

ASSIGNMENT-5.5

ROLL-NO:2303A51600

BATCH-29

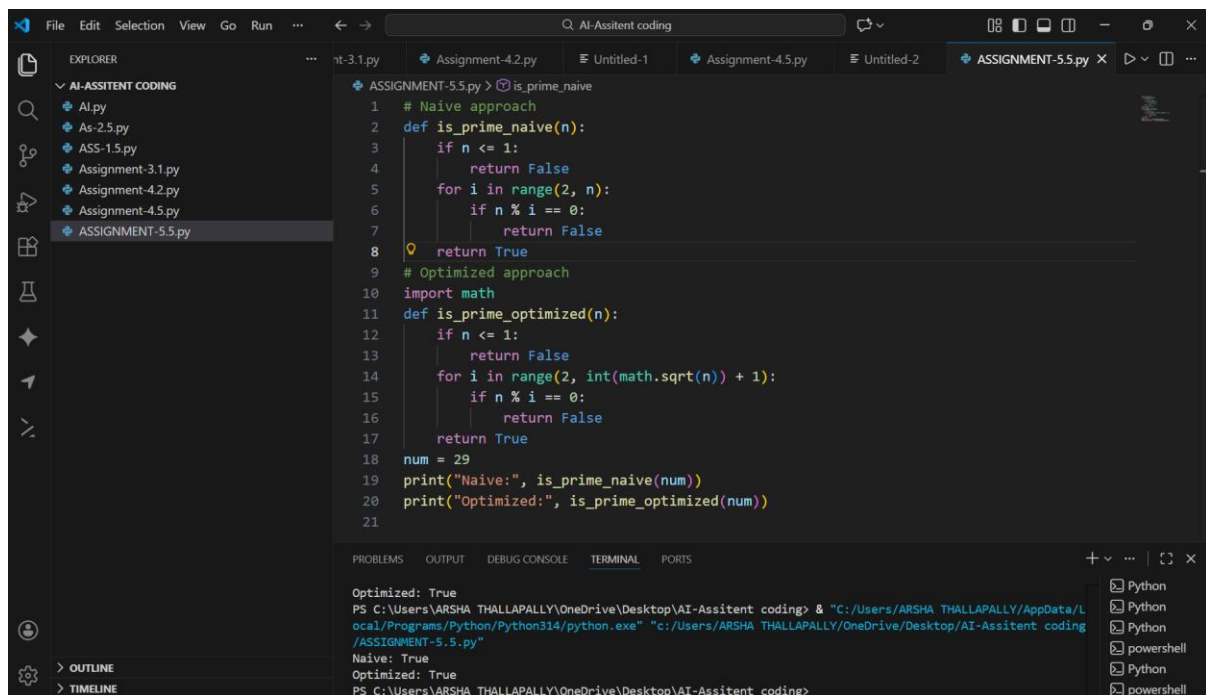
TASK-1

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

Optimized: True
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> & "C:/Users/ARSHA THALLAPALLY/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assitent coding/ASSIGNMENT-5.5.py"

Naive: True
Optimized: True
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent 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 $O(\sqrt{n})$, 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

