

LAB ASSIGNMENT 5.5

GOUTHAM

2303A51209

04

Lab Objectives:

- To explore the ethical risks associated with AI-generated Week3 - code.
- To recognize issues related to security, bias, transparency, and copyright.
- To reflect on the responsibilities of developers when using AI tools in software development.
- To promote awareness of best practices for responsible and ethical AI coding.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Identify and avoid insecure coding patterns generated by AI tools.
- Detect and analyze potential bias or discriminatory logic in AI-generated outputs.
- Evaluate originality and licensing concerns in reused AI-generated code.
- Understand the importance of explainability and transparency in AI-assisted programming.
- Reflect on accountability and the human role in ethical AI coding practices

Task Description #1 (Transparency in Algorithm Optimization)

Task:

Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach

Prompt:

Generate Python code for two prime-checking methods:

- 1) Naive approach
- 2) Optimized approach

Also explain how the optimized version improves performance and time complexity.

Code:

```
# Naive Prime Checking Method
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 Prime Checking Method
import math

def is_prime_optimized(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    limit = int(math.sqrt(n)) + 1
    for i in range(3, limit, 2):
        if n % i == 0:
            return False

    return True

```

Explanation:

$29 > 1 \rightarrow$ continue

Loop checks divisibility from 2 to 28

No number divides 29

So, function returns True

Output confirms **29 is prime**

Time Complexity:

Naive $\rightarrow O(n)$

Optimized $\rightarrow O(\sqrt{n})$

TASK 2: Recursive Fibonacci (Transparency)

Objective: Use AI to generate a recursive function to calculate

Fibonacci numbers

Prompt:

Generate a recursive Python function to calculate Fibonacci numbers. Add clear comments explaining recursion. Explain base cases and recursive calls.

Code:

```
AI_assisted_coding_lab-5.5_1209.py > ...
1 def fibonacci(n):
2     # Base Case 1:
3     # If n is 0, return 0
4     if n == 0:
5         return 0
6
7     # Base Case 2:
8     # If n is 1, return 1
9     if n == 1:
10        return 1
11
12    # Recursive Case:
13    # Function calls itself to calculate previous two values
14    return fibonacci(n - 1) + fibonacci(n - 2)
15
16
17 # Driver code
18 num = 6
19 print("Fibonacci of", num, "is:", fibonacci(num))
20

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS TIMELINE
powershell + × └ ┌ ...

PS C:\Users\gouth\OneDrive\Desktop\3-2 SEM\Ai Assisted coding> & C:\Python314\python.exe "c:/Users/gouth/OneDrive/top/3-2 SEM/Ai Assisted coding/AI_assisted_coding_lab-5.5_1209.py"
● Fibonacci of 6 is: 8
○ PS C:\Users\gouth\OneDrive\Desktop\3-2 SEM\Ai Assisted coding>
```

Explanation:

Fibonacci Using Recursion – Short Explanation

The Fibonacci series is a sequence where each number is the **sum of the previous two numbers**.

In recursion, the function **calls itself** to solve smaller subproblems.

Base Cases

$$\text{fibonacci}(0) = 0$$

$$\text{fibonacci}(1) = 1$$

These stop the recursion.

Recursive Case

For $n > 1$,

$$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$$

The function keeps calling itself until it reaches the base cases, then returns the result step by step.

Time Complexity

$O(2^n)$ due to repeated recursive calls.

TASK 3:

Use AI to generate a Python program that reads a file and processes data.

Prompt:

Generate a Python program that reads a file and processes data. Include proper exception handling and explain each exception.

Code:

```
1  try:
2      file = open("data.txt", "r")
3      content = file.read()
4      print("File Content:")
5      print(content)
6      file.close()
7
8  except FileNotFoundError:
9      print("Error: File not found.")
10
11 except PermissionError:
12     print("Error: Permission denied while accessing the file.")
13
14 except Exception as e:
15     print("Unexpected error:", e)
```

Output:

```
PS C:\Users\gouth\OneDrive\Desktop\3-2 SEM\Ai Assisted coding> & C:\Python314\python.exe "c:/Users/gouth/OneDrive/Desktop/3-2 SEM/Ai Assisted coding/AI_assisted_coding_lab-5.5_1209.py"
Error: File not found.
PS C:\Users\gouth\OneDrive\Desktop\3-2 SEM\Ai Assisted coding>
```

Explanation:

Explanation of the Error (Point-wise)

The error occurs because 10 20 30 is typed directly in **PowerShell**.

PowerShell tries to treat these numbers as a **command or expression**.

Since there is **no operator** between the numbers, PowerShell throws an **Unexpected token** error.

PowerShell cannot read raw data like this as input.

The correct way is to **store the data in a file** and read it using a Python program.

Task4:

Use an AI tool to generate a Python-based login system.

Prompt:

Generate a simple Python-based login system using username and password.

Code:

```
import hashlib

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

stored_username = "admin"
stored_password = hash_password("Admin@123")

username = input("Enter username: ")
password = input("Enter password: ")

if username == stored_username and hash_password(password) == stored_password:
    print("Login successful")
else:
    print("Invalid credentials")
```

Output:

```
PS D:\3-2 SEM\AI ASSISTED CODING - 1221>
Enter username: admin
Enter password: Admin@123
Login successful
```

Explanation:

Explanation

The program stores a predefined username and password.

The user enters login details using input().

The if condition checks whether both username and password match.

If they match, login is successful.

Otherwise, access is denied.

Task5:

Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp)

Prompt:

Generate a Python script that logs user activity including username, IP address, and timestamp.

Code:

```
import logging

logging.basicConfig(
    filename="activity.log",
    level=logging.INFO,
    format"%(asctime)s - User logged in"
)

def log_user_activity():
    logging.info("User activity recorded")

log_user_activity()
print("Activity logged securely")
```

Output:

```
PS D:\3-2 SEM\AI ASSISTED CODING - 1221>
Activity logged securely
PS D:\3-2 SEM\AI ASSISTED CODING - 1221>
```

Explanation:

Username is partially hidden using

Password is never logged

Logs store only **minimum required information**

This prevents sensitive data leaks

