Name: T. Snuhith Reddy

Roll Number: 2303A510H9

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

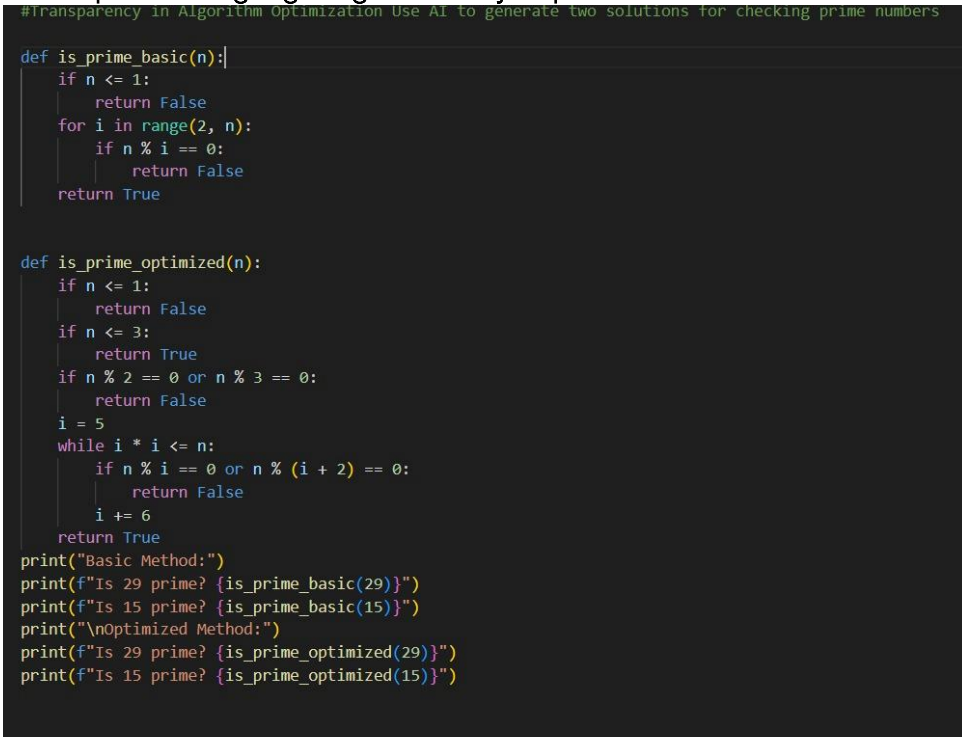
explain how the optimized version improves performance.”

Expected Output:

• Code for both methods.

• Transparent explanation of time complexity.

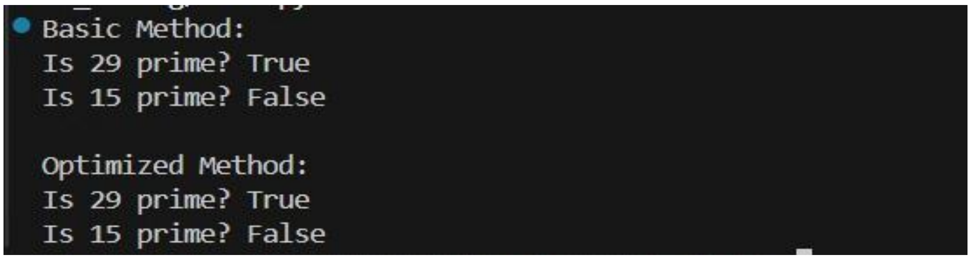
• Comparison highlighting efficiency improvements.



```python
#Transparency in Algorithm Optimization Use AI to generate two solutions for checking prime numbers

def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True


def is_prime_optimized(n):
    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
print("Basic Method:")
print(f"Is 29 prime? {is_prime_basic(29)}")
print(f"Is 15 prime? {is_prime_basic(15)}")
print("\nOptimized Method:")
print(f"Is 29 prime? {is_prime_optimized(29)}")
print(f"Is 15 prime? {is_prime_optimized(15)}")
```



```
Basic Method:
Is 29 prime? True
Is 15 prime? False

Optimized Method:
Is 29 prime? True
Is 15 prime? False
```

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 `√n` because if `n` has a factor greater than `√n`, it must also have a corresponding factor smaller than `√n`.