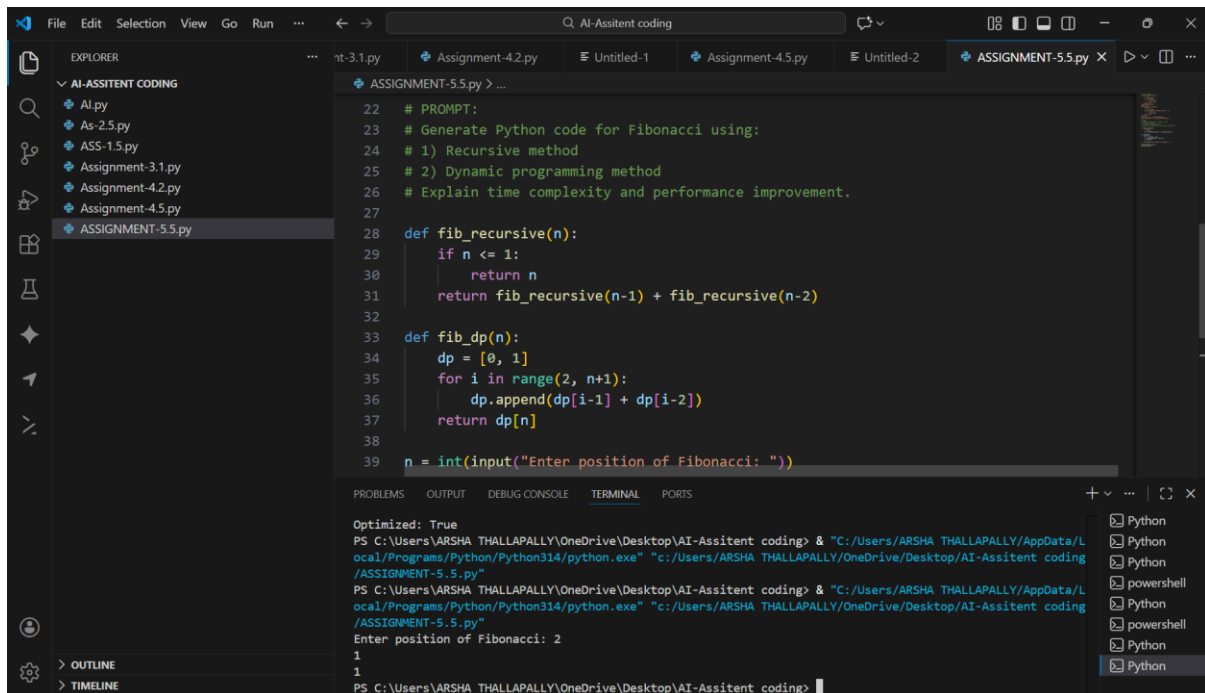
PROMPT:

Generate Python code for Fibonacci using:

- 1) Recursive method
- 2) Dynamic programming method

Explain time complexity and performance improvement.

CODE:



```
22 # PROMPT:
23 # Generate Python code for Fibonacci using:
24 # 1) Recursive method
25 # 2) Dynamic programming method
26 # Explain time complexity and performance improvement.
27
28 def fib_recursive(n):
29     if n <= 1:
30         return n
31     return fib_recursive(n-1) + fib_recursive(n-2)
32
33 def fib_dp(n):
34     dp = [0, 1]
35     for i in range(2, n+1):
36         dp.append(dp[i-1] + dp[i-2])
37     return dp[n]
38
39 n = int(input("Enter position of Fibonacci: "))
```

Optimized: True
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> "C:/Users/ARSHA THALLAPALLY/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assitent coding/ASSIGNMENT-5.5.py"
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> "C:/Users/ARSHA THALLAPALLY/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assitent coding/ASSIGNMENT-5.5.py"
Enter position of Fibonacci: 2
1
1
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent 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.

