AI ASSISTANT CODING

ASSIGNMENT – 5.5

NAME : K. LAXMAN

HT.NO : 2303A51428

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.

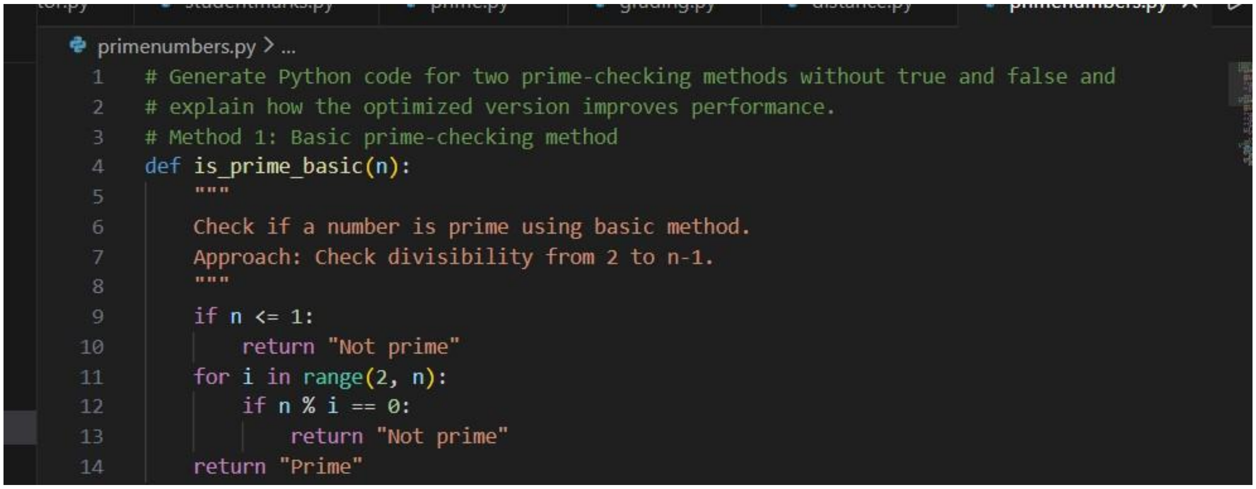
Task Description #1 (Transparency in Algorithm Optimization) Task: Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach Prompt:

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

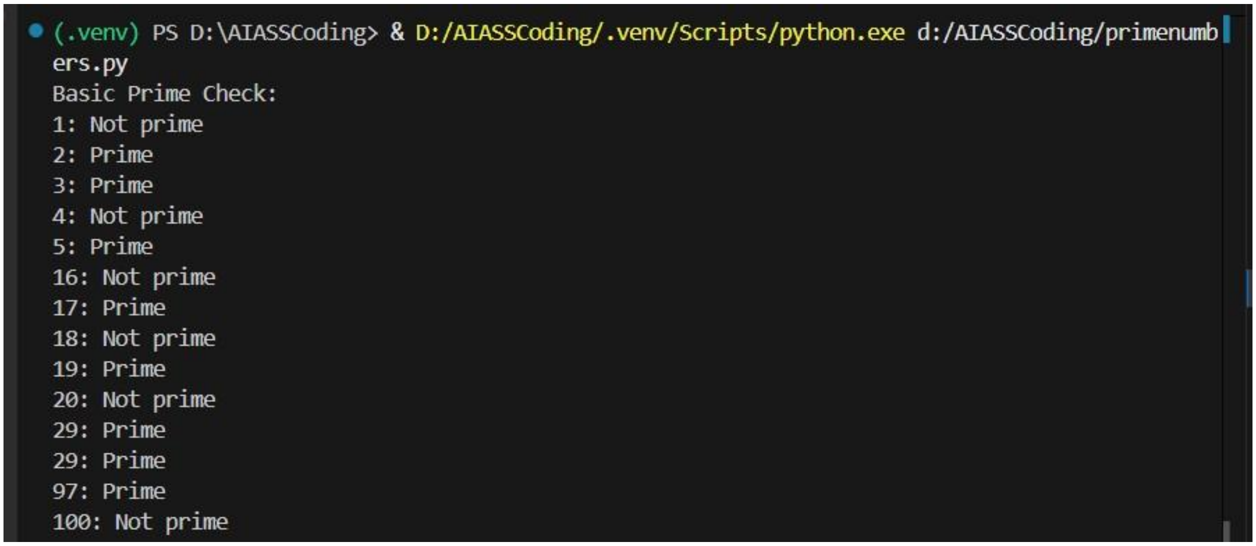
- Code for both methods.
- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements.

