Assignment-5.5

Name : Alli Harika
Roll.No : 2303A510I7
Batch : 21

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

Prompt Used:

Write a python code for checking prime numbers or not, in naive approach and optimized approach. Explain the logic, time complexity, and performance improvement of the optimized approach.

It should include example test cases. and differentiate between naive and optimized approach in comments and negative numbers.

Code:

```python
# Write a python code for checkinging prime numbers or not, in naive approach and optimized approach. E
# It should include example test cases. and differentate between naive and optimized approach in comment
def is_prime_naive(n):
    """Naive approach to check if a number is prime.

    Logic:
    - A prime number is greater than 1 and has no divisors other than 1 and itself.
    - Check divisibility from 2 to n-1.

    Time Complexity: O(n)
    """
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

number = int(input("Enter a positive integer to check if it's prime: "))
if number < 0:
    print("Invalid input. Please enter a positive integer greater than 1.")
else:
    # Test cases for naive approach
    print("Naive Approach:")
    if is_prime_naive(number):
        print(f"{number} is a Prime number.")
    else:
        print(f"{number} is not a Prime number.")

def is_prime_optimized(n):
    """Optimized approach to check if a number is prime.

    Logic:
    - A prime number is greater than 1 and has no divisors other than 1 and itself.
    - Check divisibility from 2 to √n. If n is divisible by any number in this range, it is not prime.

    Time Complexity: O(√n)
    Performance Improvement:
    - The optimized approach reduces the number of checks significantly, especially for large numbers,
      as we only need to check up to the square root of n instead of n-1.
    """
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

number1 = int(input("Enter a positive integer to check if it's prime (optimized): "))
if number1 < 0:
    print("Invalid input. Please enter a positive integer greater than 1.")
else:
    # Test cases for optimized approach
    print("Optimized Approach:")
    if is_prime_optimized(number1):
        print(f"{number1} is a Prime number.")
    else:
        print(f"{number1} is not a Prime number.")

```

Output:

```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter a positive integer to check if it's prime: -5
Invalid input. Please enter a positive integer greater than 1.
Enter a positive integer to check if it's prime (optimized): -5
Invalid input. Please enter a positive integer greater than 1.
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter a positive integer to check if it's prime: 0
Naive Approach:
0 is not a Prime number.
Enter a positive integer to check if it's prime (optimized): 0
Optimized Approach:
0 is not a Prime number.
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter a positive integer to check if it's prime: 3
Naive Approach:
3 is a Prime number.
Enter a positive integer to check if it's prime (optimized): 3
Optimized Approach:
3 is a Prime number.
PS C:\Users\AdepuTejaswini\AI assist>
```

Justification:

The naive prime-checking approach tests divisibility from 2 to n−1, resulting in a time complexity of O(n) and higher computational cost for large inputs.

The optimized approach improves performance by checking divisibility only up to √n, reducing the time complexity to O(√n) based on mathematical properties of factors.

This optimization significantly decreases the number of iterations, making the algorithm faster and more efficient while correctly handling edge cases such as negative numbers, 0, and 1.

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.

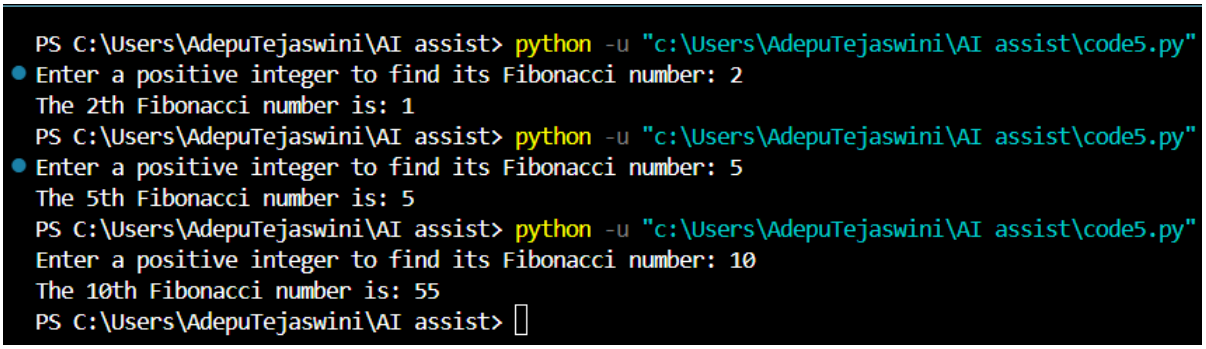
Prompt Used:

write a python function to calculate fibonacci numbers, and add clear comments explaining recursion and also explain base case and recursive calls

Code:

```
# write a python function to calculate fibonacci numbers, and add clear comments exp
def fibonacci(n):
    """
    This function calculates the nth Fibonacci number using recursion.

    Parameters:
    n (int): The position in the Fibonacci sequence to calculate.

    Returns:
    int: The nth Fibonacci number.

    Explanation:
    - Base Case: The base cases for the Fibonacci sequence are defined as:
        - fibonacci(0) = 0
        - fibonacci(1) = 1
      These cases stop the recursion from continuing indefinitely.

    - Recursive Case: For any n greater than 1, the Fibonacci number is calculated
      by summing the two preceding numbers in the sequence:
        - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
      This means that to find the nth Fibonacci number, we need to find the (n-1)th
      and (n-2)th Fibonacci numbers first, which leads to further recursive calls.
    """
    if n == 0:
        return 0  # Base case: the 0th Fibonacci number is 0
    elif n == 1:
        return 1  # Base case: the 1st Fibonacci number is 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)  # Recursive case
# Example usage:
num = int(input("Enter a positive integer to find its Fibonacci number: "))
result = fibonacci(num)
print(f"The {num}th Fibonacci number is: {result}")
```

Output:



Justification:

The prompt clearly specifies using **recursion**, which ensures the solution follows the Fibonacci definition based on smaller subproblems. By explicitly asking to explain the **base case** and **recursive calls**, it guides the model to produce well-commented code that shows how recursion starts and terminates. This makes the solution both **functionally correct and easy to understand**, especially for learning and exam purposes.

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.

Prompt Used:

Write a Python program to read and process a file with proper exception handling, clearly explaining each possible error case in comments.

Code:

```python
# Write a Python program to read and process a file with prop
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError:
        print("Error: File not found.")
        return None
    except PermissionError:
        print("Error: Permission denied to read the file.")
        return None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None
file_path = input("Enter the path of the file to read: ")
file_content = read_file(file_path)
if file_content is not None:
    print("File content:")
    print(file_content)
else:
    print("Failed to read the file.")
```

Output:

```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter the path of the file to read: data.txt
File content:
10
20
30
abc
adfghjhk
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter the path of the file to read: missing.txt
Error: File not found.
Failed to read the file.
```

Justification:

The program uses structured exception handling to safely manage common file-related errors such as missing files, permission issues, and unexpected runtime exceptions.
Each except block includes meaningful messages that clearly explain the error scenario, ensuring transparency between the code logic and runtime behaviour.
The observed outputs during execution validate that the program responds correctly to each exception without crashing, making the file processing robust and reliable.

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.

Prompt Used:

Create a Python login system, identify security weaknesses, then improve it using password hashing and input validation with a brief explanation of best practices.