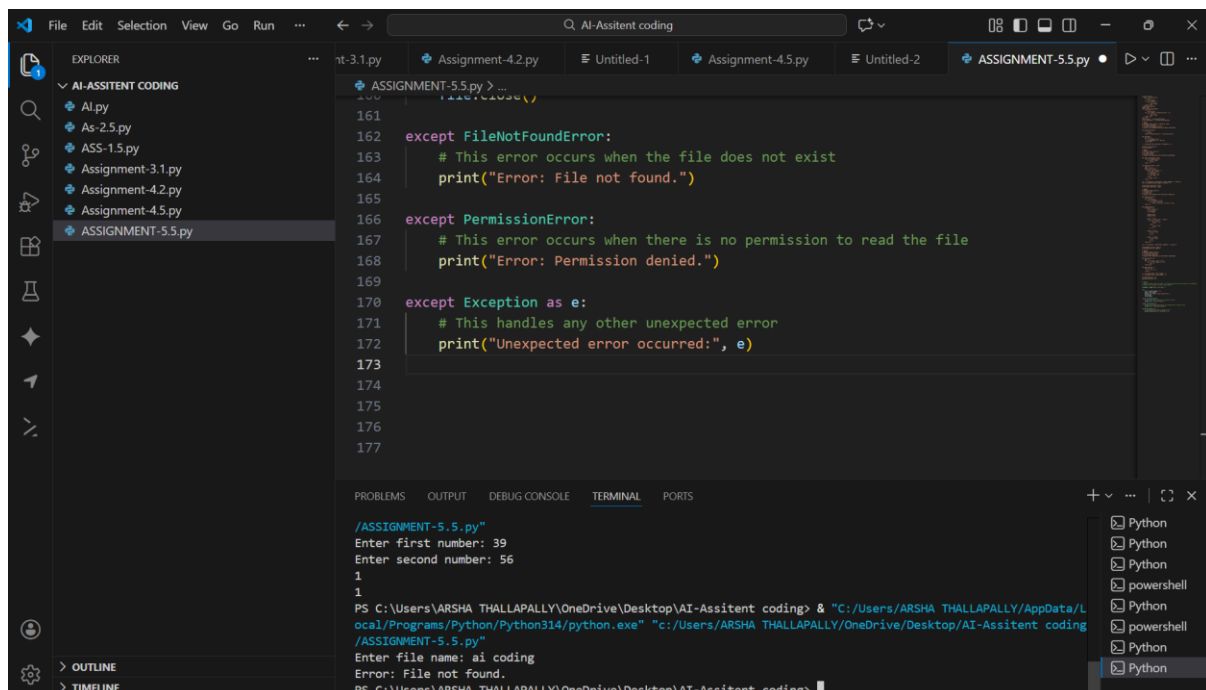
TASK-3

PROMPT:

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

Explain each exception clearly using comments.

CODE:



```
161
162 except FileNotFoundError:
163     # This error occurs when the file does not exist
164     print("Error: File not found.")
165
166 except PermissionError:
167     # This error occurs when there is no permission to read the file
168     print("Error: Permission denied.")
169
170 except Exception as e:
171     # This handles any other unexpected error
172     print("Unexpected error occurred:", e)
173
174
175
176
177
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/ASSIGNMENT-5.5.py"
Enter first number: 39
Enter second number: 56
1
1
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> "C:/Users/ARSHA THALLAPALLY/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assitent coding/ASSIGNMENT-5.5.py"
Enter file name: ai coding
Error: File not found.
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>
```

OBSERVATION:

The program clearly separates different types of errors.

Each exception is handled with a meaningful message.

FileNotFoundError explains missing file issues.

PermissionError explains access-related problems.

A general exception block handles unknown runtime errors.

The explanations match the behavior seen during execution.

