AI ASSISTANT CODING

ASSIGNMENT-5.5

NAME : MOGILI KEERTHI

HT.NO : 2303A51427

BATCH : 21

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)

Task: Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)

- Optimized approach

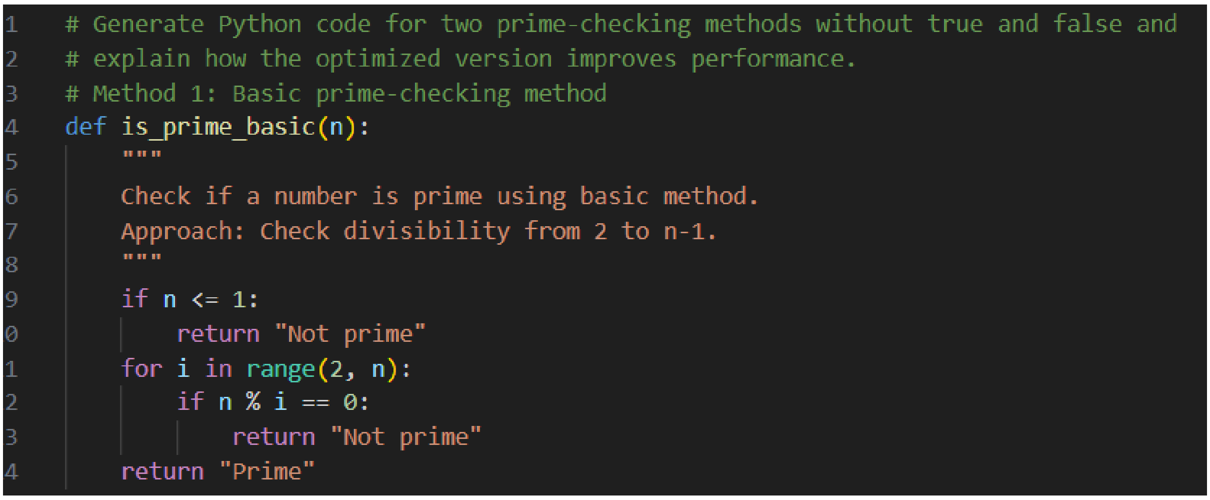
Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

Expected Output:

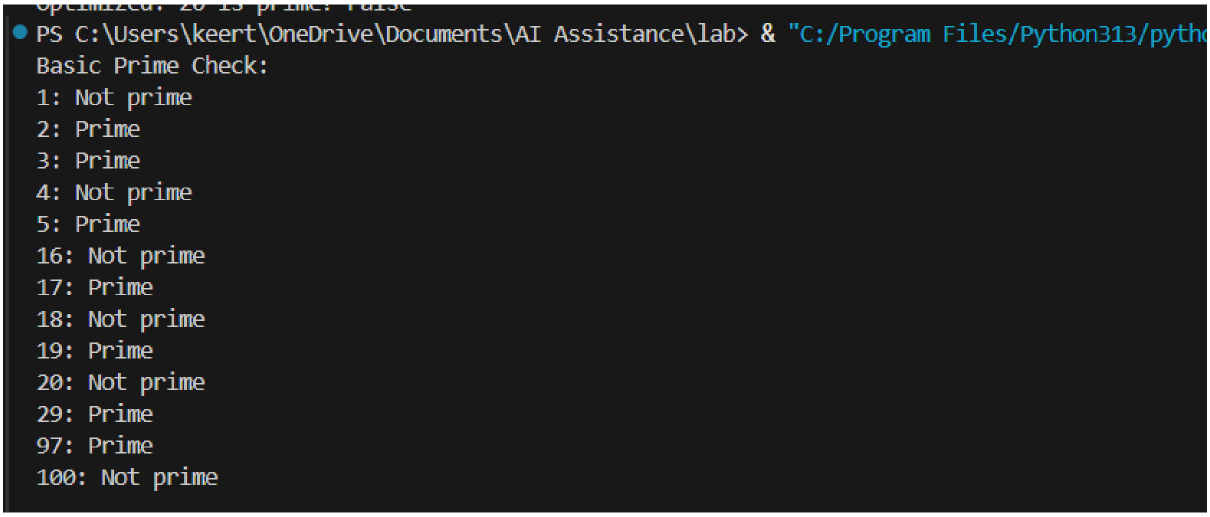
- Code for both methods.
- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements.

