AI Assistant Coding

Assignment 5

Name : Sahastra HT. No : 2303A52499 **Batch:** 50

Lab 5: Ethical Foundations – Responsible AI Coding Practices

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

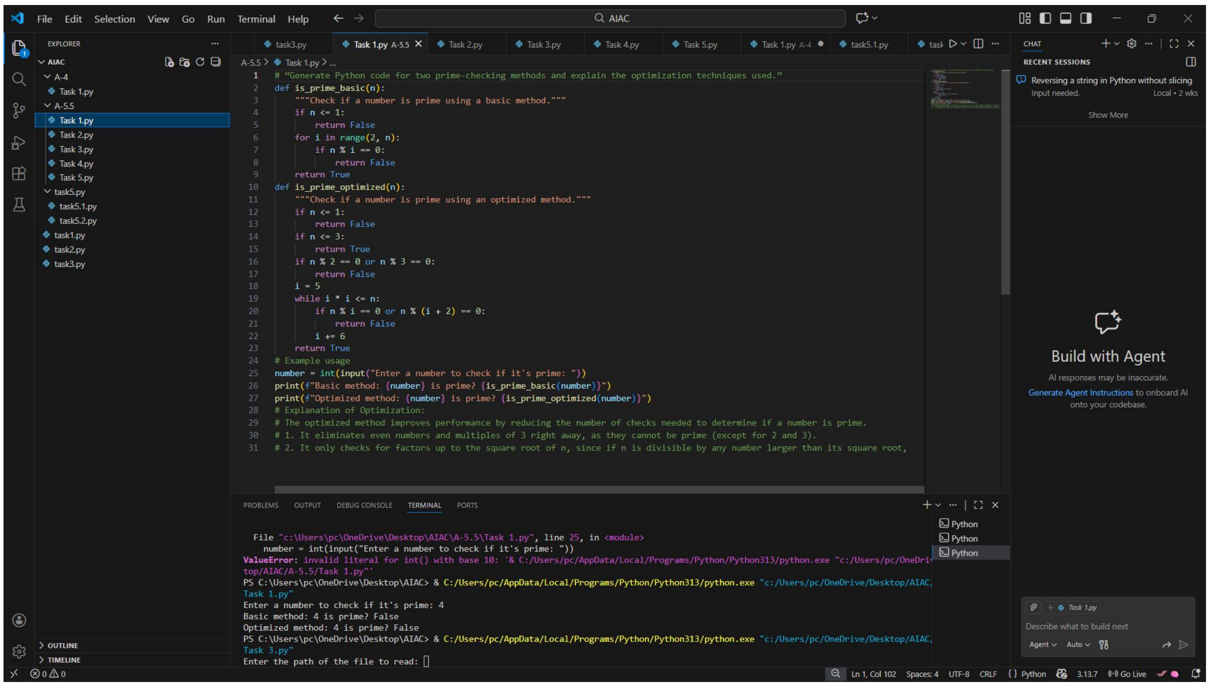
coding practices.

Task Description #1 (Transparency in Algorithm Optimization)

“Generate Python code for two prime-checking methods and

explain how the optimized version improves performance.”