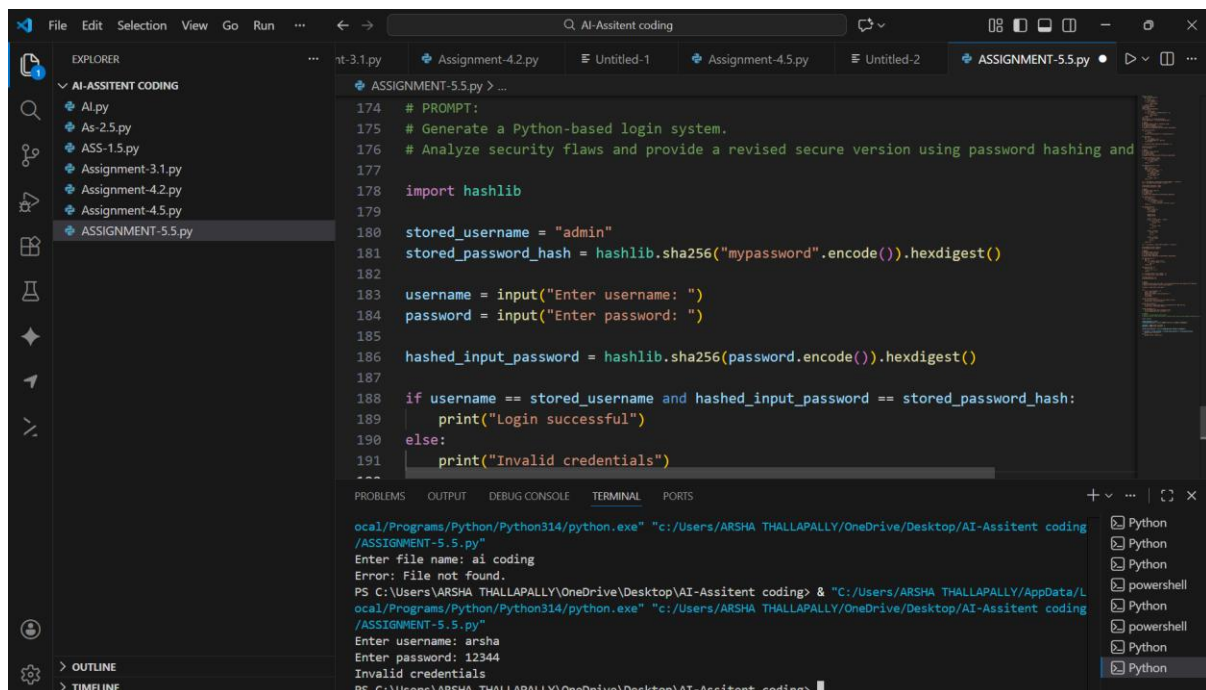
TASK-4

PROMPT:

Generate a Python-based login system.

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

CODE:



The screenshot shows a Visual Studio Code editor with a file explorer on the left containing several Python files. The main editor window displays the code for 'ASSIGNMENT-5.5.py'. The code implements a login system with a hardcoded username 'admin' and a password 'mypassword' that is hashed using SHA-256. It prompts the user for a username and password, hashes the input password, and compares it with the stored hash. The terminal at the bottom shows the execution of the script, where the user enters 'arsha' as the username and '12344' as the password, resulting in 'Invalid credentials'.

```
174 # PROMPT:
175 # Generate a Python-based login system.
176 # Analyze security flaws and provide a revised secure version using password hashing and
177
178 import hashlib
179
180 stored_username = "admin"
181 stored_password_hash = hashlib.sha256("mypassword".encode()).hexdigest()
182
183 username = input("Enter username: ")
184 password = input("Enter password: ")
185
186 hashed_input_password = hashlib.sha256(password.encode()).hexdigest()
187
188 if username == stored_username and hashed_input_password == stored_password_hash:
189     print("Login successful")
190 else:
191     print("Invalid credentials")
192
```

Terminal Output:

```
ocal/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assitent coding /ASSIGNMENT-5.5.py"
Enter file name: ai coding
Error: File not found.
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> "C:/Users/ARSHA THALLAPALLY/AppData/L ocal/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assitent coding /ASSIGNMENT-5.5.py"
Enter username: arsha
Enter password: 12344
Invalid credentials
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent 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:

The screenshot displays a Windows desktop environment. In the foreground, a Windows File Explorer window is open, showing the contents of a folder named 'AI-Assitent coding'. The folder contains several Python files: 'activity_log.txt', 'AI.py', 'As-2.5.py', 'ASS-1.5.py', 'Assignment-3.1.py', 'Assignment-4.2.py', 'Assignment-4.5.py', and 'ASSIGNMENT-5.5.py'. The 'ASSIGNMENT-5.5.py' file is selected and its contents are displayed in a code editor window.

The code in 'ASSIGNMENT-5.5.py' is a Python script that implements a simple login system. It prompts the user for a username and password, masks the password, and logs the activity to a file named 'activity_log.txt'. The script uses the 'datetime' module to record the time of the login attempt.

```

199 ip_address = input("Enter IP address: ")
200
201 masked_ip = ip_address[:3] + ".xxx.xxx"
202
203 time = datetime.datetime.now()
204
205 log_entry = f"{username}, {masked_ip}, {time}"
206
207 file = open("activity_log.txt", "a")
208 file.write(log_entry + "\n")
209 file.close()
210
211 print("Activity logged successfully.")
212
213
214
215
216
217

```

Below the code editor, the Windows Taskbar is visible, showing the Start button and several open applications: File Explorer, VS Code, and a terminal window. The terminal window is active, showing the command prompt and the output of the script execution.

The terminal output shows the following commands and results:

```

PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> "C:\Users\ARSHA THALLAPALLY\AppData\Local\Programs\Python\Python314\python.exe" "c:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding\ASSIGNMENT-5.5.py"
Enter username: arsha
Enter password: 12344
Invalid credentials
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding> "C:\Users\ARSHA THALLAPALLY\AppData\Local\Programs\Python\Python314\python.exe" "c:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding\ASSIGNMENT-5.5.py"
Enter username: aksudhg
Enter IP address: njh
Activity logged successfully.
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>

```

The background of the image is a blurred photograph of a person's face, which is partially obscured by the application windows.

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.