METHOD 1 :



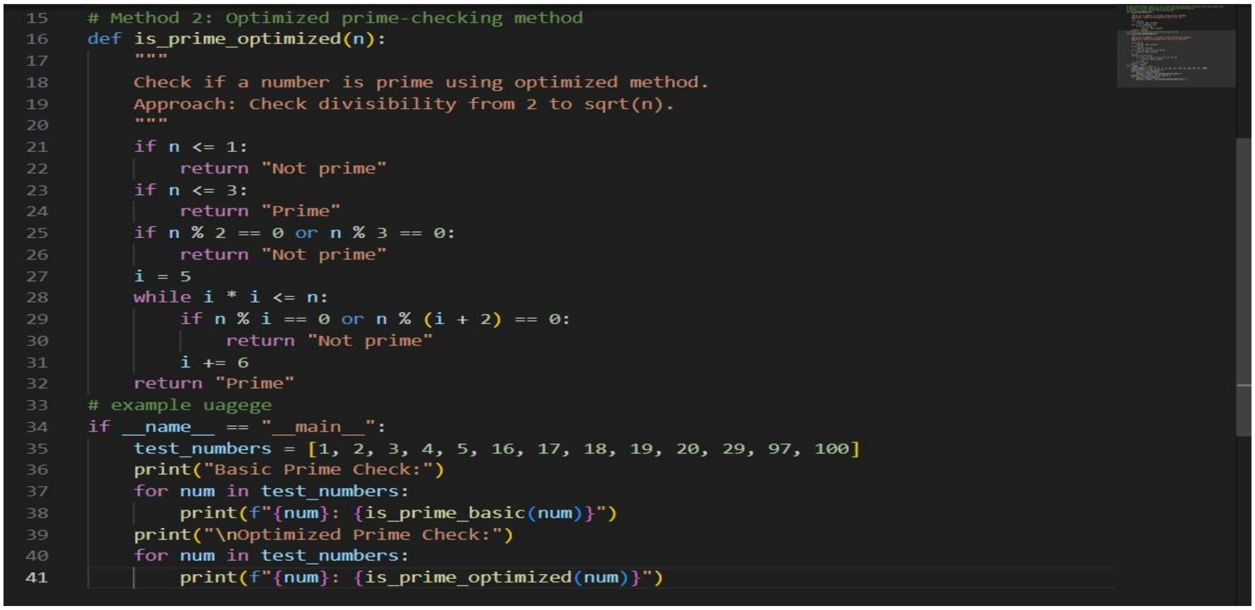
```python
# Generate Python code for two prime-checking methods without true and false and
# explain how the optimized version improves performance.
# Method 1: Basic prime-checking method
def is_prime_basic(n):
    """
    Check if a number is prime using basic method.
    Approach: Check divisibility 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:



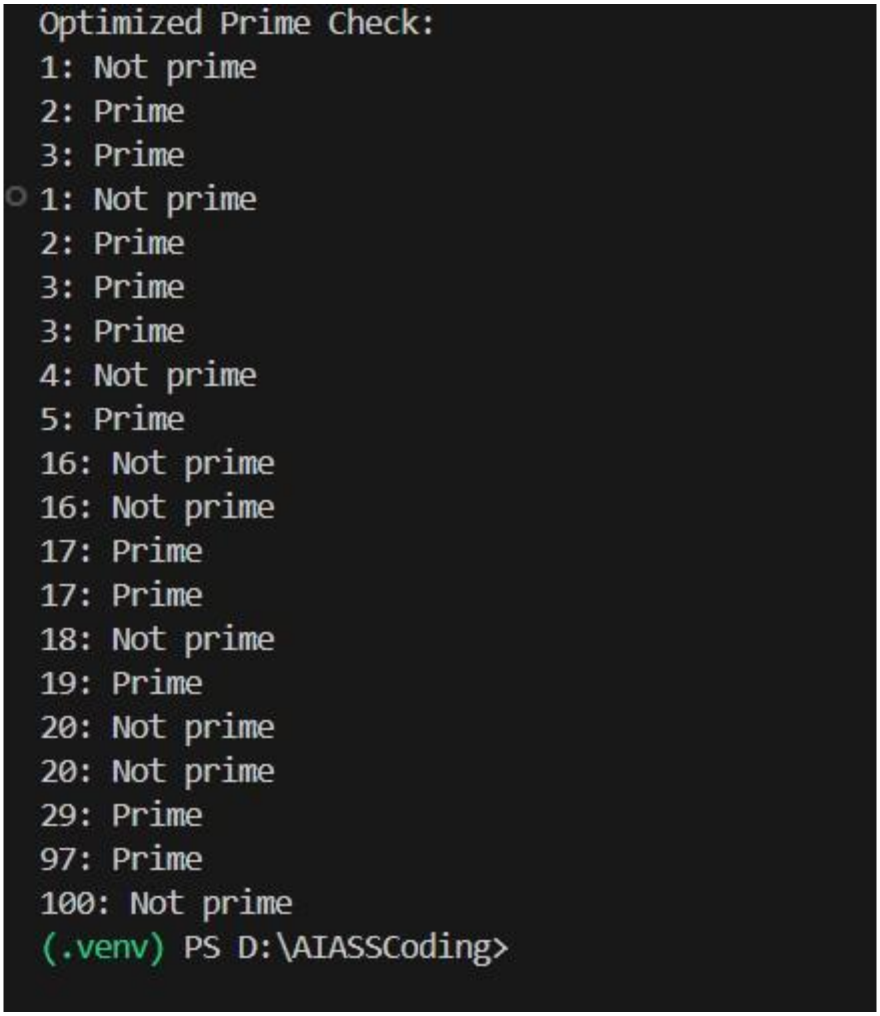
```
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/primenumb
ers.py
Basic Prime Check:
1: Not prime
2: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
17: Prime
18: Not prime
19: Prime
20: Not prime
29: Prime
29: Prime
97: Prime
100: Not prime
```

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
1: Not prime
2: Prime
3: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
16: Not prime
17: Prime
17: Prime
18: Not prime
19: Prime
20: Not prime
20: Not prime
29: Prime
97: Prime
100: Not prime
(.venv) PS D:\AIASSCoding>
```

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.

