

AI ASSISTANT CODING

ASSIGNMENT-5.5

Name: B. Akhira Nandhini

Hallticket:2303A51516

Batch:22

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.

Code:

The screenshot shows a code editor interface with a dark theme. The left sidebar lists files in a tree view under 'AI ASSISTED CODING'. The current file is 'task1.py', which contains the following Python code:

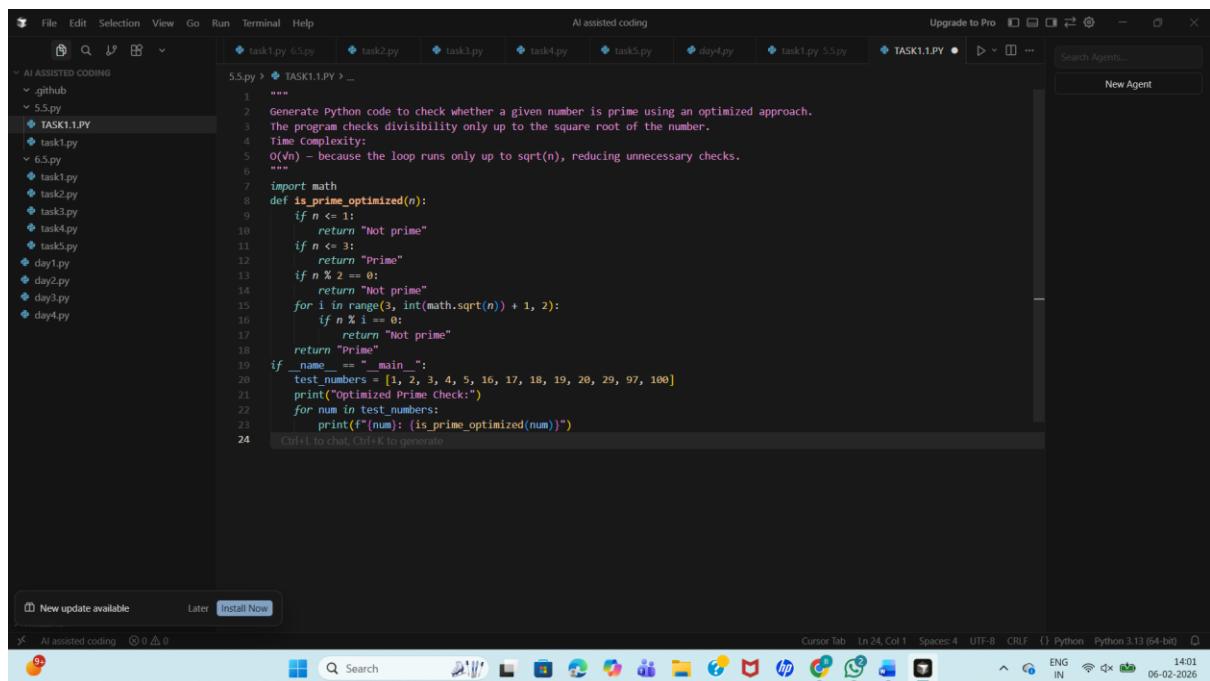
```
task1.py > task1.py > ...
1  '''Generate Python code to check whether a given number is prime using a naive approach.
2  The program should test divisibility by all numbers from 2 to n and explain the time complexity.'''
3  #Method 1: Basic (Naive) prime-checking method
4  def is_prime_basic(n):
5      if n <= 1:
6          return "Not prime"
7      for i in range(2, n):
8          if n % i == 0:
9              return "Not prime"
10     return "Prime"
11 if __name__ == "__main__":
12     test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 29, 97, 100]
13
14     print("Basic Prime Check:")
15     for num in test_numbers:
16         print(f"{num}: {is_prime_basic(num)}")
```

The code defines a function `is_prime_basic` that checks if a number is prime by testing divisibility from 2 to $n-1$. It includes a main block to test the function with various numbers.

Output:

```
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:\Users\abhin\AppData\Local\Programs\Python\Python313\python.exe\Drive\Desktop/AI assisted coding/5.5.py/task1.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
97: Prime
100: Not prime
```

METHOD 2:



The screenshot shows a code editor window titled "AI assisted coding". The left sidebar lists files under "AI ASSISTED CODING" including .github, 5.5.py, and several task and day files. The main pane displays the content of TASK1.1.PY. The code is as follows:

```
5.5.py > ◆ TASK1.1.PY > ...
1 """
2 Generate Python code to check whether a given number is prime using an optimized approach.
3 The program checks divisibility only up to the square root of the number.
4 Time Complexity:
5 O(√n) - because the loop runs only up to sqrt(n), reducing unnecessary checks.
6 """
7 import math
8 def is_prime_optimized(n):
9     if n <= 1:
10         return "Not prime"
11     if n <= 3:
12         return "prime"
13     if n % 2 == 0:
14         return "Not prime"
15     for i in range(3, int(math.sqrt(n)) + 1, 2):
16         if n % i == 0:
17             return "Not prime"
18     return "prime"
19 if __name__ == "__main__":
20     test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 29, 97, 100]
21     print("Optimized Prime Check:")
22     for num in test_numbers:
23         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
20: Not prime
29: Prime
97: Prime
100: Not prime
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> 
```

