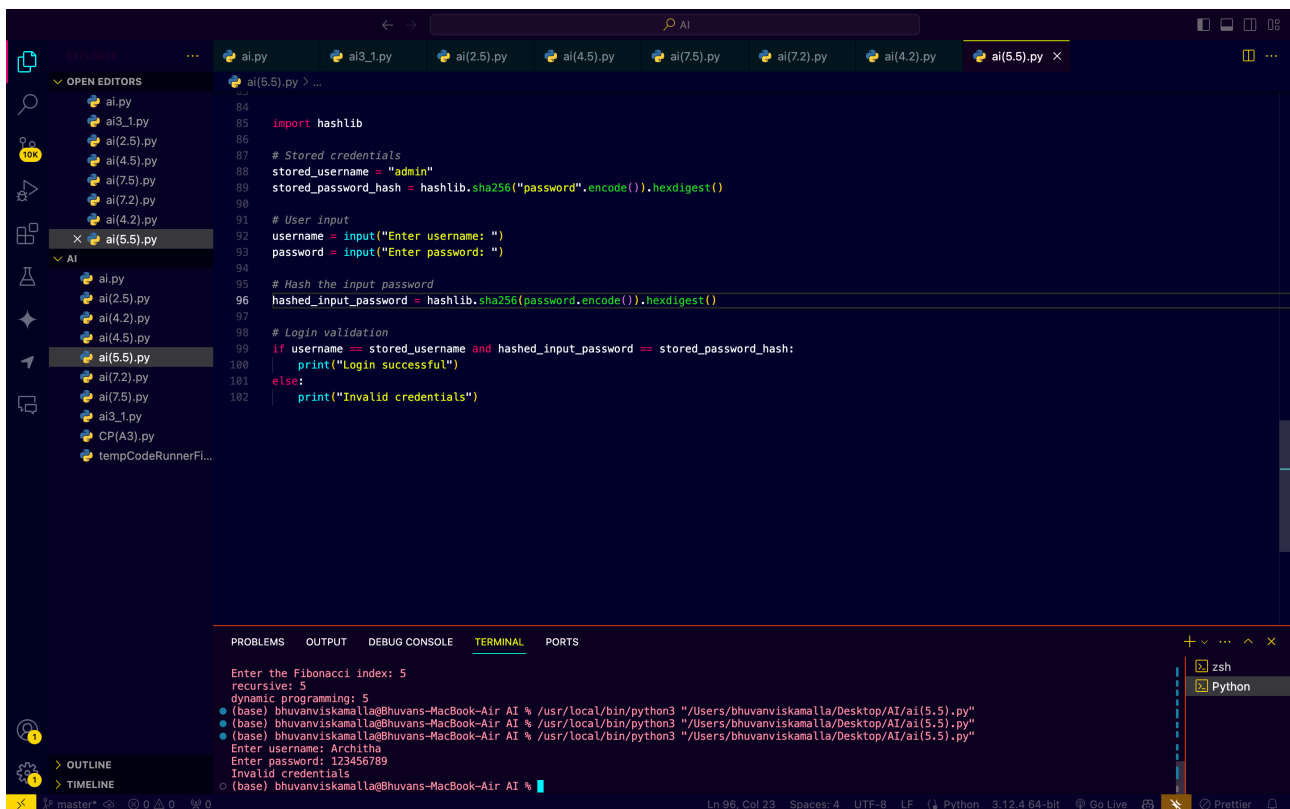
The program clearly separates different types of errors. Each exception is handled with a meaningful message. File NotFoundError explains missing file issues. PermissionError explains access-related problems. A general exception block handles unknown runtime errors. The explanations match the behaviour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

TASK-4

PROMPT:

Generate a Python-based login system.
Analyse security flaws and provide a revised secure version using password hashing and input validation.

CODE:



```
84
85 import hashlib
86
87 # Stored credentials
88 stored_username = "admin"
89 stored_password_hash = hashlib.sha256("password".encode()).hexdigest()
90
91 # User input
92 username = input("Enter username: ")
93 password = input("Enter password: ")
94
95 # Hash the input password
96 hashed_input_password = hashlib.sha256(password.encode()).hexdigest()
97
98 # Login validation
99 if username == stored_username and hashed_input_password == stored_password_hash:
100     print("Login successful")
101 else:
102     print("Invalid credentials")
```

Enter the Fibonacci index: 5
recursive: 5
dynamic programming: 5
● (base) bhuvanviskamalla@bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"
● (base) bhuvanviskamalla@bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"
● (base) bhuvanviskamalla@bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"
Enter username: Architha
Enter password: 123456789
Invalid credentials
● (base) bhuvanviskamalla@bhuvans-MacBook-Air AI %

