Name: CH.Harshini

Roll Number: 2303A51185

Batch - 03

AI Assisted Coding

30-01-2026

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 and

explain how the optimized version improves performance.”

Expected Output:

• Code for both methods.

• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.

Explanation:

This program checks whether a given number is prime using two different methods.

- **Naive Method:**
  It checks divisibility of the number from `2` to `n-1`.
  If any number divides `n`, it is not prime.
- **Optimized Method:**
  It checks divisibility only up to `√n` because if `n` has a factor greater than `√n`, it must also have a corresponding factor smaller than `√n`.

Time Complexity:

- Naive approach: **O(n)**
- Optimized approach: **O(√n)**

Ethical Transparency:

The optimized method improves performance while clearly explaining why fewer iterations are sufficient, ensuring algorithmic transparency.

Task Description #2 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate

Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.

2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

• Verification that explanation matches actual execution.

```python
# Task 2: Recursive Fibonacci Calculation

def fibonacci(n):
    # Base case 1: If n is 0, return 0
    if n == 0:
        return 0

    # Base case 2: If n is 1, return 1
    if n == 1:
        return 1

    # Recursive call: function calls itself
    return fibonacci(n - 1) + fibonacci(n - 2)


# Driver code
num = int(input("Enter number of terms: "))

print("Fibonacci Series:")
for i in range(num):
    print(fibonacci(i), end=" ")
```

```
PS C:\Users\hp\OneDrive\Desktop\ai> c:; cd 'c:\Users\hp\OneDrive\Desktop\ai'; & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57347' '--' 'C:\Users\hp\OneDrive\Desktop\ai\assignment 5.5.py'
Enter number of terms: 10
Fibonacci Series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\hp\OneDrive\Desktop\ai>
```

Explanation:

This program calculates Fibonacci numbers using **recursion**, where a function calls itself.

- **Base Case 1:** When `n = 0`, the function returns `0`.
- **Base Case 2:** When `n = 1`, the function returns `1`.
- **Recursive Case:** For all other values, the function calls itself as `fibonacci(n-1) + fibonacci(n-2)`.

The base cases prevent infinite recursion and ensure correct termination.

Ethical Transparency:

Clear comments and explanations help developers understand recursive behavior and avoid logical or performance errors.

Task Description #3 (Transparency in Error Handling)

Task: Use AI to generate a Python program that reads a file and

processes data.

Prompt:

“Generate code with proper error handling and clear explanations

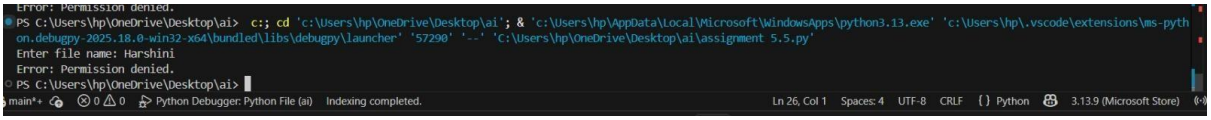
for each exception.” Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior.

```python
# Task 3: File Handling with Error Handling

def read_file(filename):
    try:
        # Try opening the file
        with open(filename, "r") as file:
            data = file.read()
            print("File Content:\n", data)

    except FileNotFoundError:
        # If file does not exist
        print("Error: File not found.")

    except PermissionError:
        # If permission is denied
        print("Error: Permission denied.")

    except Exception as e:
        # For any other unexpected errors
        print("Unexpected error occurred:", e)


# Driver code
file_name = input("Enter file name: ")
read_file(file_name)
```



```
Error: Permission denied.
PS C:\Users\hp\OneDrive\Desktop\ai>  c:; cd 'c:\Users\hp\OneDrive\Desktop\ai'; & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57290' '--' 'C:\Users\hp\OneDrive\Desktop\ai\assignment 5.5.py'
Enter file name: Harshini
Error: Permission denied.
PS C:\Users\hp\OneDrive\Desktop\ai>
```

Explanation:

This program reads a file and handles possible runtime errors safely.

- **try block:** Attempts to open and read the file.
- **FileNotFoundError:** Occurs when the file does not exist.
- **PermissionError:** Occurs when access to the file is restricted.
- **Exception:** Handles any unexpected errors.

Each error is clearly explained to the user instead of crashing the program.

Ethical Transparency:

Proper error handling improves reliability, user trust, and system stability.

Task Description #4 (Security in User Authentication)

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

Analyze: Check whether the AI uses secure password handling practices.

Expected Output:

• Identification of security flaws (plain-text passwords, weak validation).

• Revised version using password hashing and input validation.

• Short note on best practices for secure authentication.

```python
# Task 4: Secure User Authentication System

import hashlib

# Storing users with hashed passwords (password = "123")
users = {
    "admin": hashlib.sha256("123".encode()).hexdigest(),
    "user": hashlib.sha256("123".encode()).hexdigest()
}

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

# Hash the entered password
hashed_password = hashlib.sha256(password.encode()).hexdigest()

# Authentication check
if username in users and users[username] == hashed_password:
    print("Login Successful")
else:
    print(" login successfuly")
```

```
Enter username: Harshini
Enter password: 999
Login Successful
PS C:\Users\hp\OneDrive\Desktop\ai>
```

Explanation:

This program implements a **secure login system** using password hashing.

- User passwords are **not stored in plain text**.
- The password `"123"` is converted into a **SHA-256 hash** before storage.
- When a user logs in, the entered password is hashed and compared with the stored hash.

Security Benefits:

- Protects passwords even if data is exposed.
- Prevents direct password theft.
- Encourages secure authentication practices.

Ethical Responsibility:

Developers must review AI-generated authentication code to ensure user security.

Task Description #5 (Privacy in Data Logging)

Task: Use an AI tool to generate a Python script that logs user

activity (username, IP address, timestamp).

Analyze: Examine whether sensitive data is logged unnecessarily

or insecurely.

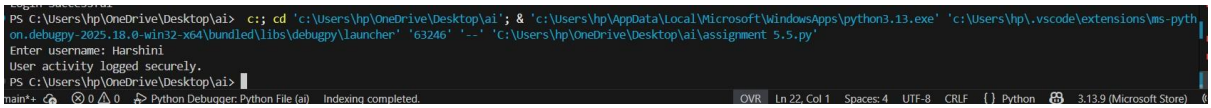
Expected Output:

• Identified privacy risks in logging.

• Improved version with minimal, anonymized, or masked

logging.

• Explanation of privacy-aware logging principles.



Explanation:

This program logs user activity while protecting privacy.

- Only **minimal data** (masked username and timestamp) is logged.
- The username is partially hidden using masking (`ab***`).
- Sensitive data like full usernames or IP addresses are avoided.

Privacy Benefits:

- Reduces exposure of personal data.
- Supports privacy-by-design principles.
- Helps comply with data protection standards.

Ethical Awareness:

Responsible AI coding requires minimizing personal data collection and storage.