Time Complexity:

- Naive approach: **O(n)**
- Optimized approach: **O(√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.

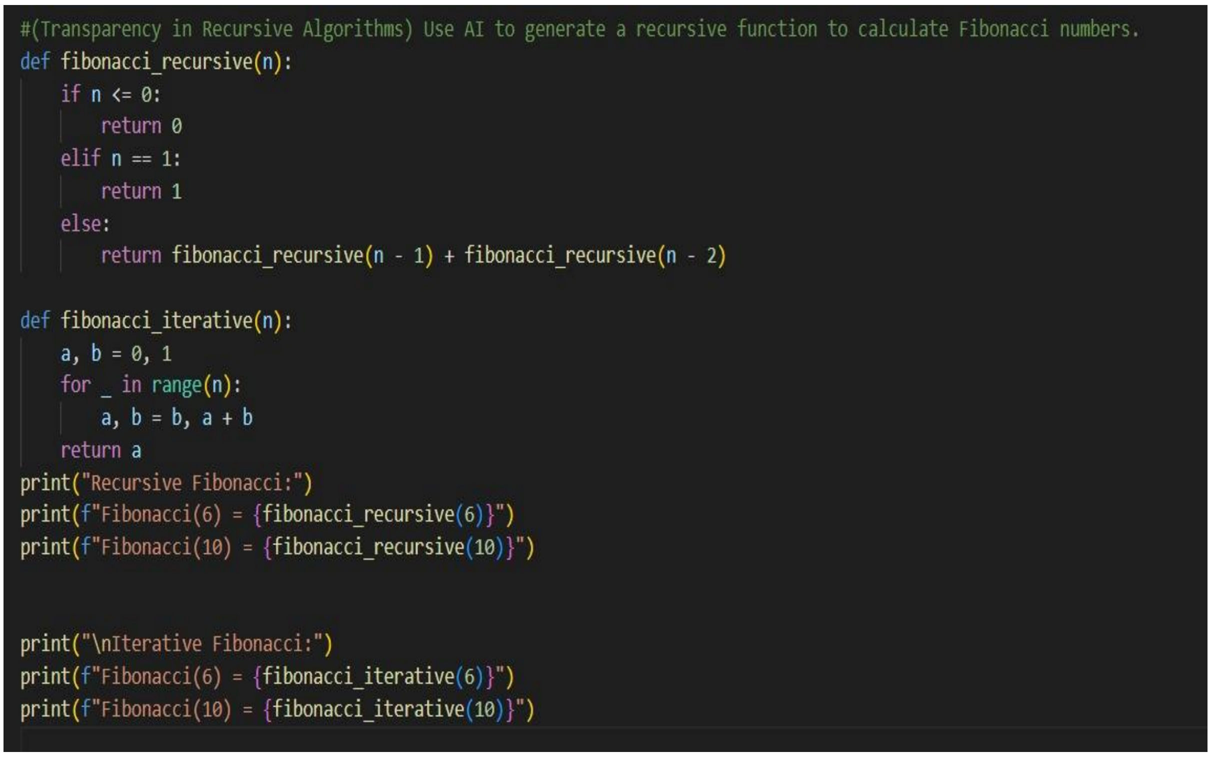
2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

