

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.

```
* assignment 5.5.py > ...
1  # Task 1: Prime Number Checking
2
3  import math
4
5  # Naive approach (Basic Method)
6  def is_prime_naive(n):
7      if n <= 1:
8          return False
9
10     for i in range(2, n):
11         if n % i == 0:
12             return False
13     return True
14
15
16 # Optimized approach (Checks up to sqrt(n))
17 def is_prime_optimized(n):
18     if n <= 1:
19         return False
20
21     for i in range(2, int(math.sqrt(n)) + 1):
22         if n % i == 0:
23             return False
24     return True
25
```

Enter a number: 3  
Naive Method: True  
Optimized Method: True  
PS C:\Users\hp\OneDrive\Desktop\ai>

### 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  $\sqrt{n}$  because if  $n$  has a factor greater than  $\sqrt{n}$ , it must also have a corresponding factor smaller than  $\sqrt{n}$ .

### Time Complexity:

- Naive approach:  $O(n)$
- Optimized approach:  $O(\sqrt{n})$

### 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.

```

❶ assignment 5.5.py > ..
1 # Task 2: Recursive Fibonacci Calculation
2
3 def fibonacci(n):
4     # Base case 1: If n is 0, return 0
5     if n == 0:
6         return 0
7
8     # Base case 2: If n is 1, return 1
9     if n == 1:
10        return 1
11
12     # Recursive call: function calls itself
13     return fibonacci(n - 1) + fibonacci(n - 2)
14
15
16 # Driver code
17 num = int(input("Enter number of terms: "))
18
19 print("Fibonacci Series:")
20 for i in range(num):
21     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\python3.10.exe' 'c:\Users\hp\vscode\extensions\ms-python
on.debugger-2023.10.0-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)

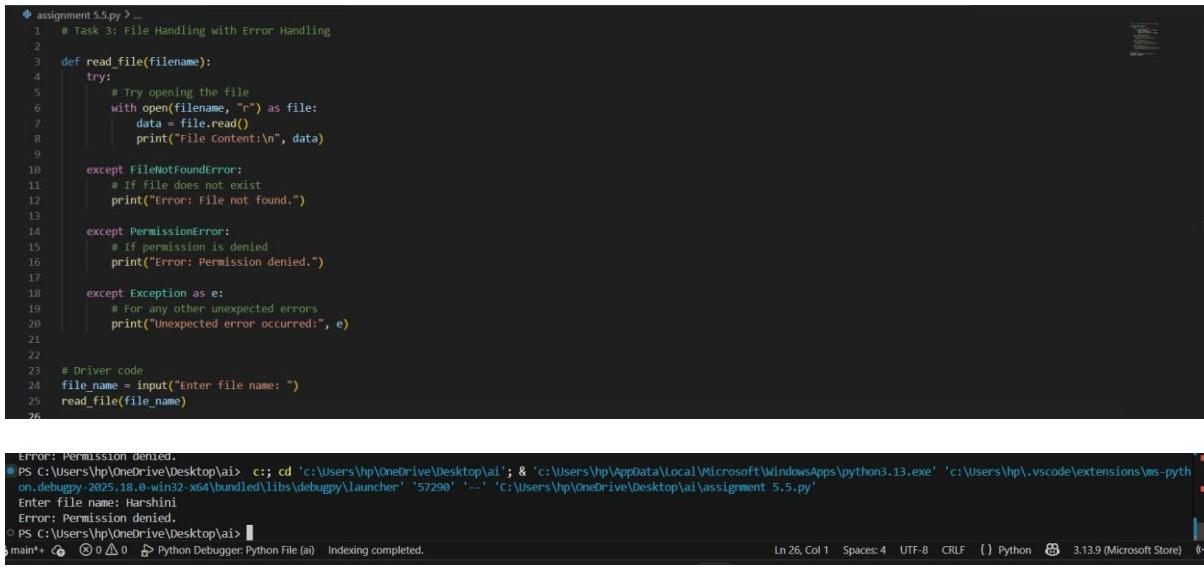
Task: Use AI to generate a Python program that reads a file and processes data.

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.