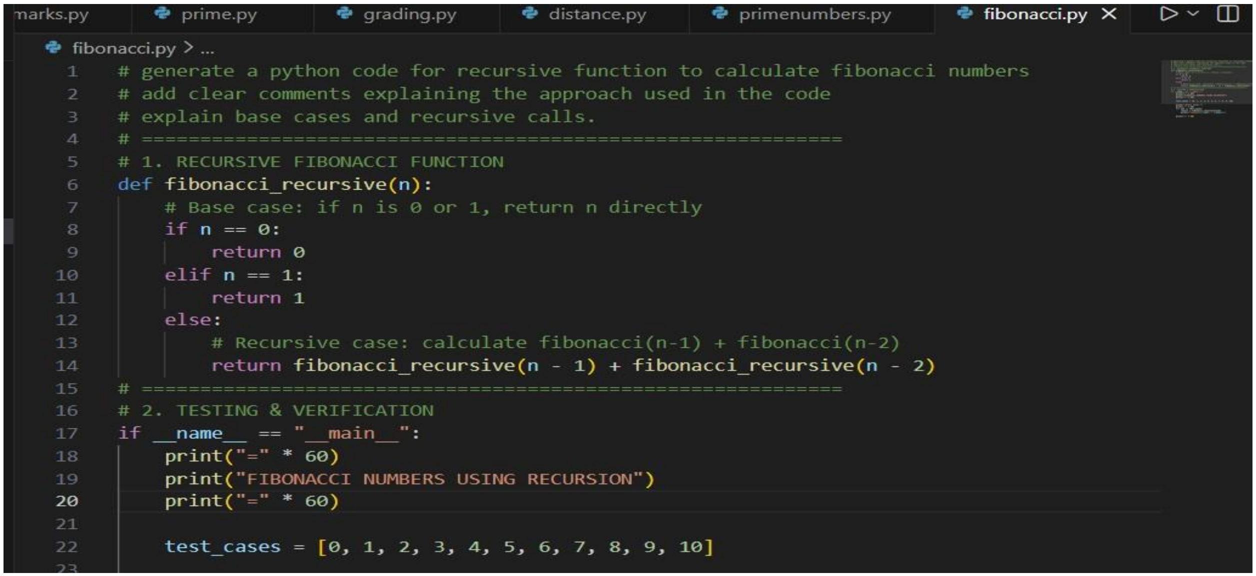
Expected Output:

- Well-commented recursive code.

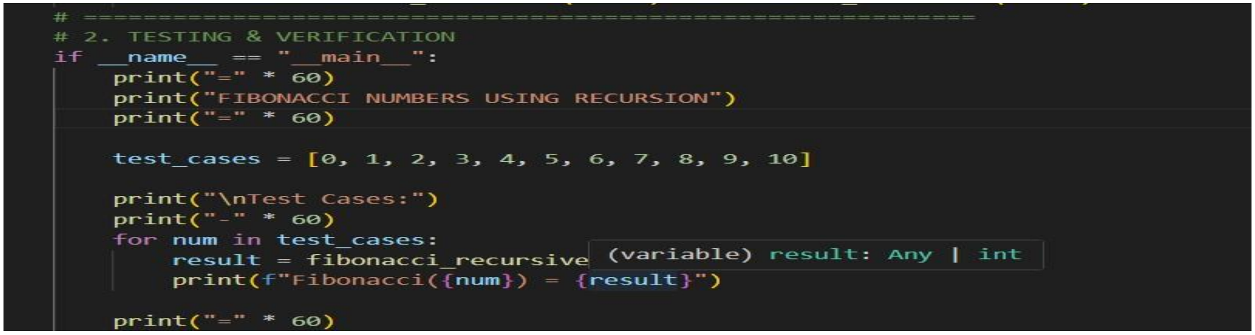
- Clear explanation of how recursion works.

- Verification that explanation matches actual execution.

CODE :