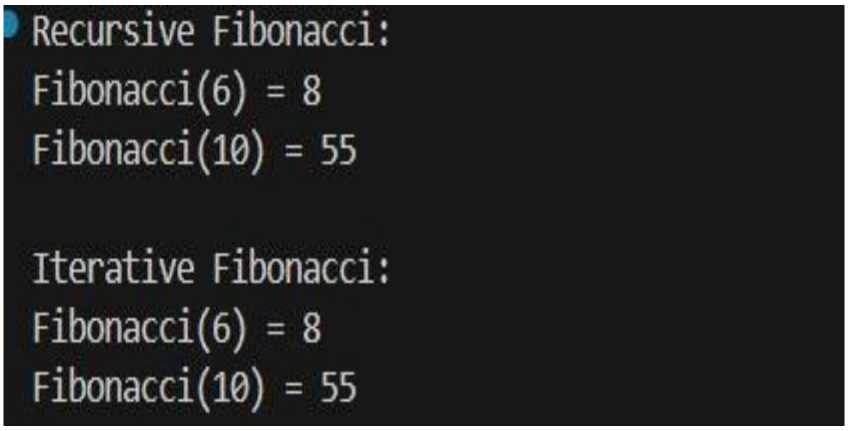
• Verification that explanation matches actual execution.



```
#(Transparency in Recursive Algorithms) Use AI to generate a recursive function to calculate Fibonacci numbers.
def fibonacci_recursive(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

def fibonacci_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
print("Recursive Fibonacci:")
print(f"Fibonacci(6) = {fibonacci_recursive(6)}")
print(f"Fibonacci(10) = {fibonacci_recursive(10)}")


print("\nIterative Fibonacci:")
print(f"Fibonacci(6) = {fibonacci_iterative(6)}")
print(f"Fibonacci(10) = {fibonacci_iterative(10)}")
```



```
Recursive Fibonacci:
Fibonacci(6) = 8
Fibonacci(10) = 55

Iterative Fibonacci:
Fibonacci(6) = 8
Fibonacci(10) = 55
```

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:

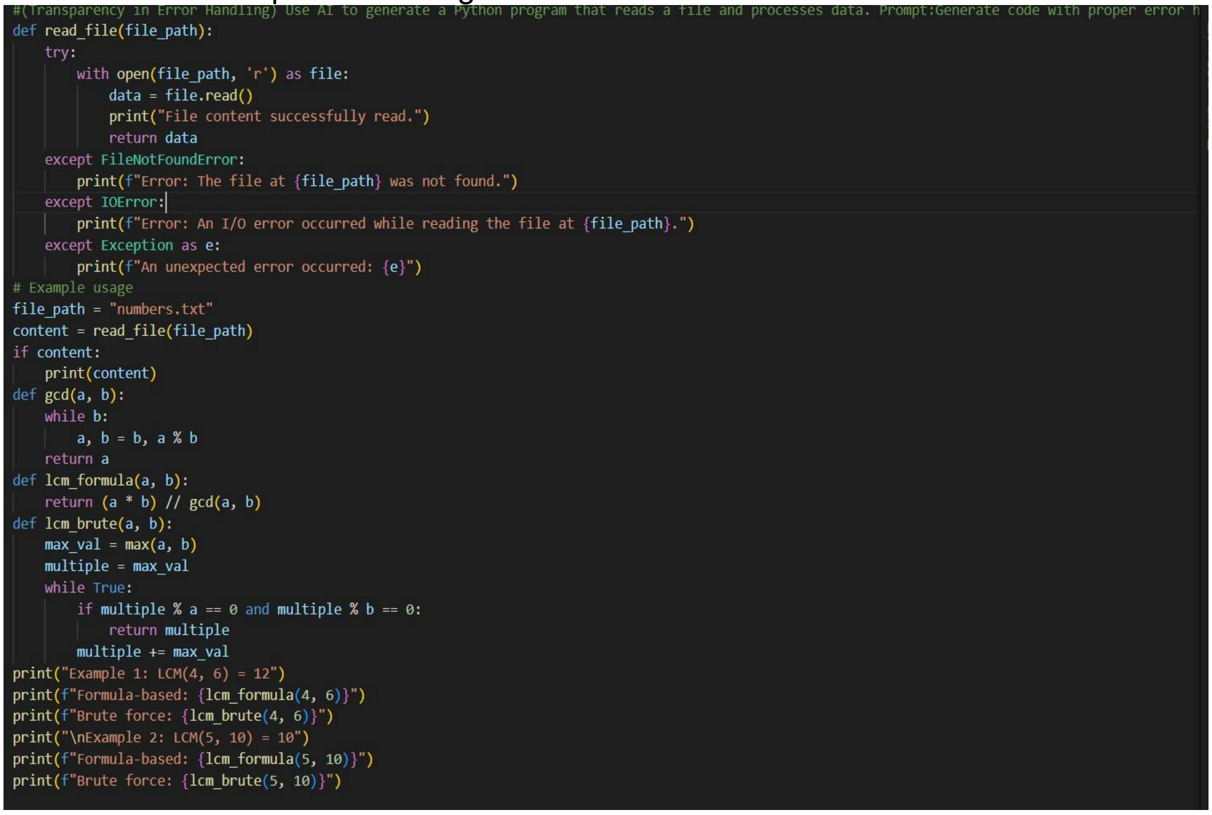
“Generate code with proper error handling and clear explanations