Explanation:

- In the naive approach, the program checks whether a number is prime by testing its divisibility with **all integers from 2 to n-1**.
If the number is divisible by any of these values, it is not a prime number; otherwise, it is considered prime.
- This method is easy to understand and implement, but it performs a large number of unnecessary checks, especially for bigger numbers.
- In the optimized approach, the program checks divisibility **only up to the square root of the number**.
This is based on the mathematical fact that if a number has a factor greater than \sqrt{n} , it must also have a corresponding factor smaller than \sqrt{n} .
- Additionally, even numbers are skipped after checking divisibility by 2, which further reduces the number of iterations

Task Description #2 (Transparency in Recursive Algorithms)

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

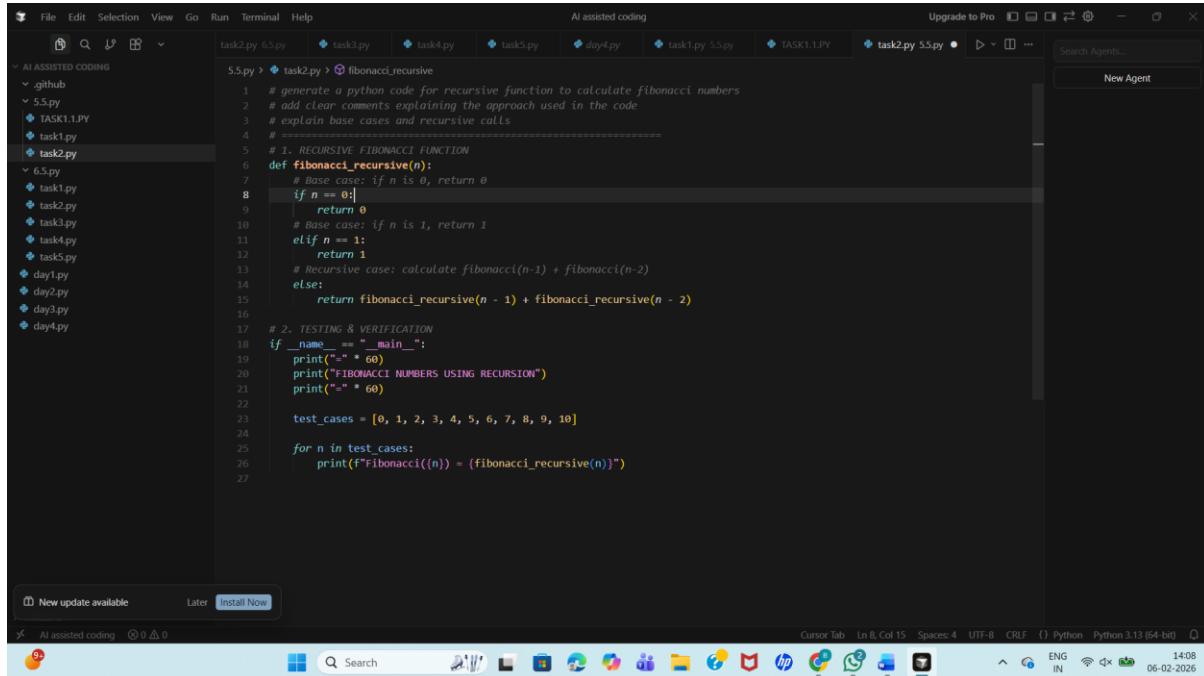
Instructions:

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:



```

File Edit Selection View Go Run Terminal Help
task2.py 6.5.py task3.py task4.py task5.py day4.py task1.py 5.5.py TASK1.1.PY task2.py 5.5.py
AI assisted coding
task2.py > fibonacci_recursive
1 # generate a python code for recursive function to calculate fibonacci numbers
2 # add clear comments explaining the approach used in the code
3 # explain base cases and recursive calls
4 #
5 # 1. RECURSIVE FIBONACCI FUNCTION
6 def fibonacci_recursive(n):
7     # Base case: if n is 0, return 0
8     if n == 0:
9         return 0
10    # Base case: if n is 1, return 1
11    elif n == 1:
12        return 1
13    # Recursive case: calculate fibonacci(n-1) + fibonacci(n-2)
14    else:
15        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
16
17 # 2. TESTING & VERIFICATION
18 if __name__ == "__main__":
19     print("+" * 60)
20     print("FIBONACCI NUMBERS USING RECURSION")
21     print("+" * 60)
22
23 test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
24
25 for n in test_cases:
26     print(Fibonacci((n)) = {fibonacci_recursive(n)})
27

```

Output:

```

PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:\Users\abhi
eDrive/Desktop/AI assisted coding/5.5.py/task2.py"
=====
FIBONACCI NUMBERS USING RECURSION
=====
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>

```

Explanation:

The Fibonacci series is a sequence of numbers in which each number is the sum of the previous two numbers.