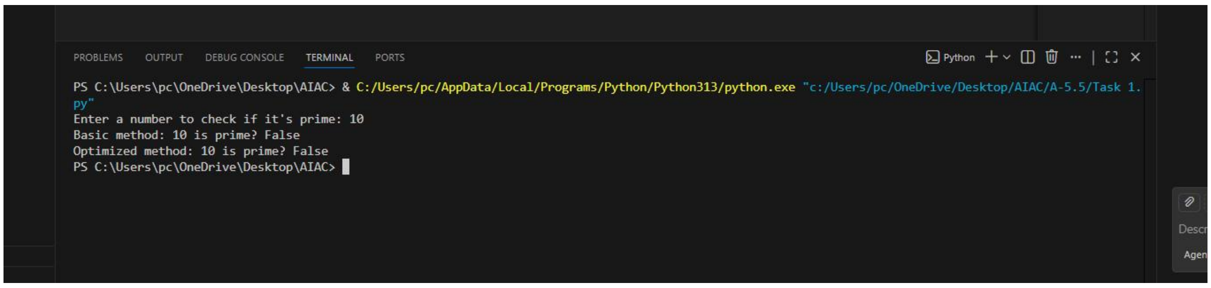
#CODE



```python
# “Generate Python code for two prime-checking methods and explain the optimization techniques used.”
def is_prime_basic(n):
    """Check if a number is prime using a basic method."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """Check if a number is prime using an optimized method."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
# Example usage
number = int(input("Enter a number to check if it's prime: "))
print(f"Basic method: {number} is prime? {is_prime_basic(number)}")
print(f"Optimized method: {number} is prime? {is_prime_optimized(number)}")
# Explanation of Optimization:
# The optimized method improves performance by reducing the number of checks needed to determine if a number is prime.
# 1. It eliminates even numbers and multiples of 3 right away, as they cannot be prime (except for 2 and 3).
# 2. It only checks for factors up to the square root of n, since if n is divisible by any number larger than its square root,
```

```
  File "c:\Users\pc\OneDrive\Desktop\AIAC\A-5.5\Task 1.py", line 25, in <module>
    number = int(input("Enter a number to check if it's prime: "))
ValueError: invalid literal for int() with base 10: '& C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDri
top/AIAC/A-5.5/Task 1.py"'
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC,
Task 1.py"
Enter a number to check if it's prime: 4
Basic method: 4 is prime? False
Optimized method: 4 is prime? False
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC,
Task 3.py"
Enter the path of the file to read: 
```

#OUTPUT



```
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC/A-5.5/Task 1.
py"
Enter a number to check if it's prime: 10
Basic method: 10 is prime? False
Optimized method: 10 is prime? False
PS C:\Users\pc\OneDrive\Desktop\AIAC> 
```

Explanation :

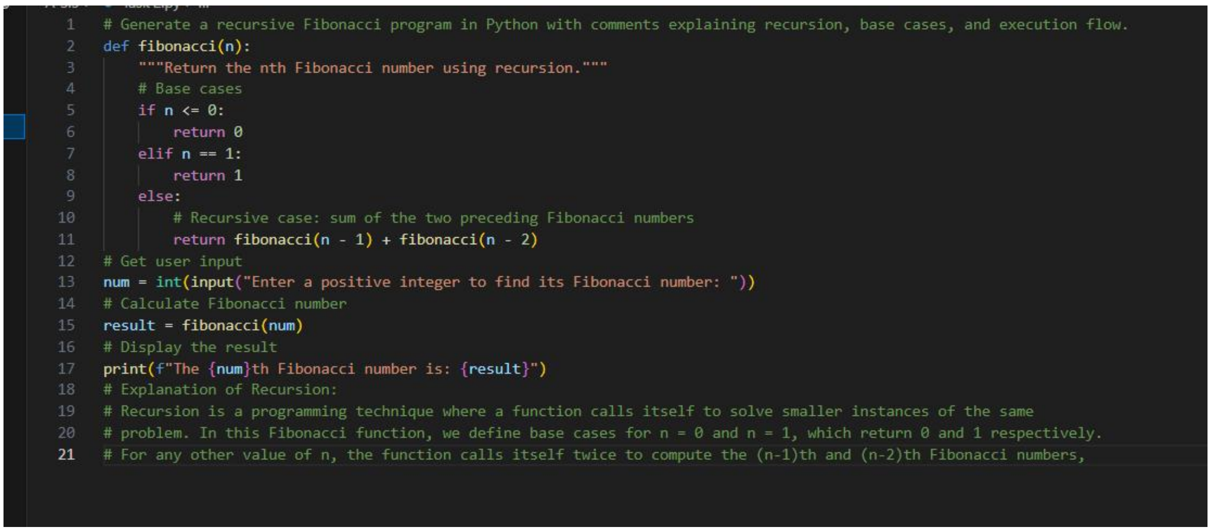
This program checks whether a number is prime using two different methods. The basic method checks if the number is divisible by any number from 2 up to the number minus one; if it finds a divisor, the number is not prime. This method is easy to understand but becomes slow for large numbers because it checks many values. The optimized method improves performance by first removing simple cases and then checking divisibility only up to the square root of the number, skipping unnecessary checks. Because it performs fewer calculations, the optimized method works faster, especially for large numbers.