for each exception.” Expected Output:

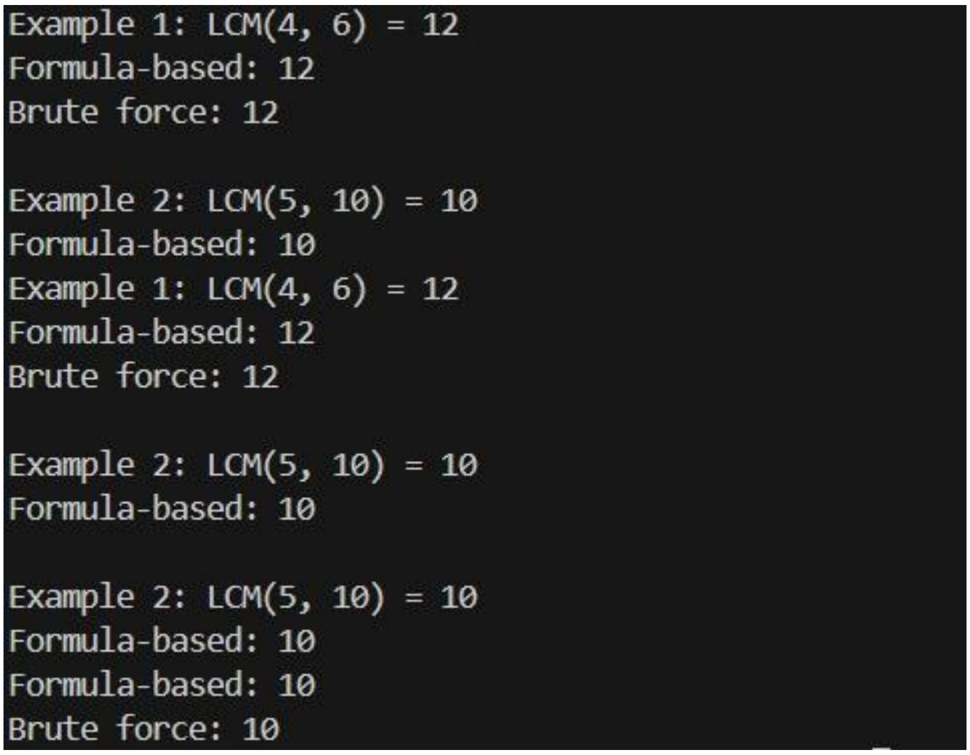
• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior.



```python
#(Transparency in Error Handling) Use AI to generate a Python program that reads a file and processes data. Prompt:Generate code with proper error h
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            data = file.read()
            print("File content successfully read.")
            return data
    except FileNotFoundError:
        print(f"Error: The file at {file_path} was not found.")
    except IOError:
        print(f"Error: An I/O error occurred while reading the file at {file_path}.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
# Example usage
file_path = "numbers.txt"
content = read_file(file_path)
if content:
    print(content)
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def lcm_formula(a, b):
    return (a * b) // gcd(a, b)
def lcm_brute(a, b):
    max_val = max(a, b)
    multiple = max_val
    while True:
        if multiple % a == 0 and multiple % b == 0:
            return multiple
        multiple += max_val
print("Example 1: LCM(4, 6) = 12")
print(f"Formula-based: {lcm_formula(4, 6)}")
print(f"Brute force: {lcm_brute(4, 6)}")
print("\nExample 2: LCM(5, 10) = 10")
print(f"Formula-based: {lcm_formula(5, 10)}")
print(f"Brute force: {lcm_brute(5, 10)}")
```



```
Example 1: LCM(4, 6) = 12
Formula-based: 12
Brute force: 12

Example 2: LCM(5, 10) = 10
Formula-based: 10
Example 1: LCM(4, 6) = 12
Formula-based: 12
Brute force: 12

Example 2: LCM(5, 10) = 10
Formula-based: 10

Example 2: LCM(5, 10) = 10
Formula-based: 10
Formula-based: 10
Brute force: 10
```