The sequence starts as: **0, 1, 1, 2, 3, 5, 8, ...**

This program uses **recursion**, where a function calls itself to solve a problem by breaking it into smaller subproblems.

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:

The screenshot shows a dark-themed AI-assisted coding interface. On the left, a sidebar lists files: .github, 5.5.py, TASK1.1.PY, task1.py, task2.py, task3.py, Untitled, and several day*.py files. The main area displays a Python script with code for exception handling. The code defines two functions: `divide_numbers(a, b)` and `access_list_element(lst, index)`. Both functions include detailed docstrings explaining their approach to handling errors like division by zero, index out of range, and invalid types. The interface includes tabs for 'AI assisted coding' and 'Upgrades', and a status bar at the bottom.

```
# 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:
        return "Error: Invalid index type. Please provide an integer."
    else:
        return element
```

The screenshot shows a software interface for 'AI assisted coding'. The top menu includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A toolbar below has icons for search, refresh, and file operations. The left sidebar lists files under 'AI ASSISTED CODING' such as .github, 5.5.py, TASK1.1.PY, task1.py, task2.py, task3.py, Untitled, and B.5.py. The main area displays Python code for exception handling and division tests. The bottom status bar shows system information like cursor position, language, and date.

```
5.5.py > Untitled > ...
31     user access_list_element(lst, index):
32         ...
33
34     # 2. TESTING & VERIFICATION
35     if name == "main__":
36         print("+" * 60)
37         print("EXCEPTION HANDLING DEMONSTRATION")
38         print("+" * 60)
39
40         # Test division
41         print("Division Tests:")
42         print(divide_numbers(10, 2))      # Valid
43         print(divide_numbers(10, 0))      # ZeroDivisionError
44         print(divide_numbers(10, "a"))    # TypeError
45
46         print("\nList Access Tests:")
47         sample_list = [10, 20, 30]
48         print(access_list_element(sample_list, 1))  # Valid
49         print(access_list_element(sample_list, 5))  # IndexError
50         print(access_list_element(sample_list, "x")) # TypeError
51
52         print("+" * 60)
53
```

Output:

The terminal output shows the execution of the Python script. It starts with the command PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:/eDrive/Desktop/AI assisted coding/5.5.py/Untitled". The output then displays the 'EXCEPTION HANDLING DEMONSTRATION' and 'Division Tests:' sections, followed by error messages for division by zero and invalid input type. Finally, it shows the 'List Access Tests:' section with an error for index out of range and invalid index type, before ending with PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>.

```
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:/eDrive/Desktop/AI assisted coding/5.5.py/Untitled"
=====
EXCEPTION HANDLING DEMONSTRATION
=====
Division Tests:
5.0
Error: Division by zero is not allowed.
Error: Invalid input type. Please provide numbers.

List Access Tests:
20
Error: Index out of range.
Error: Invalid index type. Please provide an integer.
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>
```

Explanation:

Exception handling is a mechanism in Python used to handle runtime errors so that the program does not crash and can continue executing smoothly. It helps in writing reliable and user-friendly programs.

Task Description #4 (Security in User Authentication)