Task 2: Task Description #2 (Transparency in Recursive Algorithms)

#PROMPT

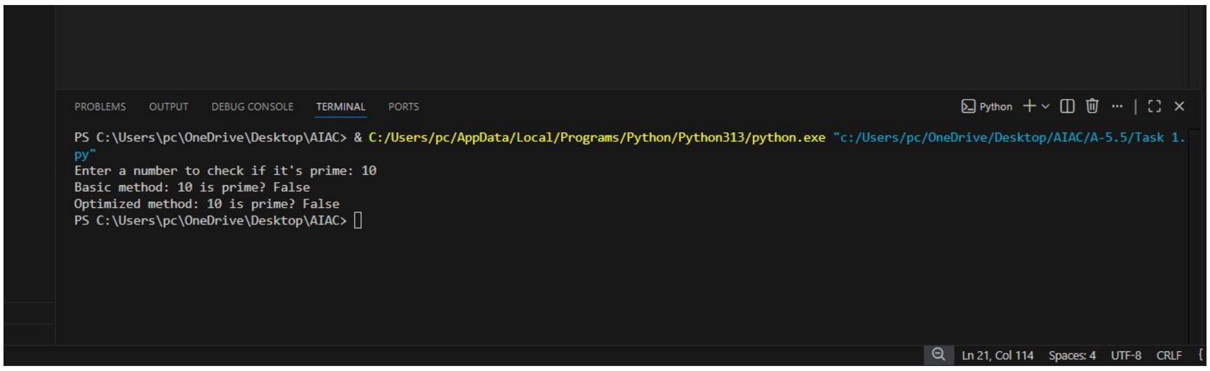
Generate a recursive Fibonacci program in Python with comments explaining recursion, base cases, and execution flow.

#CODE



```python
# Generate a recursive Fibonacci program in Python with comments explaining recursion, base cases, and execution flow.
def fibonacci(n):
    """Return the nth Fibonacci number using recursion."""
    # Base cases
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive case: sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Get user input
num = int(input("Enter a positive integer to find its Fibonacci number: "))
# Calculate Fibonacci number
result = fibonacci(num)
# Display the result
print(f"The {num}th Fibonacci number is: {result}")
# Explanation of Recursion:
# Recursion is a programming technique where a function calls itself to solve smaller instances of the same
# problem. In this Fibonacci function, we define base cases for n = 0 and n = 1, which return 0 and 1 respectively.
# For any other value of n, the function calls itself twice to compute the (n-1)th and (n-2)th Fibonacci numbers,
```

#OUTPUT



```
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC/A-5.5/Task 1.
py"
Enter a number to check if it's prime: 10
Basic method: 10 is prime? False
Optimized method: 10 is prime? False
PS C:\Users\pc\OneDrive\Desktop\AIAC> 
```

Code Explanation

This program calculates the Fibonacci number using recursion, where a function calls itself to solve smaller parts of the problem. The function first checks the base cases: if the input number is 0, it returns 0, and if it is 1, it returns 1. For any number greater than 1, the function calculates the result by adding the previous two Fibonacci numbers, which it finds by calling itself with smaller values. The program takes input from the user, computes the Fibonacci number using this recursive logic, and then prints the result. This approach clearly shows how recursion works by breaking the problem into smaller repeated calls until the base cases are reached.