Explanation:

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

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

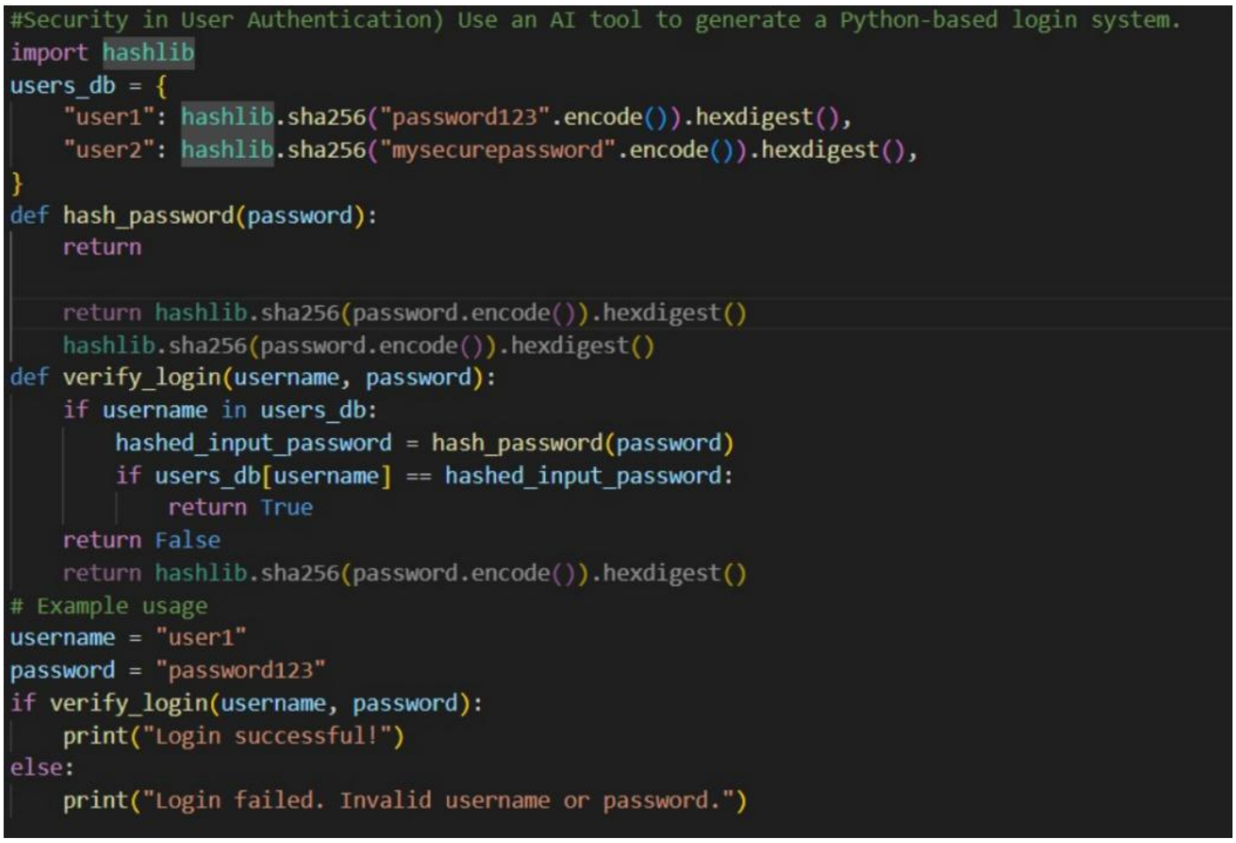
• Revised version using password hashing and input validation.

