

ASSIGNMENT-5.5

NAME:AKSHITHA

HT NO:2303A51360

BATCH NO: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

The screenshot shows a Visual Studio Code (VS Code) interface. The title bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', 'Help' and a search bar 'ASSISTANT CODINGr'. The left sidebar has sections for 'EXPLORER', 'ASSISTANT CODINGr', 'SEARCH', '4.2.PY', '4.5.PY', and '5.5.PY' (which is selected). The main editor area contains Python code for prime number checking, comparing a naive approach with an optimized one. Below the editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'GIT LENS'. The terminal at the bottom shows command-line output for both approaches on the numbers 29 and 30. The status bar at the bottom right shows 'Ln 17, Col 2' and other system information.

```
3.1.PY 4.2.PY 4.5.PY 5.5.PY > is_prime_optimized

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

def is_prime_optimized(n):
    if n < 2:
        return False
    if n % 2 == 0 and n != 2:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

print("Naive check for 29:", is_prime_naive(29))
print("Optimized check for 29:", is_prime_optimized(29))
print("Naive check for 30:", is_prime_naive(30))
print("Optimized check for 30:", is_prime_optimized(30))

Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'.

PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr> & C:\Users\nampa\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/nampa/OneDrive/Desktop/ASSISTANT CODINGr/5.5.py"
Naive check for 29: True
Optimized check for 29: True
Naive check for 30: False
Optimized check for 30: False
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr> 2303a5a1360
```

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:

A screenshot of the Visual Studio Code (VS Code) interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled 'ASSISTANT CODINGr'. The left sidebar has icons for Explorer, Search, Problems, and others. The main workspace shows a file named '5.5.PY' with the following code:

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

The bottom status bar shows 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', 'GITLENS', and a terminal window showing command-line output:

```
-enable it, run "Import-Module PSReadLine".
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr & c:\Users\nampa\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/nampa/OneDrive/Desktop/ASSISTANT CODINGr/5.5.py"
Naive check for 29: True
Optimized check for 29: True
Naive check for 30: False
Optimized check for 30: False
Enter position of Fibonacci: 2
Fibonacci using Recursion: 1
Fibonacci using Dynamic Programming: 1
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr> 2303a51360
```

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 .

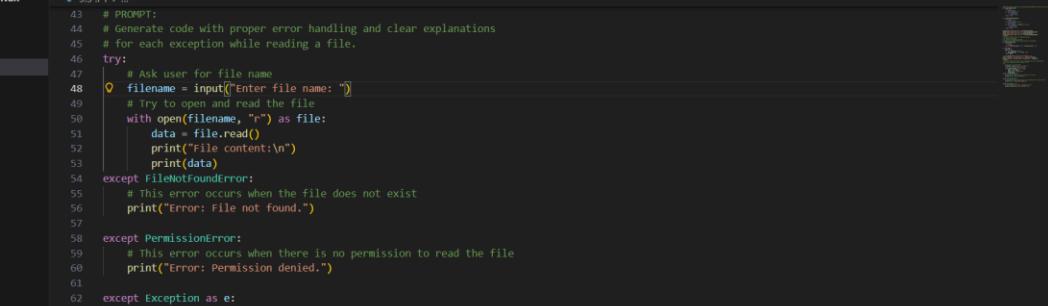
TASK-3

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 VS Code interface with the following details:

- File Explorer:** Shows files 3.1.PY, 4.2.PY, 4.5.PY, and 5.5.PY.
- Search Bar:** ASSISTANT CODING
- Code Editor:** Displays Python code for reading a file and handling exceptions. The code includes imports for os, sys, and tkinter, and uses try-except blocks to handle FileNotFoundError, PermissionError, and Exception.
- Terminal:** Shows command-line output for running the script 5.5.PY, which includes a naive check for Fibonacci numbers and an attempt to read a file named 'ai coding'.
- Bottom Status Bar:** Includes file navigation icons, status bar with 'Ln 48, Col 42', and system status icons.

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 Microsoft Visual Studio Code (VS Code) interface. The Explorer sidebar on the left lists several Python files: 3.1.PY, 4.2.PY, 4.5.PY, and 5.5.PY. The 5.5.PY file is currently open in the editor. The code implements a simple login system using the hashlib library to hash passwords. It prompts the user for a username and password, then compares the hashed input against a stored hash for 'admin'. The terminal tab at the bottom shows some test output related to Fibonacci sequence calculations and file operations.

```
66 # PROMPT:
67 # Generate a Python-based login system
68 # Analyze security flaws and provide a secure version
69 # using password hashing and input validation
70
71 import hashlib
72
73 stored_username = "admin"
74 stored_password_hash = hashlib.sha256("mypassword".encode()).hexdigest()
75
76 username = input("Enter username: ")
77 password = input("Enter password: ")
78
79 hashed_input_password = hashlib.sha256(password.encode()).hexdigest()
80
81 if username == stored_username and hashed_input_password == stored_password_hash:
82     print("Login successful")
83 else:
84     print("Invalid credentials")
```

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 Python code editor interface with the following details:

- File Explorer:** Shows files 3.1.PY, 4.2.PY, 4.5.PY, and 5.5.PY, along with an open file activity_log.txt.
- Code Editor:** Displays a Python script for logging user activity. The script prompts for a username and IP address, masks the IP address, and logs the entry to activity_log.txt. It also prints a success message.
- Terminal:** Shows command-line history and output:
 - Optimized check for 30: False
 - Enter position of Fibonacci: 2
 - Fibonacci using Recursion: 2
 - Fibonacci using Dynamic Programming: 1
 - Enter file name: ai coding
 - Error: File not found.
 - Enter username: akshuuu
 - Enter password: 2345
 - Invalid credentials
 - Enter username: akkkkuushhh
 - Enter IP address: uihi]
 - Activity logged successfully.
- Status Bar:** Shows the current working directory as PS C:\Users\Nampa\OneDrive\Desktop\ASSISTANT CODING>, line 85, column 1, and other system information.

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