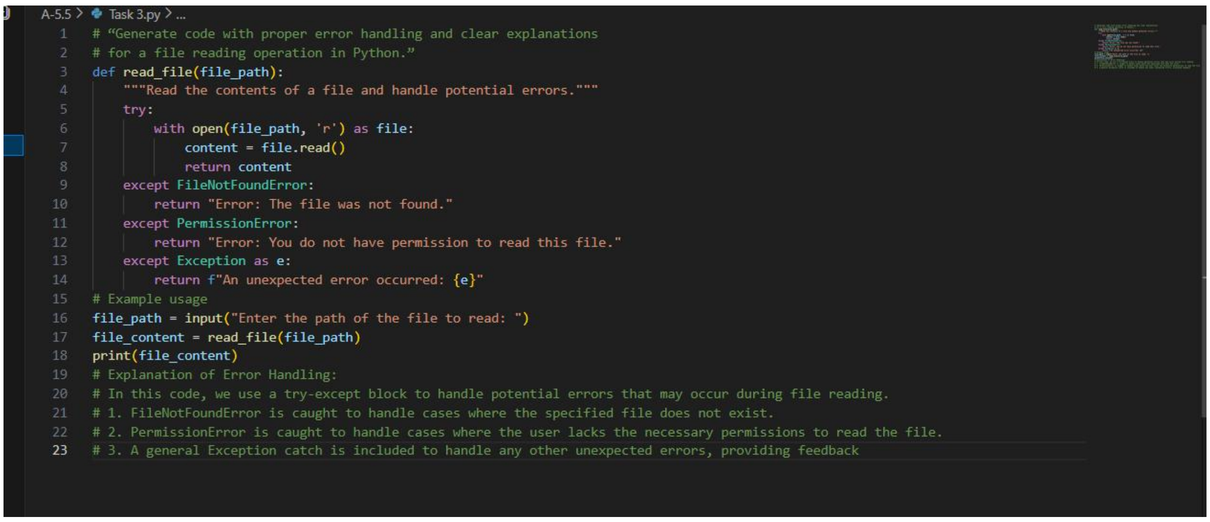
Task Description #3 (Transparency in Error Handling)

#PROMPT

“Generate code with proper error handling and clear explanations

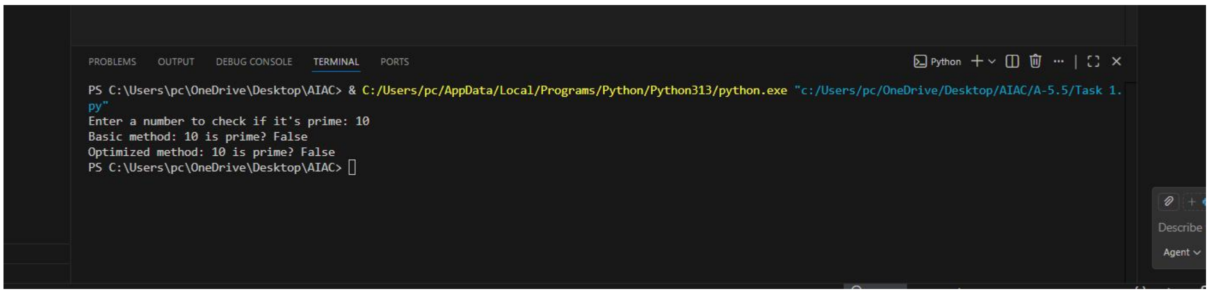
for each exception.”

#CODE



```python
# “Generate code with proper error handling and clear explanations
# for a file reading operation in Python.”
def read_file(file_path):
    """Read the contents of a file and handle potential errors."""
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError:
        return "Error: The file was not found."
    except PermissionError:
        return "Error: You do not have permission to read this file."
    except Exception as e:
        return f"An unexpected error occurred: {e}"
# Example usage
file_path = input("Enter the path of the file to read: ")
file_content = read_file(file_path)
print(file_content)
# Explanation of Error Handling:
# In this code, we use a try-except block to handle potential errors that may occur during file reading.
# 1. FileNotFoundError is caught to handle cases where the specified file does not exist.
# 2. PermissionError is caught to handle cases where the user lacks the necessary permissions to read the file.
# 3. A general Exception catch is included to handle any other unexpected errors, providing feedback
```