• Short note on best practices for secure authentication.

Explanation:

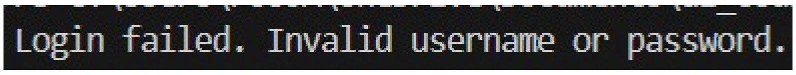
This program implements a **secure login system** using password hashing.

- 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.



```python
#Security in User Authentication) Use an AI tool to generate a Python-based login system.
import hashlib
users_db = {
    "user1": hashlib.sha256("password123".encode()).hexdigest(),
    "user2": hashlib.sha256("mysecurepassword".encode()).hexdigest(),
}
def hash_password(password):
    return

    return hashlib.sha256(password.encode()).hexdigest()
    hashlib.sha256(password.encode()).hexdigest()
def verify_login(username, password):
    if username in users_db:
        hashed_input_password = hash_password(password)
        if users_db[username] == hashed_input_password:
            return True
    return False
    return hashlib.sha256(password.encode()).hexdigest()
# Example usage
username = "user1"
password = "password123"
if verify_login(username, password):
    print("Login successful!")
else:
    print("Login failed. Invalid username or password.")
```



```
Login failed. Invalid username or password.
```

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).

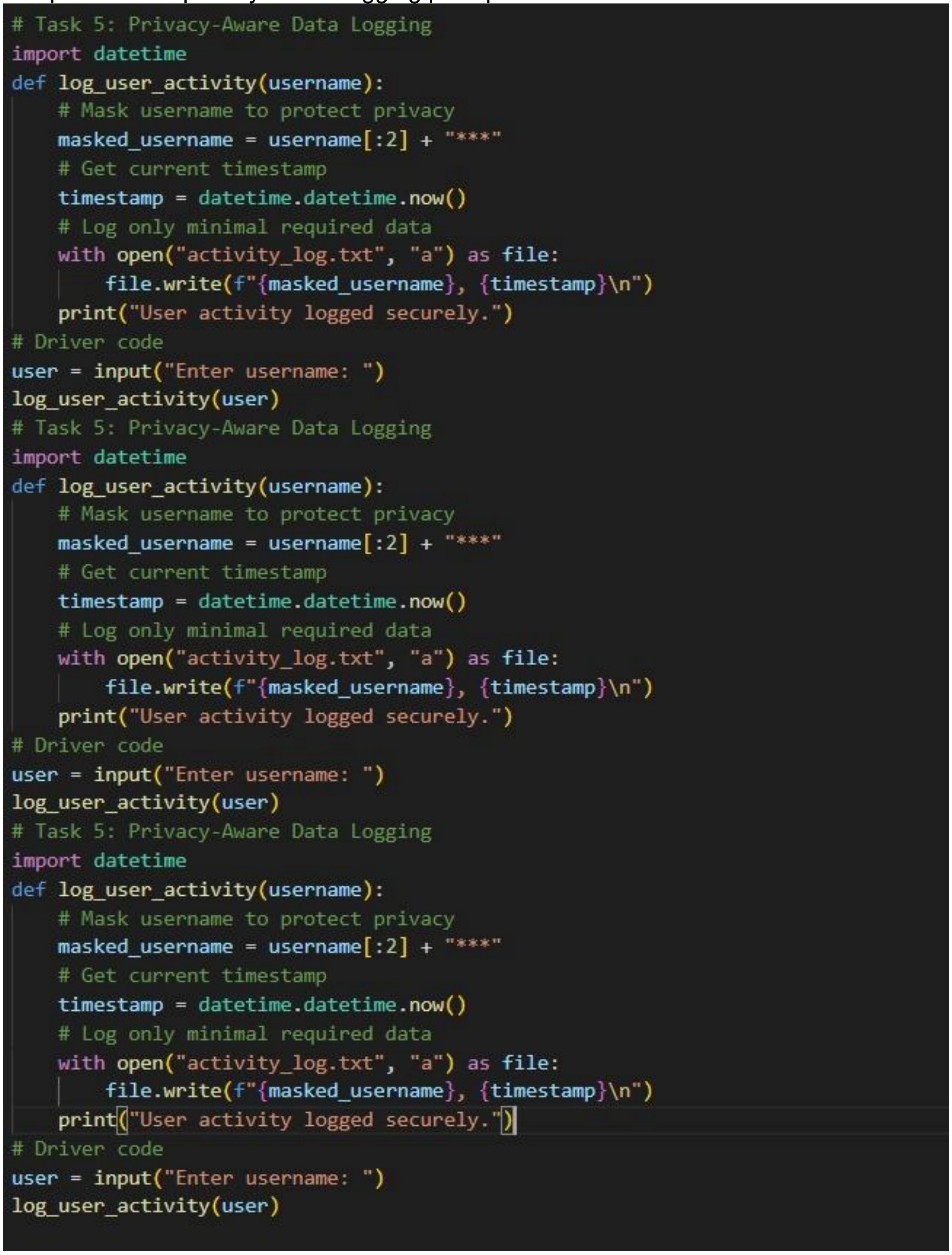
Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.

