

ASSIGNMENT-5.5

ROLL-NO:2303A51600

BATCH-29

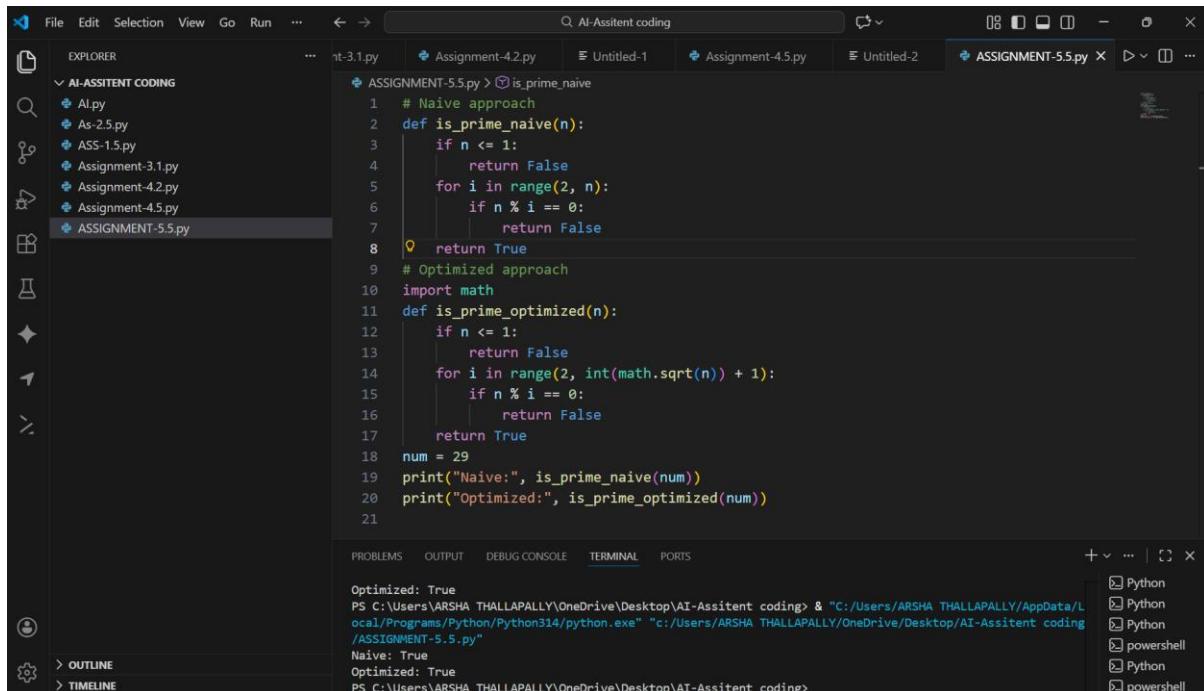
TASK-1

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:



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays a file tree with several Python files: 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 main editor area contains the following Python code:

```
# Naive approach
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

# Optimized approach
import math
def is_prime_optimized(n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

num = 29
print("Naive:", is_prime_naive(num))
print("Optimized:", is_prime_optimized(num))
```

The bottom right corner of the interface shows a terminal window with the output of the code execution:

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

The screenshot shows a Visual Studio Code (VS Code) interface. The left sidebar (EXPLORER) lists files: AI-ASSISTANT CODING, 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 right pane displays the content of ASSIGNMENT-5.5.py. The code defines two functions: fib_recursive(n) and fib_dp(n). The fib_dp function uses dynamic programming with memoization. The terminal at the bottom shows the command PS C:\... run in Python 3.14, followed by the input "Enter position of Fibonacci: 2" and the output "1". A sidebar on the right shows multiple open tabs, all labeled "Python".

```
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: "))
```

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:

The screenshot shows a Visual Studio Code interface. The Explorer sidebar on the left lists files in the 'AI-ASSISTANT CODING' folder, including '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 open in the main editor area. The code handles three types of exceptions: `FileNotFoundException`, `PermissionError`, and a general `Exception`. The terminal at the bottom shows the execution of the script and its interaction with the user. A sidebar on the right lists multiple Python environments.

```
160     print("File successfully closed")
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\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assistent coding> & "C:/Users/ARSHA THALLAPALLY/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/OneDrive/Desktop/AI-Assistent coding /ASSIGNMENT-5.5.py"
Enter file name: ai coding
Error: File not found.
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assistent coding>
```

+ ... | ☰ X

Python Python Python powershell Python powershell Python Python Python

OBSERVATION:

The program clearly separates different types of errors.
Each exception is handled with a meaningful message.
`FileNotFoundException` 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 code editor interface with the following details:

- File Explorer:** Shows a folder named "AI-ASSISTANT CODING" containing several Python files: 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 file "ASSIGNMENT-5.5.py" is currently selected.
- Code Editor:** Displays the following Python code:

```
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")
```
- Terminal:** Shows the command line output of running the script:

```
PS C:\Users\ARSHA THALLAPALLY\Desktop\AI-Assistent coding> python ASSIGNMENT-5.5.py
Enter file name: ai coding
Error: File not found.
PS C:\Users\ARSHA THALLAPALLY\Desktop\AI-Assistent coding> & "C:/Users/ARSHA THALLAPALLY/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/ARSHA THALLAPALLY/Desktop/Assignment/AI-Assistent coding/ASSIGNMENT-5.5.py"
Enter username: arsha
Enter password: 12344
Invalid credentials
PS C:\Users\ARSHA THALLAPALLY\Desktop\AI-Assistent coding>
```
- Bottom Right:** A sidebar titled "RECENT" lists several Python files.

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 shows a dark-themed instance of Visual Studio Code. In the center, the code editor displays a file named `ASSIGNMENT-5.5.py`. The code itself is as follows:

```
ASSIGNMENT-5.5.py
...
1 ip_address = input("Enter IP address: ")
2
3 masked_ip = ip_address[:3] + ".xxx.xxx"
4
5 time = datetime.datetime.now()
6
7 log_entry = f"{username}, {masked_ip}, {time}"
8
9 file = open("activity_log.txt", "a")
10 file.write(log_entry + "\n")
11 file.close()
12
13 print("Activity logged successfully.")
```

Below the code editor, the terminal window shows the execution of the script:

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

The sidebar on the left shows a folder structure under `AI-ASSISTANT CODING` containing files like `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 bottom right corner of the interface shows a list of active sessions or tasks.

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.