#OUTPUT



```
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC/A-5.5/Task 1.
py"
Enter a number to check if it's prime: 10
Basic method: 10 is prime? False
Optimized method: 10 is prime? False
PS C:\Users\pc\OneDrive\Desktop\AIAC> 
```

Explanation :

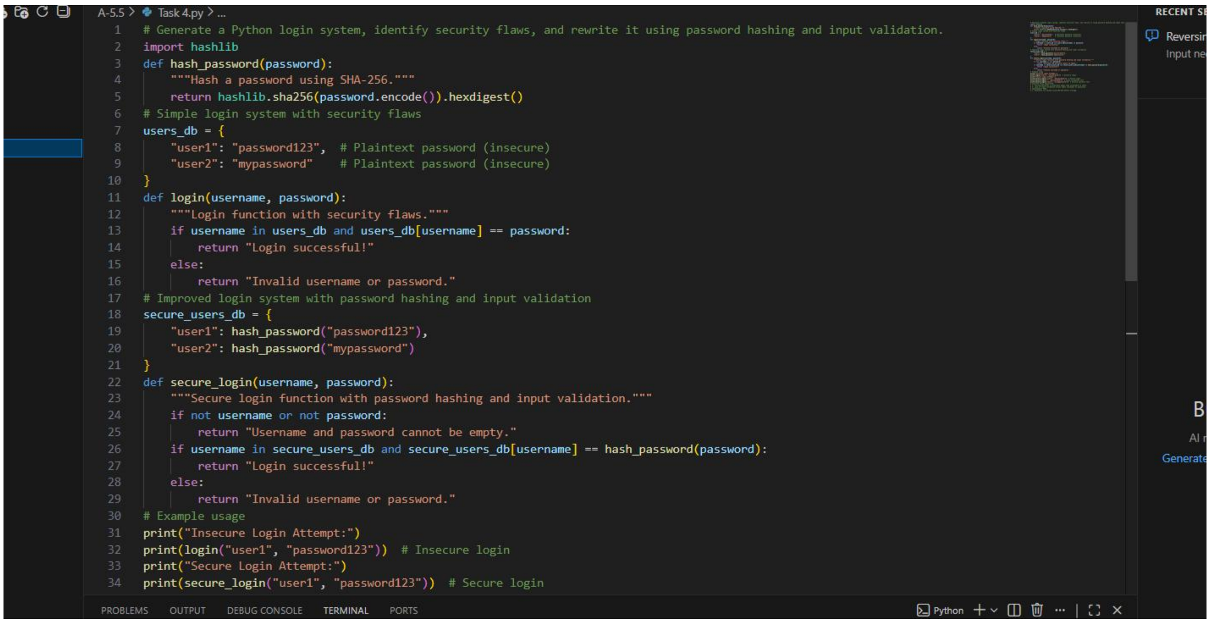
This program reads the contents of a file and safely handles possible errors that may occur. The function tries to open and read the file using the given file path. If the file does not exist, it shows a message saying the file was not found. If the user does not have permission to read the file, it displays a permission error message. Any other unexpected problem is also caught and shown as an error message. Finally, the program asks the user for the file path and prints the file's content or the error message, making the program safer and easier to use.