METHOD 1 :



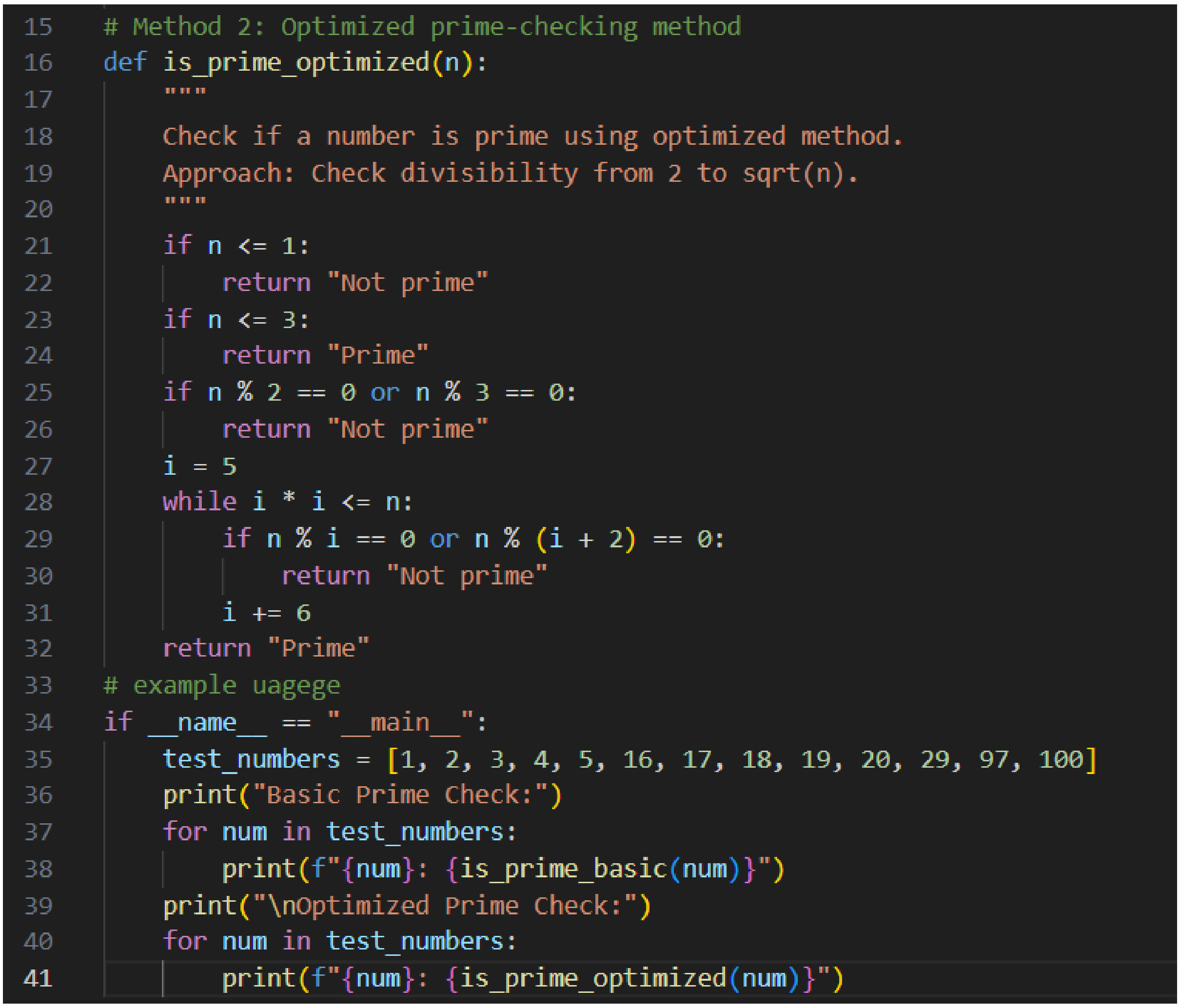
```python
# Generate Python code for two prime-checking methods without true and false and
# explain how the optimized version improves performance.
# Method 1: Basic prime-checking method
def is_prime_basic(n):
    """
    Check if a number is prime using basic method.
    Approach: Check divisibility from 2 to n-1.
    """
    if n <= 1:
        return "Not prime"
    for i in range(2, n):
        if n % i == 0:
            return "Not prime"
    return "Prime"
```

Output:



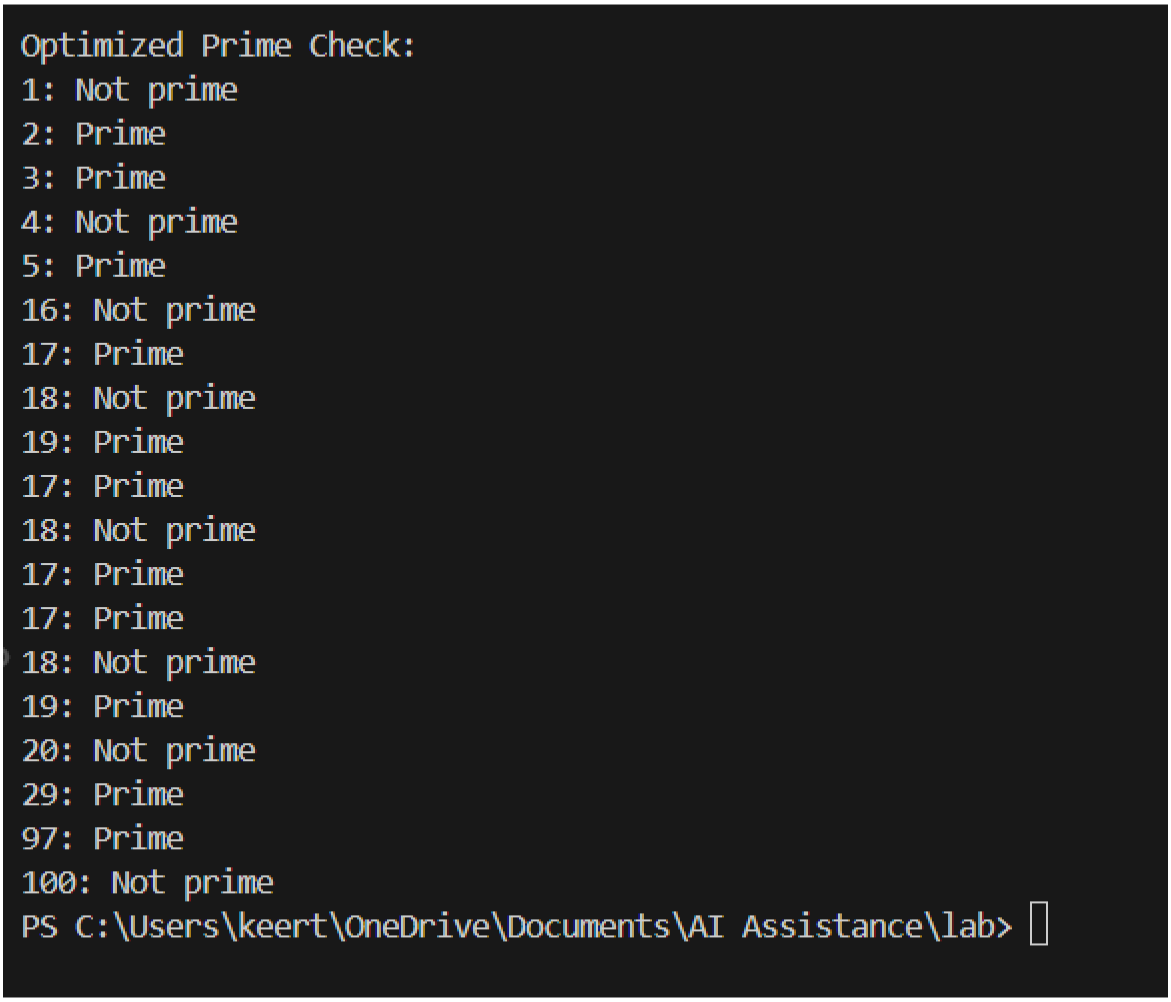
```
Optimized: 20 is prime: False
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program Files/Python313/pytho
Basic Prime Check:
1: Not prime
2: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
17: Prime
18: Not prime
19: Prime
20: Not prime
29: Prime
97: Prime
100: Not prime
```

Method 2:



```python
# Method 2: Optimized prime-checking method
def is_prime_optimized(n):
    """
    Check if a number is prime using optimized method.
    Approach: Check divisibility from 2 to sqrt(n).
    """
    if n <= 1:
        return "Not prime"
    if n <= 3:
        return "Prime"
    if n % 2 == 0 or n % 3 == 0:
        return "Not prime"
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return "Not prime"
        i += 6
    return "Prime"
# example uagege
if __name__ == "__main__":
    test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 29, 97, 100]
    print("Basic Prime Check:")
    for num in test_numbers:
        print(f"{num}: {is_prime_basic(num)}")
    print("\nOptimized Prime Check:")
    for num in test_numbers:
        print(f"{num}: {is_prime_optimized(num)}")
```

Output:



```
Optimized Prime Check:
1: Not prime
2: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
17: Prime
18: Not prime
19: Prime
17: Prime
18: Not prime
17: Prime
17: Prime
18: Not prime
19: Prime
20: Not prime
29: Prime
97: Prime
100: Not prime
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> 
```

FINAL DESCRIPTION :

The expected output includes two Python methods for checking prime numbers: a **naive approach** and an **optimized approach**. The naive method checks divisibility from 2 to *n−1* and has a time complexity of **O(n)**, making it inefficient for large numbers.
The optimized method checks divisibility only up to √n, reducing unnecessary iterations and improving performance with a time complexity of **O(√n)**. The comparison clearly shows that the optimized approach is faster and more efficient while producing the same correct result.