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

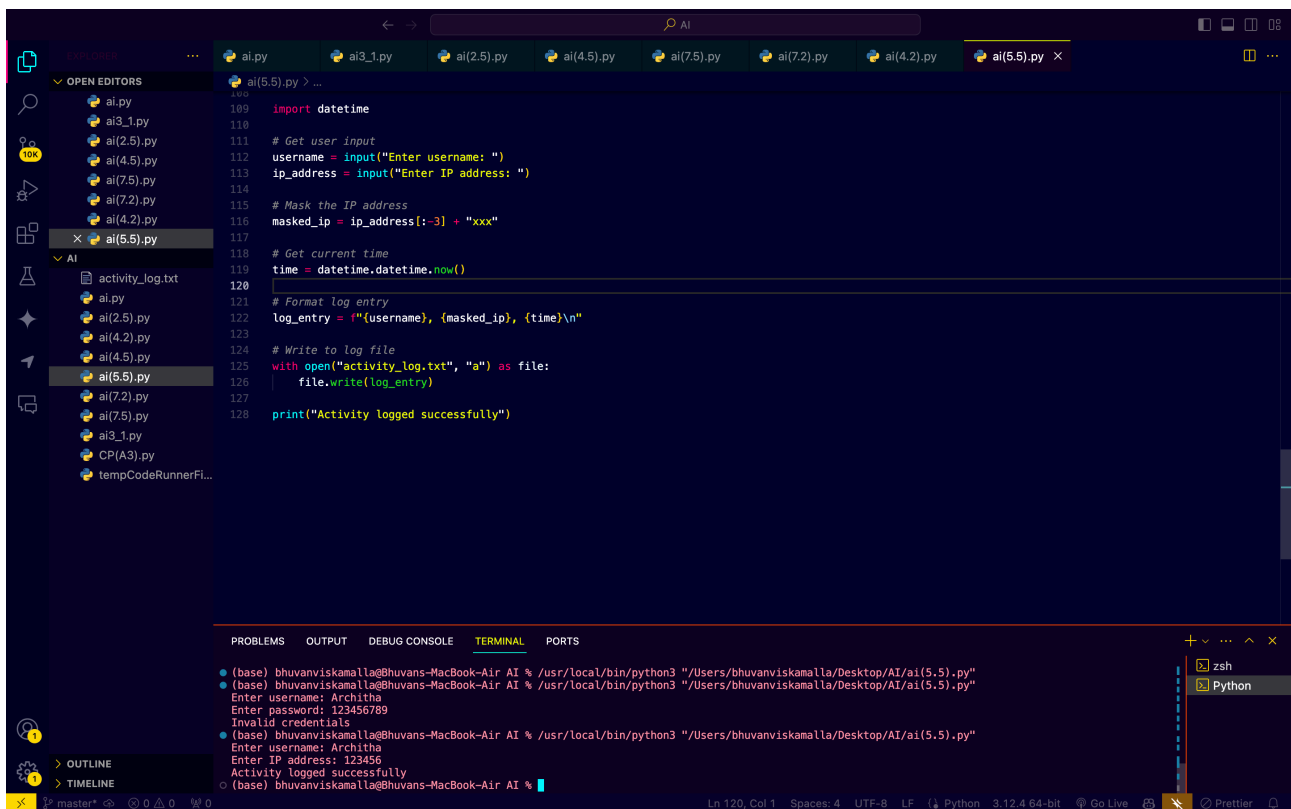
compares hashed values instead of raw passwords. This reduces the risk of password leakage. Secure authentication improves protection against attacks.

TASK-5

PROMPT:

Generate a Python script that logs user activity. Analyse privacy risks and provide an improved version using masked or minimal logging.

CODE:



```
109 import datetime
110
111 # Get user input
112 username = input("Enter username: ")
113 ip_address = input("Enter IP address: ")
114
115 # Mask the IP address
116 masked_ip = ip_address[:-3] + "xxx"
117
118 # Get current time
119 time = datetime.datetime.now()
120
121 # Format log entry
122 log_entry = f"{username}, {masked_ip}, {time}\n"
123
124 # Write to log file
125 with open("activity_log.txt", "a") as file:
126     file.write(log_entry)
127
128 print("Activity logged successfully")
```

The terminal output shows the script being executed in a shell environment. It prompts for a username and IP address, masks the IP address, and logs the activity to a file named activity_log.txt. The output is as follows:

```
(base) bhuvanviskamalla@bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"
Enter username: Architha
Enter password: 123456789
Invalid credentials
(base) bhuvanviskamalla@bhuvans-MacBook-Air AI % /usr/local/bin/python3 "/Users/bhuvanviskamalla/Desktop/AI/ai(5.5).py"
Enter username: Architha
Enter IP address: 123456
Activity logged successfully
(base) bhuvanviskamalla@bhuvans-MacBook-Air AI %
```

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.