Task 4: (Security in User Authentication)

#PROMPT

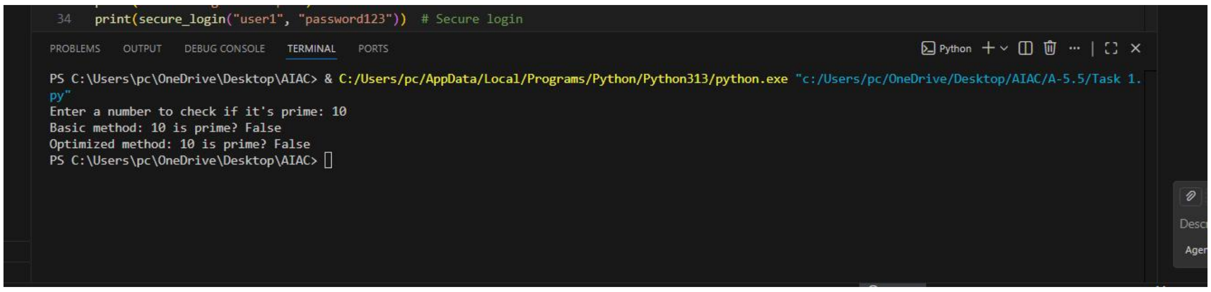
Generate a Python login system, identify security flaws, and rewrite it using password hashing and input validation.

#CODE



```python
# Generate a Python login system, identify security flaws, and rewrite it using password hashing and input validation.
import hashlib
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()
# Simple login system with security flaws
users_db = {
    "user1": "password123",  # Plaintext password (insecure)
    "user2": "mypassword"    # Plaintext password (insecure)
}
def login(username, password):
    """Login function with security flaws."""
    if username in users_db and users_db[username] == password:
        return "Login successful!"
    else:
        return "Invalid username or password."
# Improved login system with password hashing and input validation
secure_users_db = {
    "user1": hash_password("password123"),
    "user2": hash_password("mypassword")
}
def secure_login(username, password):
    """Secure login function with password hashing and input validation."""
    if not username or not password:
        return "Username and password cannot be empty."
    if username in secure_users_db and secure_users_db[username] == hash_password(password):
        return "Login successful!"
    else:
        return "Invalid username or password."
# Example usage
print("Insecure Login Attempt:")
print(login("user1", "password123"))  # Insecure login
print("Secure Login Attempt:")
print(secure_login("user1", "password123"))  # Secure login
```

#OUTPUT



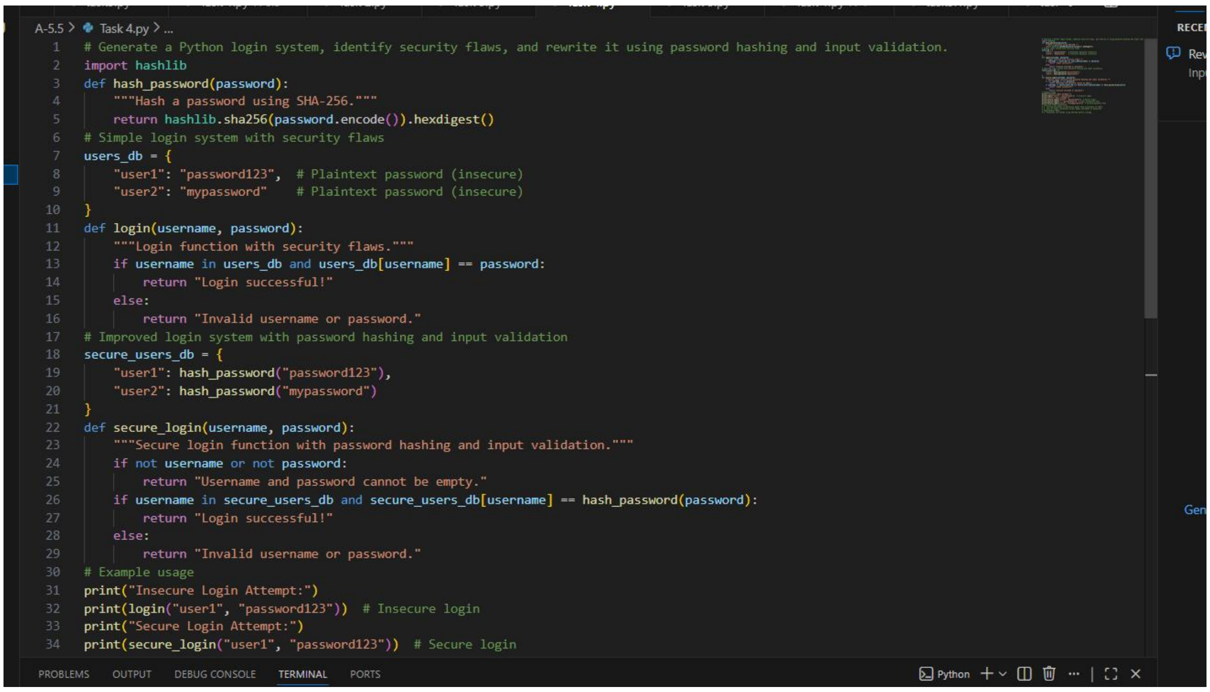
```
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC/A-5.5/Task 1.
py"
Enter a number to check if it's prime: 10
Basic method: 10 is prime? False
Optimized method: 10 is prime? False
PS C:\Users\pc\OneDrive\Desktop\AIAC>
```