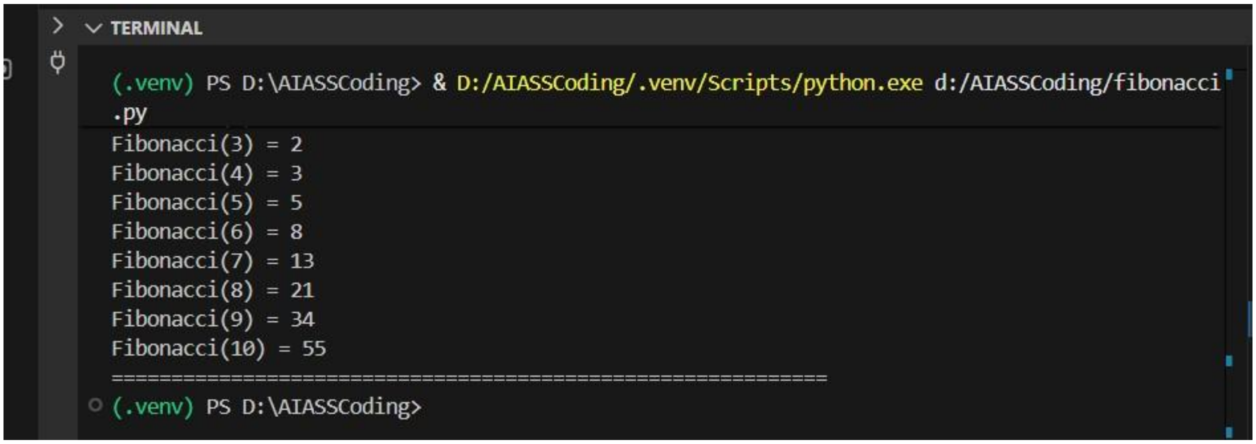


```python
# generate a python code for recursive function to calculate fibonacci numbers
# add clear comments explaining the approach used in the code
# explain base cases and recursive calls.
# ============================================================
# 1. RECURSIVE FIBONACCI FUNCTION
def fibonacci_recursive(n):
    # Base case: if n is 0 or 1, return n directly
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive case: calculate fibonacci(n-1) + fibonacci(n-2)
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
# ============================================================
# 2. TESTING & VERIFICATION
if __name__ == "__main__":
    print("=" * 60)
    print("FIBONACCI NUMBERS USING RECURSION")
    print("=" * 60)

    test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```



```python
# ============================================================
# 2. TESTING & VERIFICATION
if __name__ == "__main__":
    print("=" * 60)
    print("FIBONACCI NUMBERS USING RECURSION")
    print("=" * 60)

    test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    print("\nTest Cases:")
    print("-" * 60)
    for num in test_cases:
        result = fibonacci_recursive
        print(f"Fibonacci({num}) = {result}")

    print("=" * 60)
```

OUTPUT :