```

1  # Task 3: File Handling with Error Handling
2
3 def read_file(filename):
4     try:
5         # Try opening the file
6         with open(filename, "r") as file:
7             data = file.read()
8             print("File content:\n", data)
9
10    except FileNotFoundError:
11        # If file does not exist
12        print("Error: File not found.")
13
14    except PermissionError:
15        # If permission is denied
16        print("Error: Permission denied.")
17
18    except Exception as e:
19        # For any other unexpected errors
20        print("Unexpected error occurred:", e)
21
22
23 # Driver code
24 file_name = input("Enter file name: ")
25 read_file(file_name)

```

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-on.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>

Ln 26, Col 1 Spaces: 4 UTF-8 CRLF () Python 3.13.9 (Microsoft Store) 6:0

## Explanation:

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

- **try block:** Attempts to open and read the file.
- **FileNotFoundException:** 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)

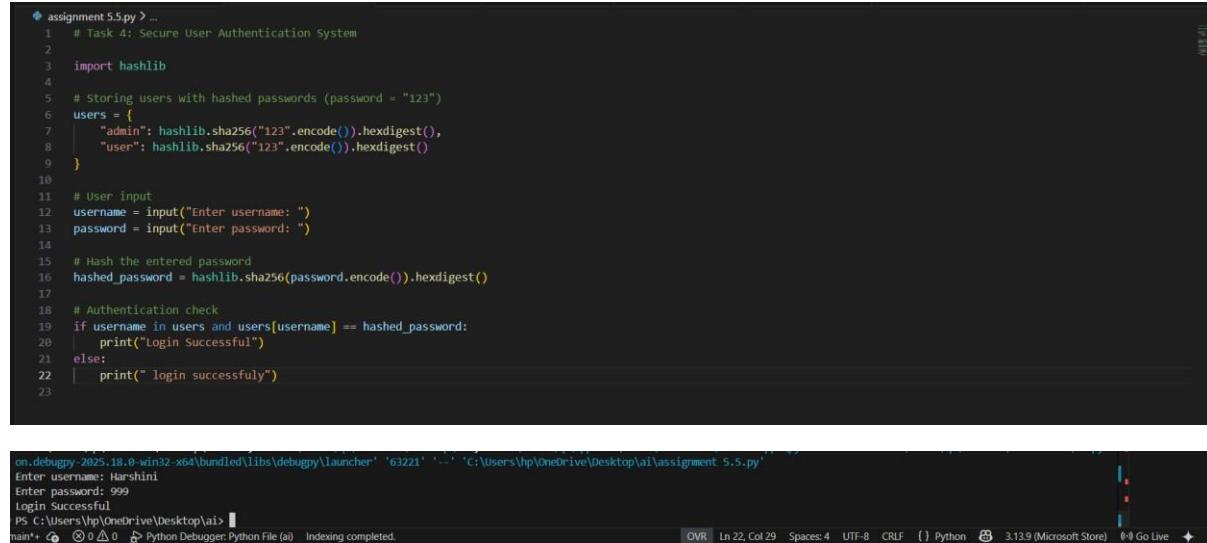
Task: Use an AI tool to generate a Python-based login system.

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.



```

assignment_5.5.py > ...
1 # Task 4: Secure User Authentication System
2
3 import hashlib
4
5 # Storing users with hashed passwords (password = "123")
6 users = {
7     "admin": hashlib.sha256("123".encode()).hexdigest(),
8     "user": hashlib.sha256("123".encode()).hexdigest()
9 }
10
11 # User input
12 username = input("Enter username: ")
13 password = input("Enter password: ")
14
15 # Hash the entered password
16 hashed_password = hashlib.sha256(password.encode()).hexdigest()
17
18 # Authentication check
19 if username in users and users[username] == hashed_password:
20     print("Login Successful")
21 else:
22     print("Login Unsuccessful")
23

```

on.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63221' '--' 'C:\Users\hp\OneDrive\Desktop\ai\assignment\_5.5.py'  
 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.

```

assignment 5.5.py > ...
1 # Task 5: Privacy-Aware Data Logging
2
3 import datetime
4
5 def log_user_activity(username):
6     # Mask username to protect privacy
7     masked_username = username[:2] + "****"
8
9     # Get current timestamp
10    timestamp = datetime.datetime.now()
11
12    # Log only minimal required data
13    with open("activity_log.txt", "a") as file:
14        file.write(f"{masked_username}, {timestamp}\n")
15
16    print("User activity logged securely.")
17
18
19 # Driver code
20 user = input("Enter username: ")
21 log_user_activity(user)
22

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.on.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63246' '--' 'c:\Users\hp\OneDrive\Desktop\ai\assignment 5.5.py'
Enter username: Harshini
User activity logged securely.
PS C:\Users\hp\OneDrive\Desktop\ai>

```

The terminal shows the execution of the Python script `assignment 5.5.py`. It prompts for a username ('Enter username:'), receives 'Harshini' as input, and logs the activity ('User activity logged securely.') to a file named 'activity\_log.txt'. The code uses a partial mask for the username and logs only the masked username and timestamp.

## 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.