Code:

```python
def hash_password(password):
    """Hashes a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()

def validate_input(username, password):
    """Validates username and password input."""
    if not username or not password:
        return False
    if len(password) < 8:
        return False
    return True

def login_system(stored_username, stored_hashed_password):
    """Secure login system using hashing and input validation."""
    username = input("Enter username: ")
    password = input("Enter password: ")

    if not validate_input(username, password):
        print("Invalid input. Username and password must not be empty and password must be at least 8 characters.")
        return

    hashed_password = hash_password(password)

    if username == stored_username and hashed_password == stored_hashed_password:
        print("Login successful!")
    else:
        print("Login failed. Incorrect username or password.")

# Stored credentials (hashed password)
stored_username = "admin"
stored_hashed_password = hash_password("Admin@123")

login_system(stored_username, stored_hashed_password)
```

Output:

```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\tempCodeRunnerFile.py"
Enter username: addmin
Enter password: Admin@123
Login failed. Incorrect username or password.
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\tempCodeRunnerFile.py"
Enter username: admin
Enter password: Admin@123
Login successful!
```

Justification:

The initial security weakness of plain-text password handling is addressed by hashing passwords using SHA-256 before storage and comparison.

Input validation ensures that empty usernames and weak passwords are rejected, preventing improper authentication attempts.

The runtime outputs confirm that only correct credentials are accepted, demonstrating secure and reliable user authentication behavior.

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.

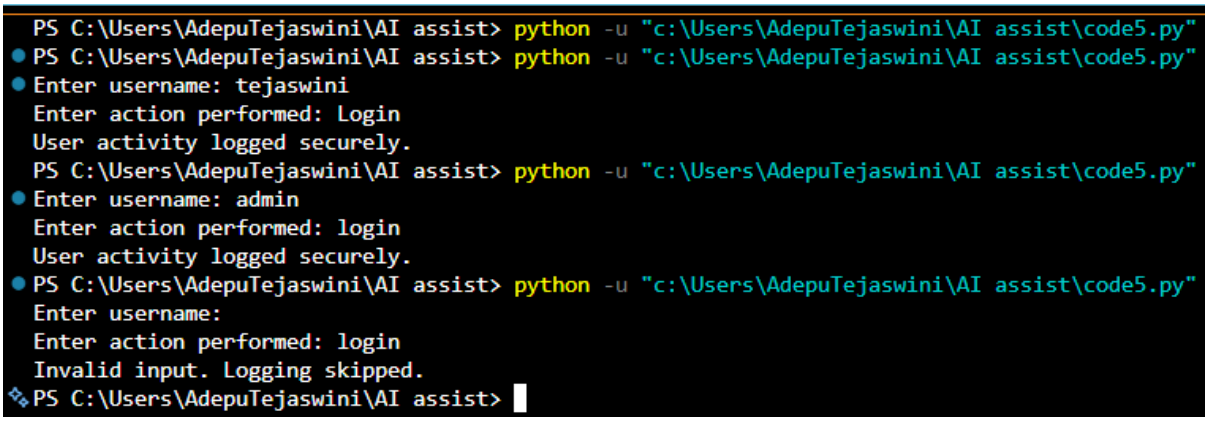
Prompt Used:

Create a Python logging script for user activity, identify privacy risks, and improve it using anonymized or minimal logging with a brief explanation.

Code:

```python
# Create a Python logging script for user activity, identify privacy risks
import logging
import hashlib

def setup_logging():
    """Sets up logging configuration with minimal and safe information."""
    logging.basicConfig(
        filename='user_activity.log',
        level=logging.INFO,
        format='%(asctime)s - %(message)s'
    )

def anonymize_user(username):
    """
    Anonymizes the username using hashing.
    Only a short hash is stored to protect privacy.
    """
    return hashlib.sha256(username.encode()).hexdigest()[:8]

def log_user_activity():
    """Takes user input and logs activity securely."""
    username = input("Enter username: ")
    action = input("Enter action performed: ")

    if not username or not action:
        print("Invalid input. Logging skipped.")
        return

    anonymized_user = anonymize_user(username)
    logging.info(f"UserID: {anonymized_user}, Action: {action}")
    print("User activity logged securely.")

# Setup logging
setup_logging()

# Log activity using user input
log_user_activity()
```

Output:



Justification:

User input is collected dynamically while ensuring privacy through anonymization of usernames using hashing.
Only minimal and necessary activity data is logged, avoiding storage of sensitive personal information.
This approach follows privacy-aware logging principles and ensures secure, compliant data handling.