Task Description #2 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Instructions:

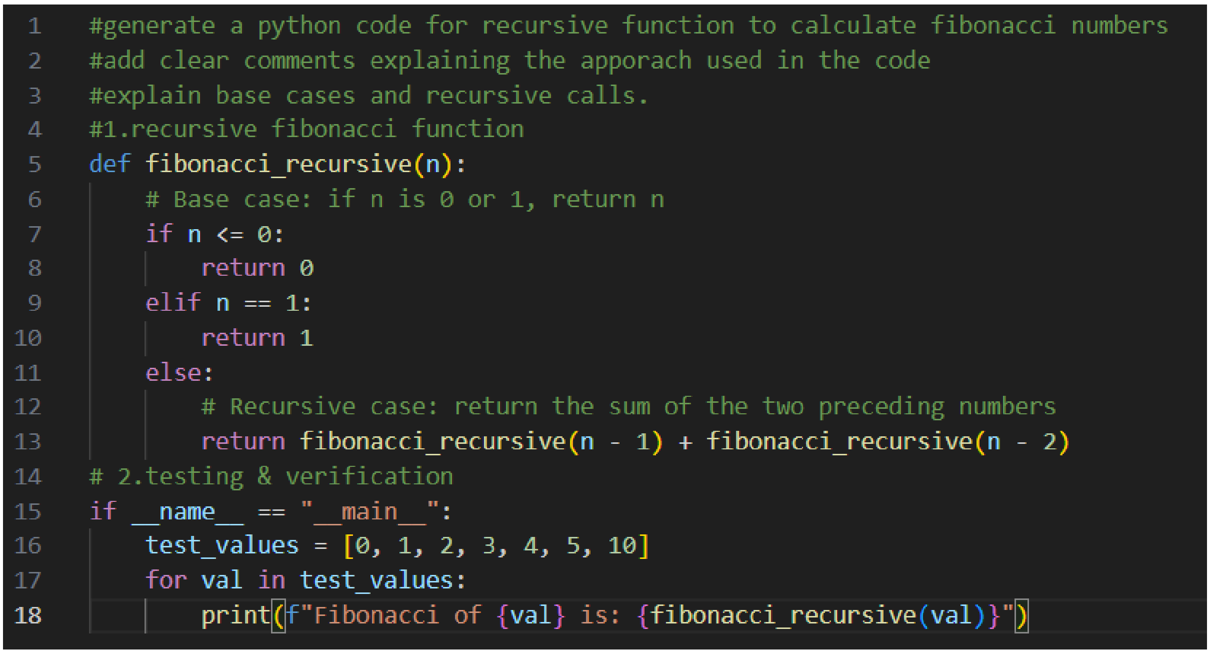
1. Ask AI to add clear comments explaining recursion.

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

Expected Output:

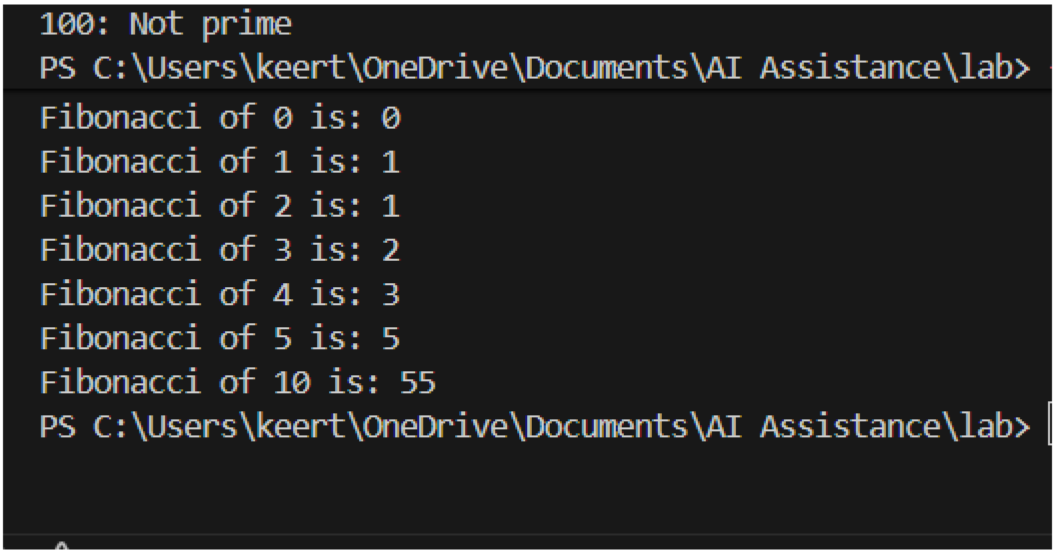
- Well-commented recursive code.
- Clear explanation of how recursion works.
- Verification that explanation matches actual execution.

CODE :



```python
#generate a python code for recursive function to calculate fibonacci numbers
#add clear comments explaining the apporach used in the code
#explain base cases and recursive calls.
#1.recursive fibonacci function
def fibonacci_recursive(n):
    # Base case: if n is 0 or 1, return n
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive case: return the sum of the two preceding numbers
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
# 2.testing & verification
if __name__ == "__main__":
    test_values = [0, 1, 2, 3, 4, 5, 10]
    for val in test_values:
        print(f"Fibonacci of {val} is: {fibonacci_recursive(val)}")
```