Task Description #5 (Privacy in Data Logging)

#PROMPT

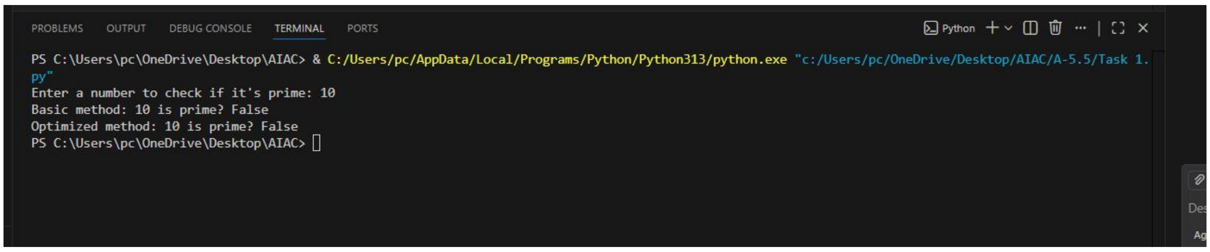
Generate a program and analyze it for possible bias or discriminatory logic, then revise it to ensure fairness and explain the improvements.

#CODE



```python
# Generate a Python login system, identify security flaws, and rewrite it using password hashing and input validation.
import hashlib
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()
# Simple login system with security flaws
users_db = {
    "user1": "password123",  # Plaintext password (insecure)
    "user2": "mypassword"    # Plaintext password (insecure)
}
def login(username, password):
    """Login function with security flaws."""
    if username in users_db and users_db[username] == password:
        return "Login successful!"
    else:
        return "Invalid username or password."
# Improved login system with password hashing and input validation
secure_users_db = {
    "user1": hash_password("password123"),
    "user2": hash_password("mypassword")
}
def secure_login(username, password):
    """Secure login function with password hashing and input validation."""
    if not username or not password:
        return "Username and password cannot be empty."
    if username in secure_users_db and secure_users_db[username] == hash_password(password):
        return "Login successful!"
    else:
        return "Invalid username or password."
# Example usage
print("Insecure Login Attempt:")
print(login("user1", "password123"))  # Insecure login
print("Secure Login Attempt:")
print(secure_login("user1", "password123"))  # Secure login
```

#OUTPUT



```
PS C:\Users\pc\OneDrive\Desktop\AIAC> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pc/OneDrive/Desktop/AIAC/A-5.5/Task 1.
py"
Enter a number to check if it's prime: 10
Basic method: 10 is prime? False
Optimized method: 10 is prime? False
PS C:\Users\pc\OneDrive\Desktop\AIAC> 
```

Explanation :

This program shows two login systems: one unsafe and one safer. In the unsafe version, passwords are stored as normal text, so anyone who sees the data can know the passwords. In the safer version, passwords are stored in a coded (hashed) form instead of plain text. When a user logs in, the entered password is converted into a hash and then compared with the stored one. The improved version also checks that the username and password are not empty. These changes help keep user passwords more secure.