```
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/fibonacci
.py
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
============================================================
(.venv) PS D:\AIASSCoding>
```

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.

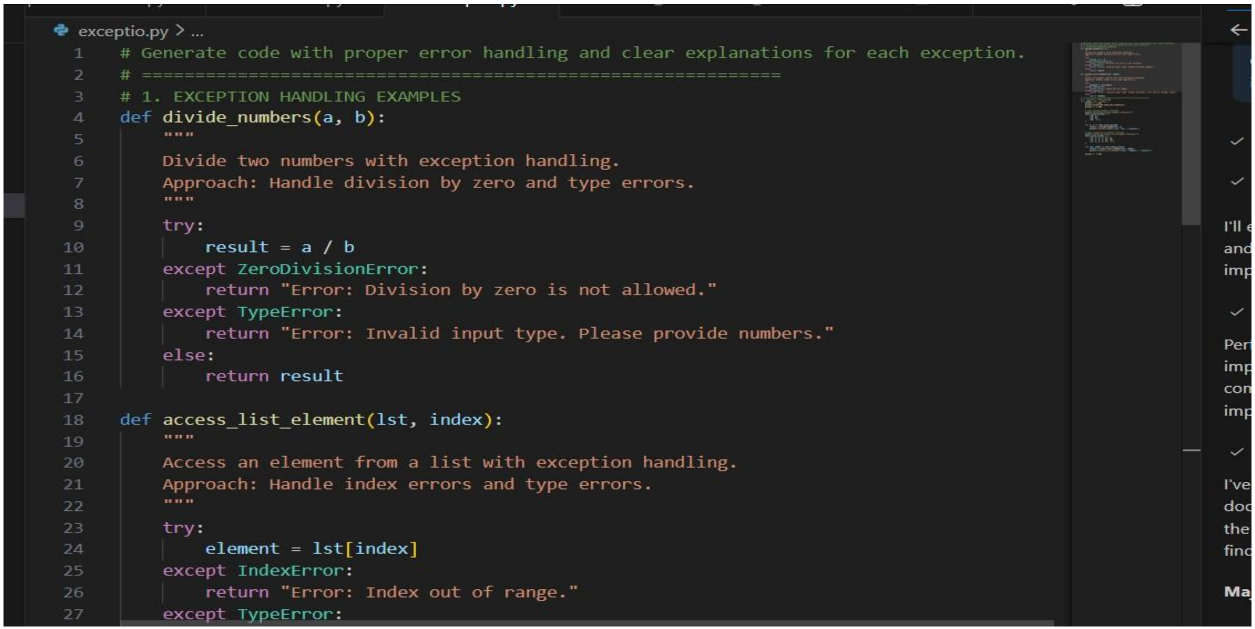
Prompt:

“Generate code with proper error handling and clear explanations for

each exception.” Expected Output:

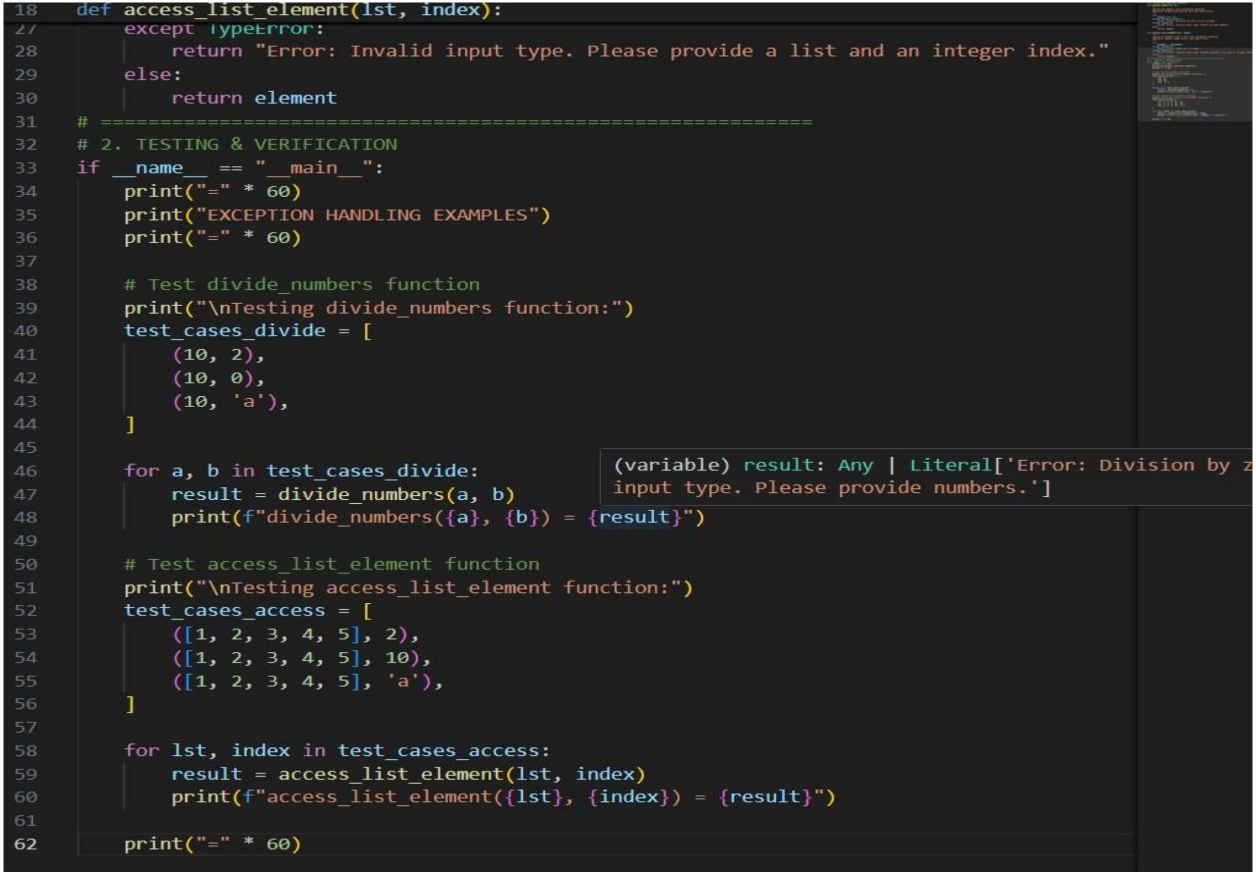
- Code with meaningful exception handling.
- Clear comments explaining each error scenario.
- Validation that explanations align with runtime behavior.

CODE :



exceptio.py > ...

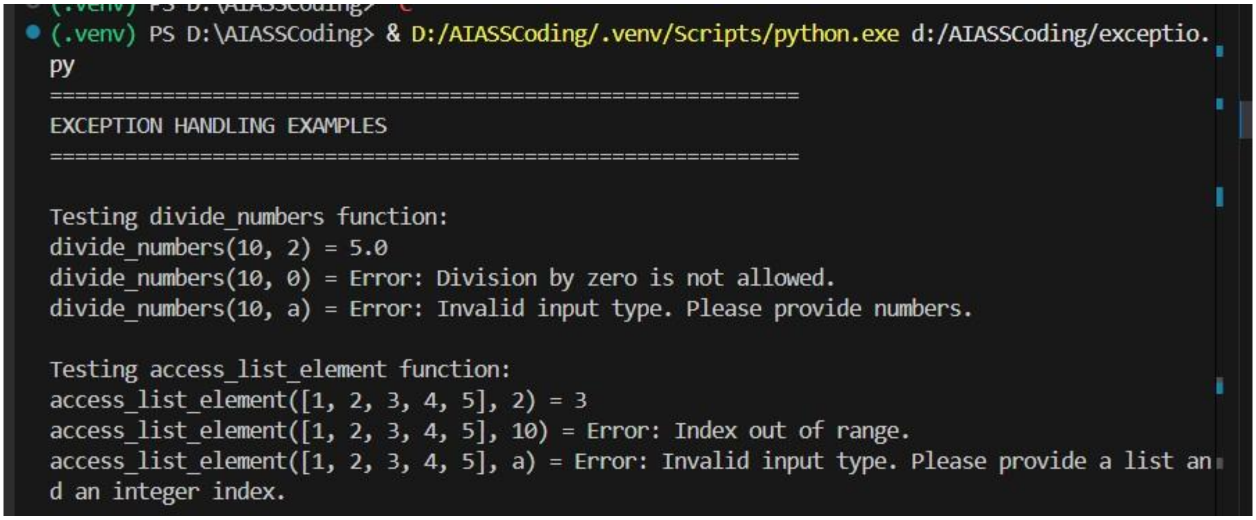
```python
# Generate code with proper error handling and clear explanations for each exception.
# ============================================================
# 1. EXCEPTION HANDLING EXAMPLES
def divide_numbers(a, b):
    """
    Divide two numbers with exception handling.
    Approach: Handle division by zero and type errors.
    """
    try:
        result = a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Invalid input type. Please provide numbers."
    else:
        return result

def access_list_element(lst, index):
    """
    Access an element from a list with exception handling.
    Approach: Handle index errors and type errors.
    """
    try:
        element = lst[index]
    except IndexError:
        return "Error: Index out of range."
    except TypeError:
```