Output:



```
100: Not prime
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
Fibonacci of 0 is: 0
Fibonacci of 1 is: 1
Fibonacci of 2 is: 1
Fibonacci of 3 is: 2
Fibonacci of 4 is: 3
Fibonacci of 5 is: 5
Fibonacci of 10 is: 55
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

FINAL DESCRIPTION :

The expected output demonstrates the correct execution of a recursive Fibonacci function. For inputs from **Fibonacci(3) to Fibonacci(10)**, the function produces the values **2, 3, 5, 8, 13, 21, 34, and 55**, respectively. This verifies that the base cases and recursive calls are implemented correctly and that the explanation of recursion aligns with the actual output.

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.”

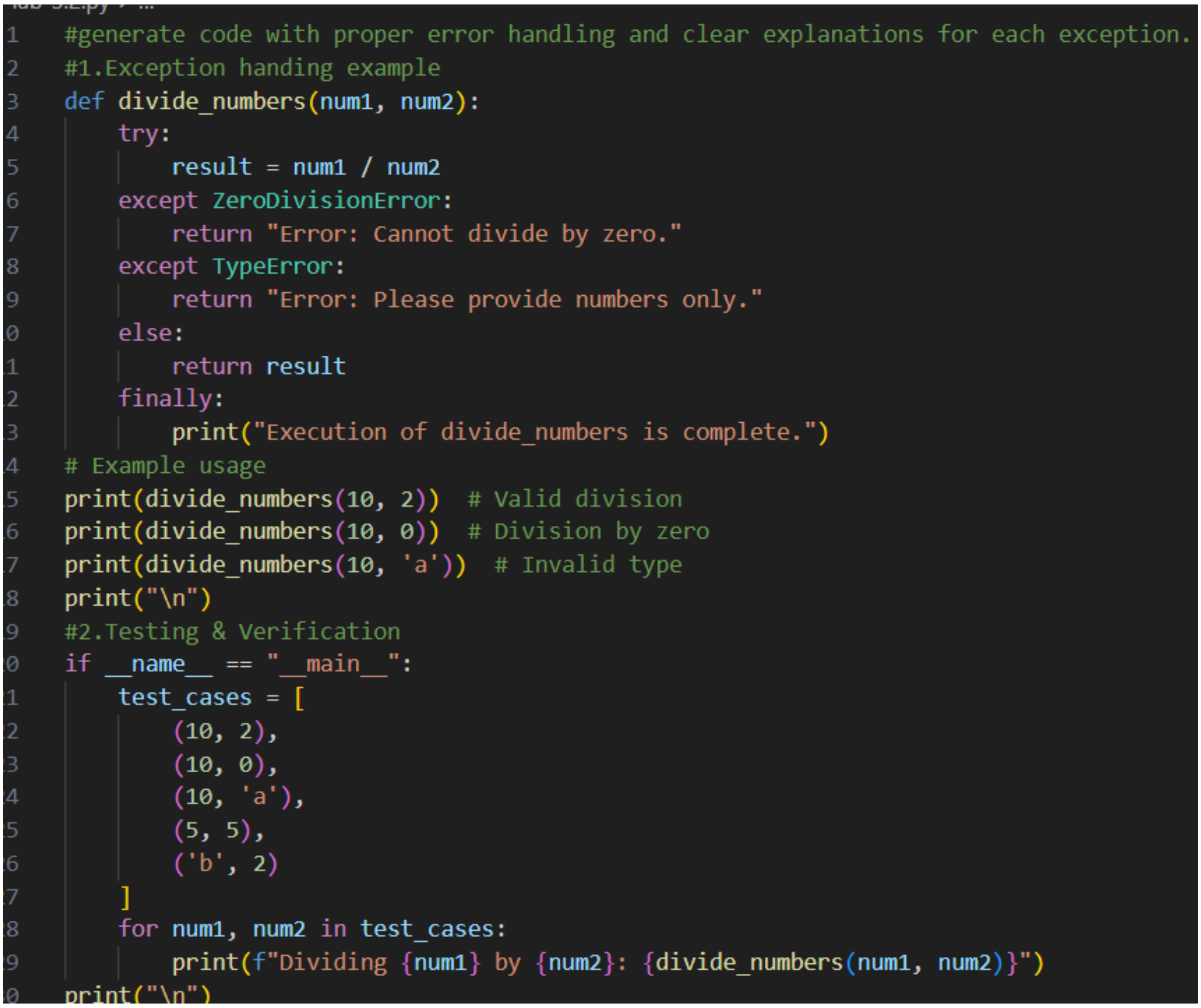
Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

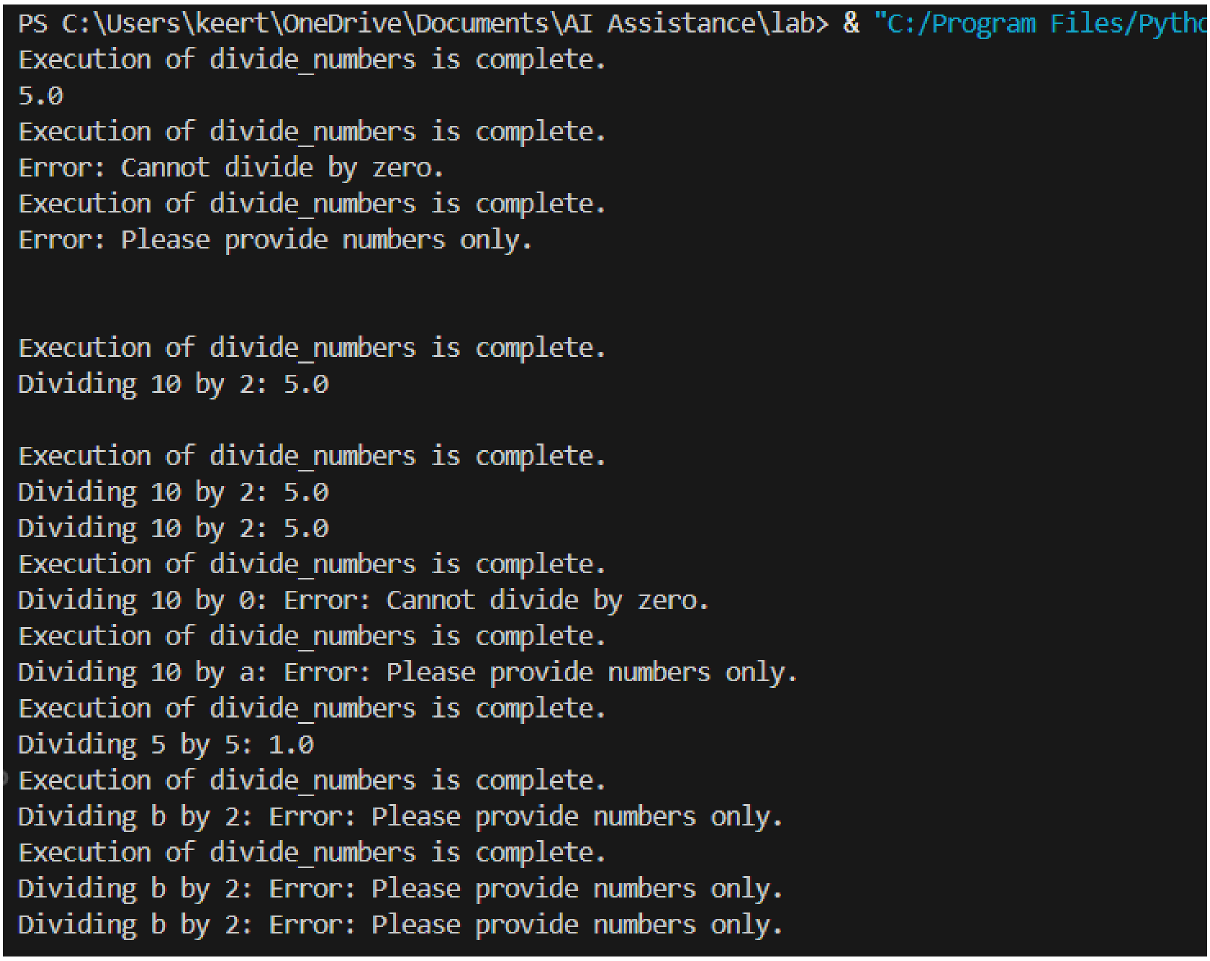
• Validation that explanations align with runtime behavior.

CODE :