Expected Output:

• Identified privacy risks in logging.

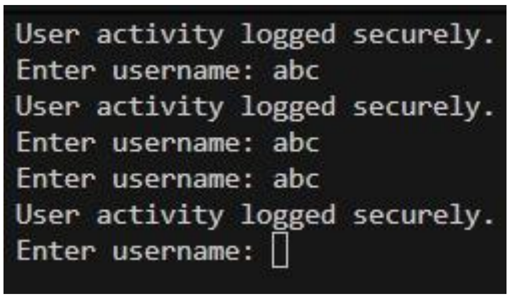
• Improved version with minimal, anonymized, or masked logging.

• Explanation of privacy-aware logging principles.



```python
# Task 5: Privacy-Aware Data Logging
import datetime
def log_user_activity(username):
    # Mask username to protect privacy
    masked_username = username[:2] + "***"
    # Get current timestamp
    timestamp = datetime.datetime.now()
    # Log only minimal required data
    with open("activity_log.txt", "a") as file:
        file.write(f"{masked_username}, {timestamp}\n")
    print("User activity logged securely.")
# Driver code
user = input("Enter username: ")
log_user_activity(user)
```

```python
# Task 5: Privacy-Aware Data Logging
import datetime
def log_user_activity(username):
    # Mask username to protect privacy
    masked_username = username[:2] + "***"
    # Get current timestamp
    timestamp = datetime.datetime.now()
    # Log only minimal required data
    with open("activity_log.txt", "a") as file:
        file.write(f"{masked_username}, {timestamp}\n")
    print("User activity logged securely.")
# Driver code
user = input("Enter username: ")
log_user_activity(user)
# Task 5: Privacy-Aware Data Logging
import datetime
def log_user_activity(username):
    # Mask username to protect privacy
    masked_username = username[:2] + "***"
    # Get current timestamp
    timestamp = datetime.datetime.now()
    # Log only minimal required data
    with open("activity_log.txt", "a") as file:
        file.write(f"{masked_username}, {timestamp}\n")
    print("User activity logged securely.")
# Driver code
user = input("Enter username: ")
log_user_activity(user)
```



```
User activity logged securely.
Enter username: abc
User activity logged securely.
Enter username: abc
Enter username: abc
User activity logged securely.
Enter username: 
```

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.