```python
def access_list_element(lst, index):
    except TypeError:
        return "Error: Invalid input type. Please provide a list and an integer index."
    else:
        return element
# ============================================================
# 2. TESTING & VERIFICATION
if __name__ == "__main__":
    print("=" * 60)
    print("EXCEPTION HANDLING EXAMPLES")
    print("=" * 60)

    # Test divide_numbers function
    print("\nTesting divide_numbers function:")
    test_cases_divide = [
        (10, 2),
        (10, 0),
        (10, 'a'),
    ]

    for a, b in test_cases_divide:
        result = divide_numbers(a, b)
        print(f"divide_numbers({a}, {b}) = {result}")

    # Test access_list_element function
    print("\nTesting access_list_element function:")
    test_cases_access = [
        ([1, 2, 3, 4, 5], 2),
        ([1, 2, 3, 4, 5], 10),
        ([1, 2, 3, 4, 5], 'a'),
    ]

    for lst, index in test_cases_access:
        result = access_list_element(lst, index)
        print(f"access_list_element({lst}, {index}) = {result}")

    print("=" * 60)
```

```
(variable) result: Any | Literal['Error: Division by z
input type. Please provide numbers.']
```

OUTPUT :



```
(.venv) PS D:\AIASSCoding> ^C
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/exceptio.
py
============================================================
EXCEPTION HANDLING EXAMPLES
============================================================

Testing divide_numbers function:
divide_numbers(10, 2) = 5.0
divide_numbers(10, 0) = Error: Division by zero is not allowed.
divide_numbers(10, a) = Error: Invalid input type. Please provide numbers.