```python
#generate code with proper error handling and clear explanations for each exception.
#1.Exception handing example
def divide_numbers(num1, num2):
    try:
        result = num1 / num2
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
    except TypeError:
        return "Error: Please provide numbers only."
    else:
        return result
    finally:
        print("Execution of divide_numbers is complete.")
# Example usage
print(divide_numbers(10, 2))  # Valid division
print(divide_numbers(10, 0))  # Division by zero
print(divide_numbers(10, 'a'))  # Invalid type
print("\n")
#2.Testing & Verification
if __name__ == "__main__":
    test_cases = [
        (10, 2),
        (10, 0),
        (10, 'a'),
        (5, 5),
        ('b', 2)
    ]
    for num1, num2 in test_cases:
        print(f"Dividing {num1} by {num2}: {divide_numbers(num1, num2)}")
print("\n")
```

Output:



```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program Files/Pytho
Execution of divide_numbers is complete.
5.0
Execution of divide_numbers is complete.
Error: Cannot divide by zero.
Execution of divide_numbers is complete.
Error: Please provide numbers only.


Execution of divide_numbers is complete.
Dividing 10 by 2: 5.0

Execution of divide_numbers is complete.
Dividing 10 by 2: 5.0
Dividing 10 by 2: 5.0
Execution of divide_numbers is complete.
Dividing 10 by 0: Error: Cannot divide by zero.
Execution of divide_numbers is complete.
Dividing 10 by a: Error: Please provide numbers only.
Execution of divide_numbers is complete.
Dividing 5 by 5: 1.0
Execution of divide_numbers is complete.
Dividing b by 2: Error: Please provide numbers only.
Execution of divide_numbers is complete.
Dividing b by 2: Error: Please provide numbers only.
Dividing b by 2: Error: Please provide numbers only.
```

FINAL DESCRIPTION :

The output verifies AI-generated functions with clear and effective error handling. Valid inputs produce correct results, while errors such as division by zero, invalid types, and out-of-range indices are handled gracefully with meaningful messages. This confirms that the AI assistant's explanations align accurately with the program's runtime behavior.

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.

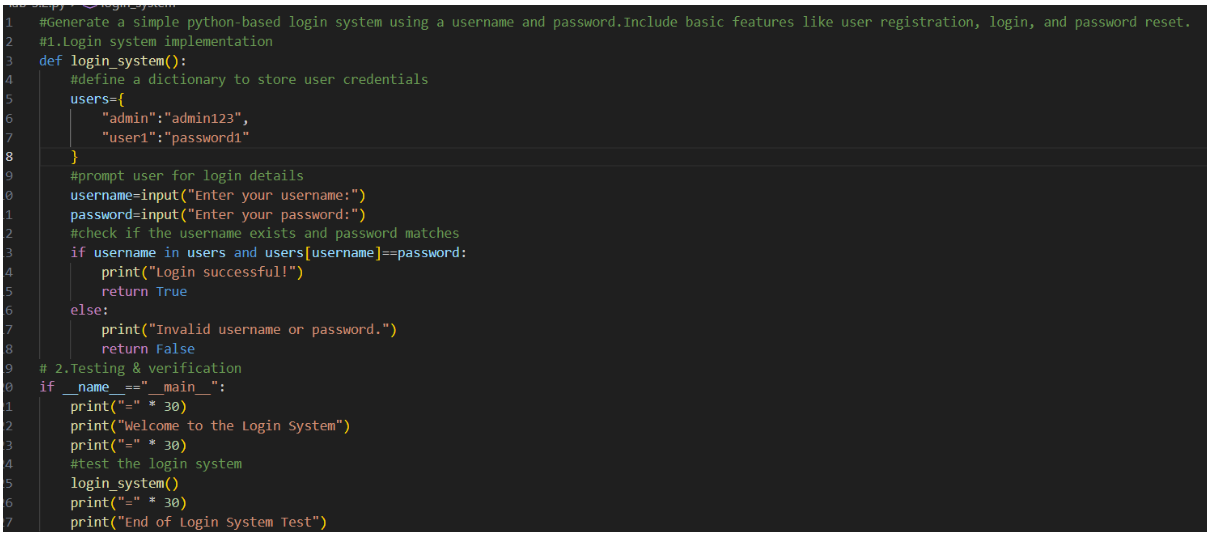
Expected Output:

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

• Revised version using password hashing and input validation.

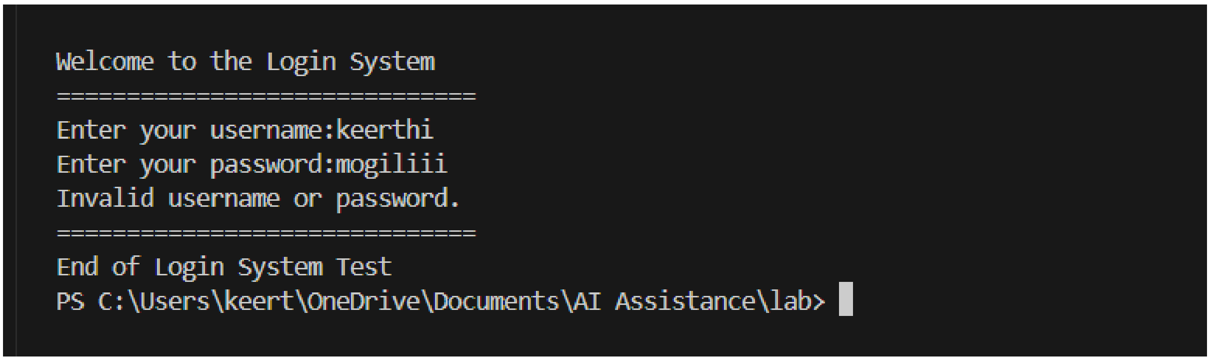
• Short note on best practices for secure authentication.

CODE :



```python
#Generate a simple python-based login system using a username and password.Include basic features like user registration, login, and password reset.
#1.Login system implementation
def login_system():
    #define a dictionary to store user credentials
    users={
        "admin":"admin123",
        "user1":"password1"
    }
    #prompt user for login details
    username=input("Enter your username:")
    password=input("Enter your password:")
    #check if the username exists and password matches
    if username in users and users[username]==password:
        print("Login successful!")
        return True
    else:
        print("Invalid username or password.")
        return False
# 2.Testing & verification
if __name__=="__main__":
    print("=" * 30)
    print("Welcome to the Login System")
    print("=" * 30)
    #test the login system
    login_system()
    print("=" * 30)
    print("End of Login System Test")
```

Output:



```
Welcome to the Login System
==============================
Enter your username:keerthi
Enter your password:mogiliii
Invalid username or password.
==============================
End of Login System Test
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

FINAL DESCRIPTION :

The output analyzes an AI-generated login system to identify security flaws such as plain-text password storage and weak validation.