Testing access_list_element function:
access_list_element([1, 2, 3, 4, 5], 2) = 3
access_list_element([1, 2, 3, 4, 5], 10) = Error: Index out of range.
access_list_element([1, 2, 3, 4, 5], a) = Error: Invalid input type. Please provide a list an
d an integer index.
```

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.

---

ask Description #4 (Security in User Authentication)

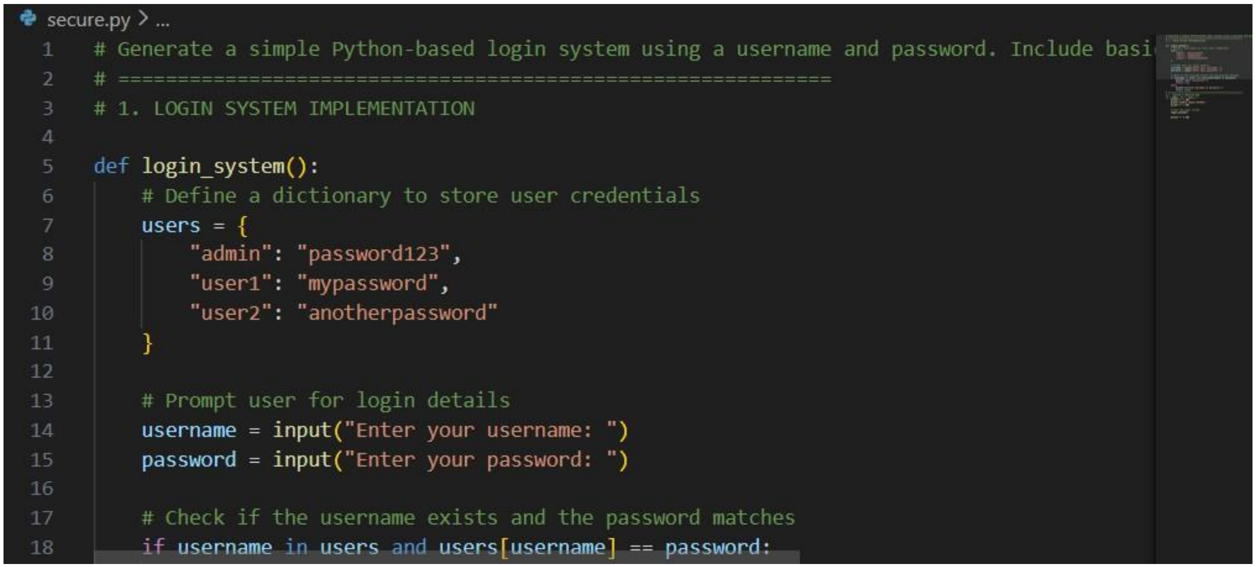
Task: Use an AI tool to generate a Python-based login system. Analyze:

Check whether the AI uses secure password handling practices.

Expected Output:

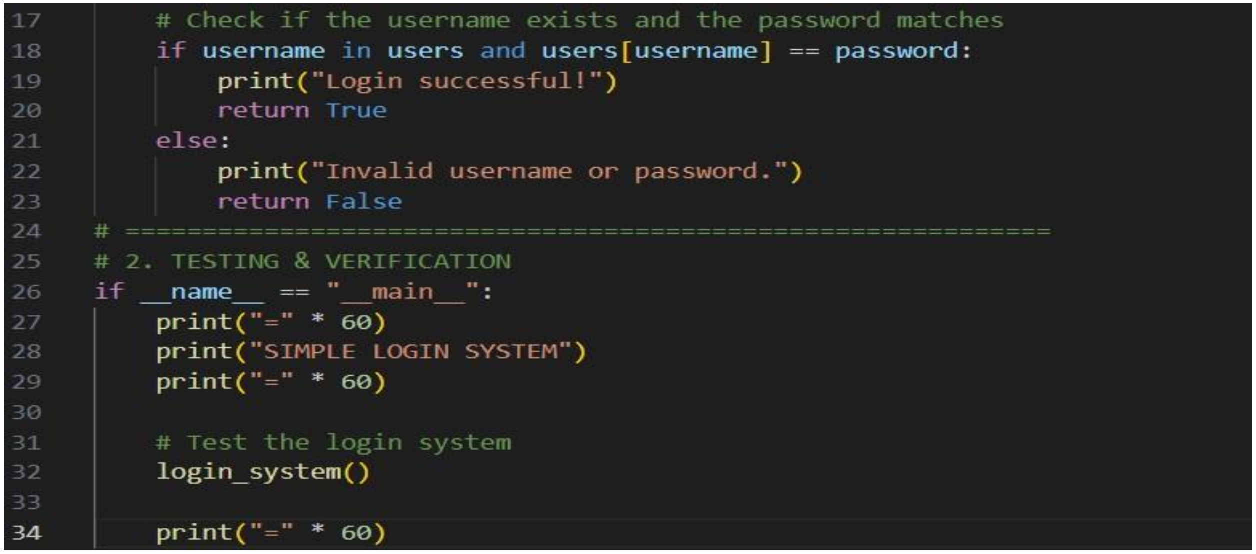
- Identification of security flaws (plain-text passwords, weak validation).
- Revised version using password hashing and input validation.
- Short note on best practices for secure authentication.

CODE :



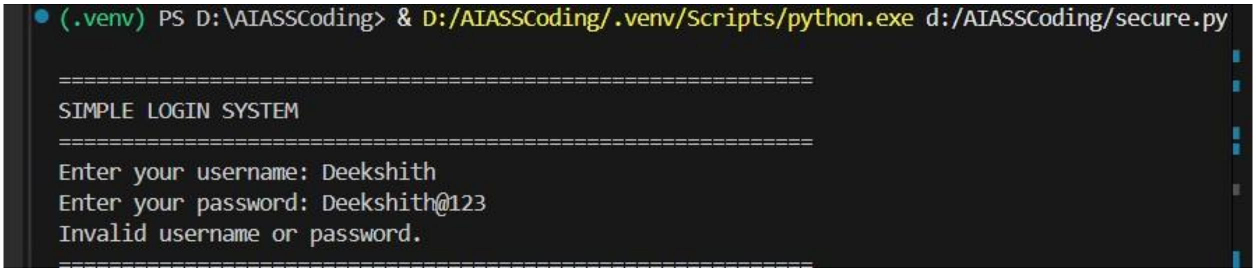
secure.py > ...

```python
# Generate a simple Python-based login system using a username and password. Include basi
# ============================================================
# 1. LOGIN SYSTEM IMPLEMENTATION

def login_system():
    # Define a dictionary to store user credentials
    users = {
        "admin": "password123",
        "user1": "mypassword",
        "user2": "anotherpassword"
    }

    # Prompt user for login details
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    # Check if the username exists and the password matches
    if username in users and users[username] == password:
```



```python
    # Check if the username exists and the 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("=" * 60)
    print("SIMPLE LOGIN SYSTEM")
    print("=" * 60)

    # Test the login system
    login_system()

    print("=" * 60)
```

OUTPUT :



```
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/secure.py

============================================================
SIMPLE LOGIN SYSTEM
============================================================
Enter your username: Deekshith
Enter your password: Deekshith@123
Invalid username or password.
============================================================
```

FINAL DESCRIPTION :

The output analyzes an AI-generated login system to identify security flaws such as plaintext 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.

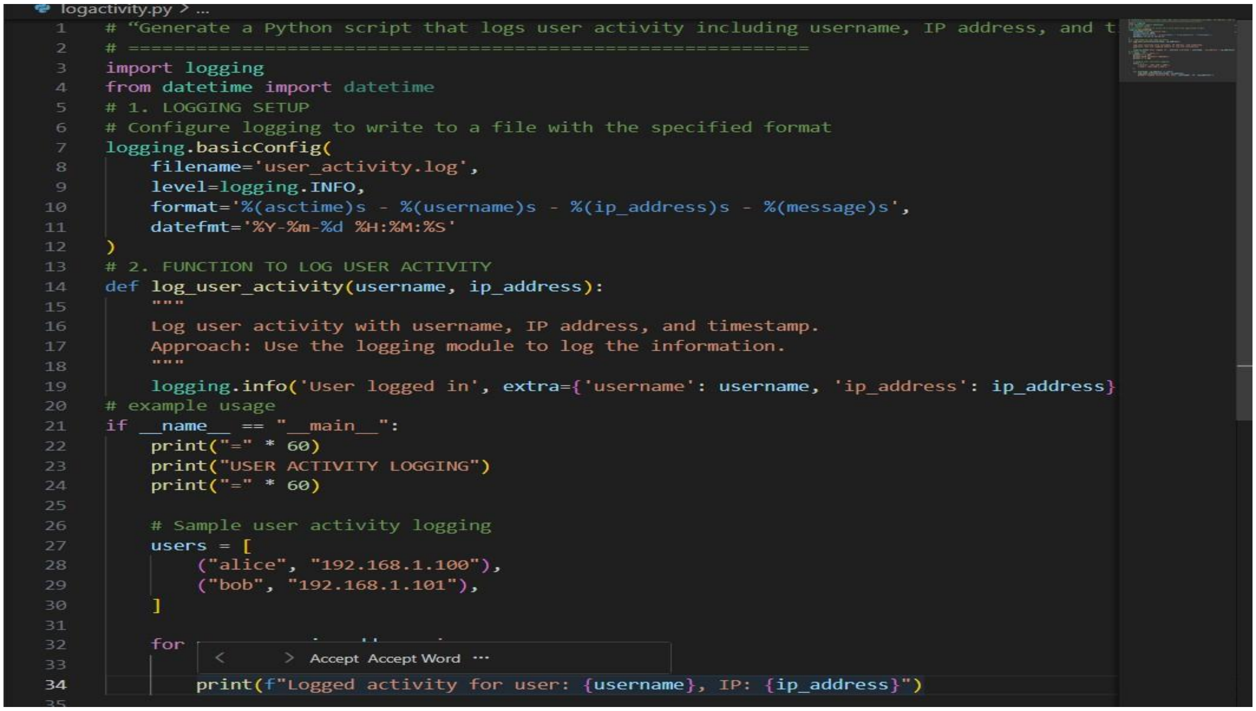
Expected Output:

• Identified privacy risks in logging.

• Improved version with minimal, anonymized, or masked
logging.

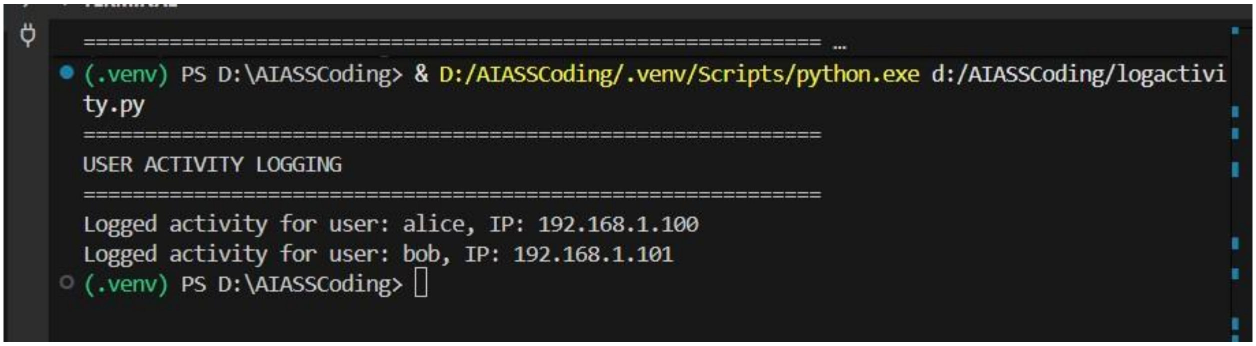
• Explanation of privacy-aware logging principles.

CODE :



```python
# “Generate a Python script that logs user activity including username, IP address, and t
# ============================================================
import logging
from datetime import datetime
# 1. LOGGING SETUP
# Configure logging to write to a file with the specified format
logging.basicConfig(
    filename='user_activity.log',
    level=logging.INFO,
    format='%(asctime)s - %(username)s - %(ip_address)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)
# 2. FUNCTION TO LOG USER ACTIVITY
def log_user_activity(username, ip_address):
    """
    Log user activity with username, IP address, and timestamp.
    Approach: Use the logging module to log the information.
    """
    logging.info('User logged in', extra={'username': username, 'ip_address': ip_address}
# example usage
if __name__ == "__main__":
    print("=" * 60)
    print("USER ACTIVITY LOGGING")
    print("=" * 60)

    # Sample user activity logging
    users = [
        ("alice", "192.168.1.100"),
        ("bob", "192.168.1.101"),
    ]

    for

        print(f"Logged activity for user: {username}, IP: {ip_address}")
```

OUTPUT :



```
============================================================ …
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/logactivi
ty.py
============================================================
USER ACTIVITY LOGGING
============================================================
Logged activity for user: alice, IP: 192.168.1.100
Logged activity for user: bob, IP: 192.168.1.101
(.venv) PS D:\AIASSCoding> 
```

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.