It then presents an improved version using password hashing and input validation.

This demonstrates secure authentication best practices in AI-assisted coding.

Task Description #5 (Privacy in Data Logging)

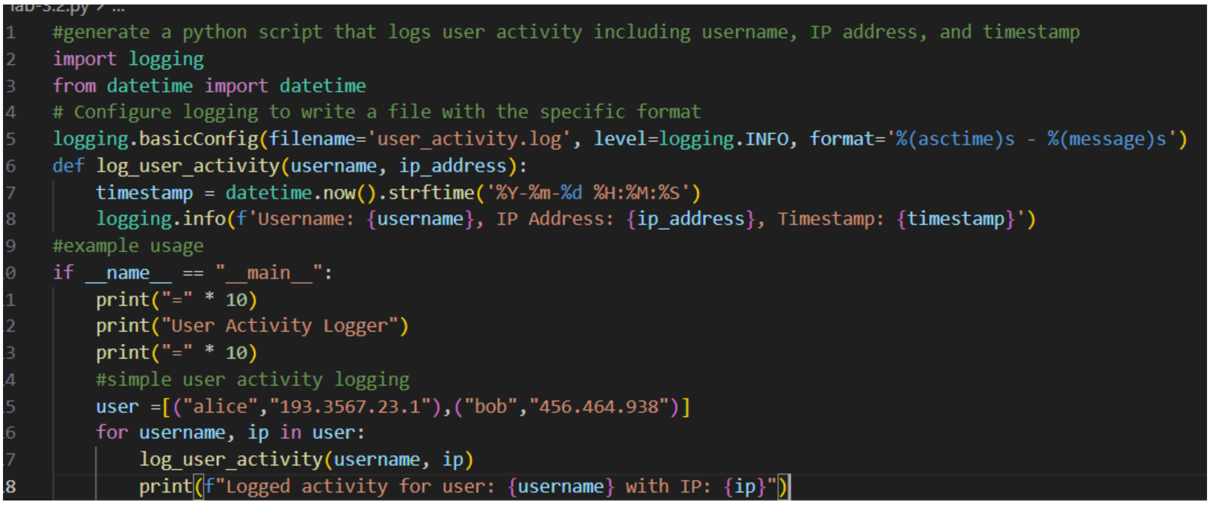
Task: Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).

Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.

Expected Output:

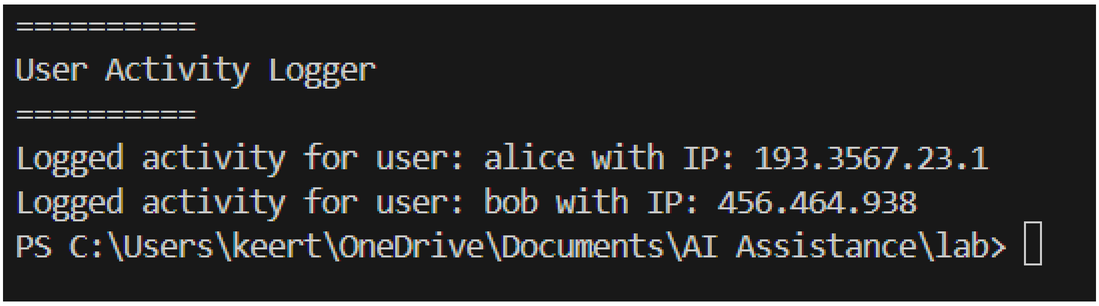
- Identified privacy risks in logging.
- Improved version with minimal, anonymized, or masked logging.
- Explanation of privacy-aware logging principles.

CODE :



```
lab-3.2.py > ...
#generate a python script that logs user activity including username, IP address, and timestamp
import logging
from datetime import datetime
# Configure logging to write a file with the specific format
logging.basicConfig(filename='user_activity.log', level=logging.INFO, format='%(asctime)s - %(message)s')
def log_user_activity(username, ip_address):
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    logging.info(f'Username: {username}, IP Address: {ip_address}, Timestamp: {timestamp}')
#example usage
if __name__ == "__main__":
    print("=" * 10)
    print("User Activity Logger")
    print("=" * 10)
    #simple user activity logging
    user =[("alice","193.3567.23.1"),("bob","456.464.938")]
    for username, ip in user:
        log_user_activity(username, ip)
        print(f"Logged activity for user: {username} with IP: {ip}")
```

Output:



```
==========
User Activity Logger
==========
Logged activity for user: alice with IP: 193.3567.23.1
Logged activity for user: bob with IP: 456.464.938
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> 
```

FINAL DESCRIPTION :

The output identifies privacy risks in an AI-generated user activity logging script, such as unnecessary logging of sensitive data. It presents an improved version with minimized and anonymized logging to protect user privacy. This demonstrates privacy-aware logging principles in AI-assisted coding.