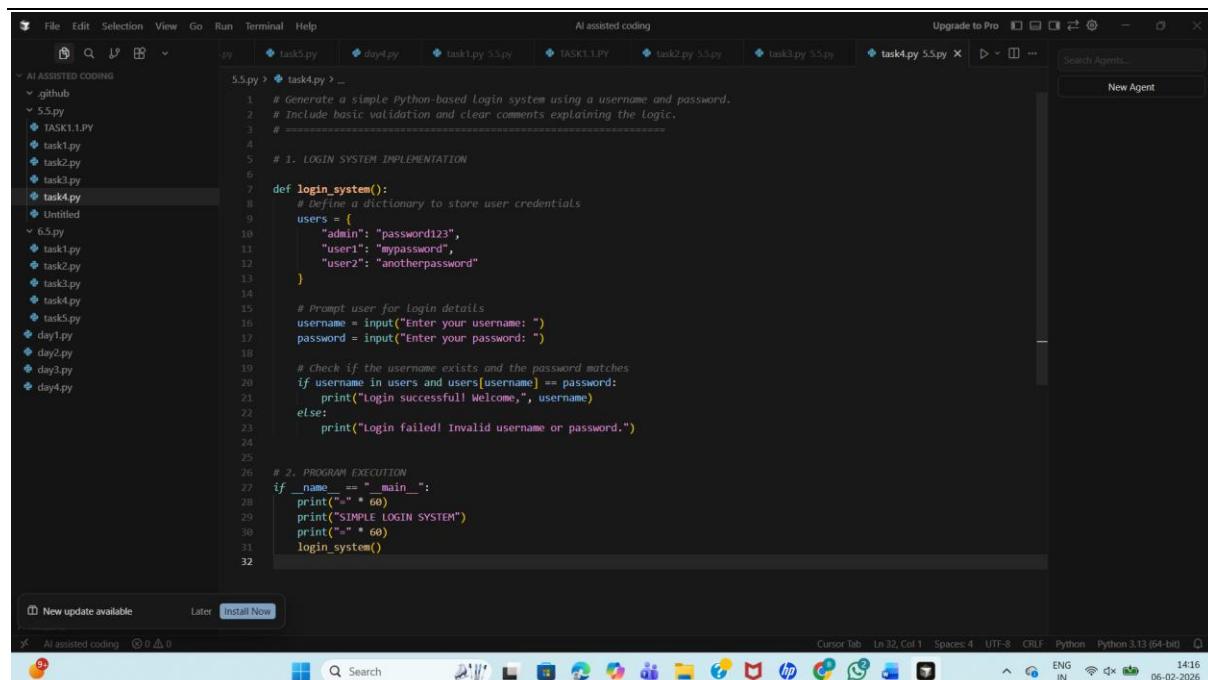
Task: Use an AI tool to generate a Python-based login system.

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:



The screenshot shows a code editor interface with multiple tabs open. The active tab is 'task4.py'. The code implements a simple login system. It defines a dictionary of users with their passwords and prompts the user for their credentials. It then checks if the entered username and password match any in the dictionary. If they do, it prints a welcome message; otherwise, it prints an error message. The code is well-commented, explaining each step of the implementation.

```
# Generate a simple Python-based Login system using a username and password.
# Include basic validation and clean comments explaining the logic.
# =====
# 1. LOGIN SYSTEM IMPLEMENTATION

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

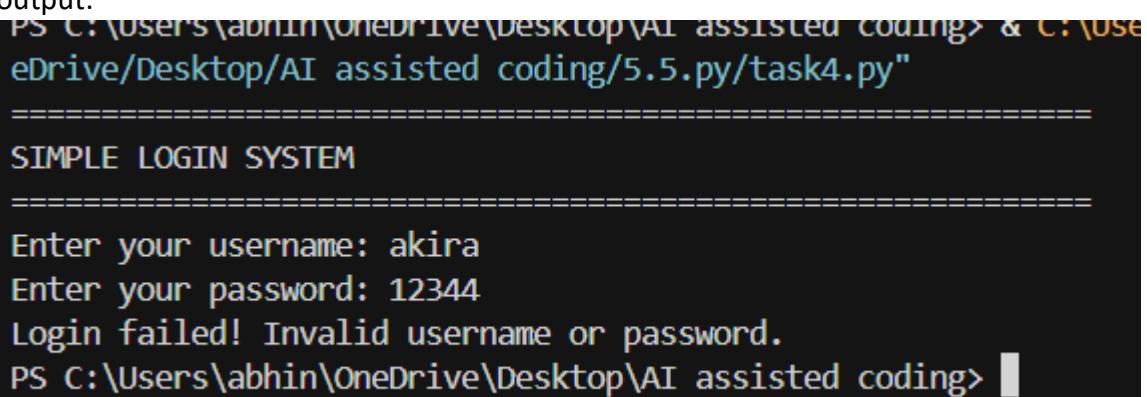
    # Prompt user for login details
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    # Check if the username exists and the password matches
    if username in users and users[username] == password:
        print("Login successful! Welcome, ", username)
    else:
        print("Login failed! Invalid username or password.")

# 2. PROGRAM EXECUTION
if __name__ == "__main__":
    print("=" * 60)
    print("SIMPLE LOGIN SYSTEM")
    print("=" * 60)
    login_system()

# New update available Later Install Now
# AI assisted coding 0 0
# Search Cursor Tab Ln 32, Col 1 Spaces: 4 UTF-8 CR LF Python Python 3.13 (64-bit) 14:16 ENG IN 06-02-2026
```

output:



The terminal window shows the execution of the Python script 'task4.py'. It prints the title 'SIMPLE LOGIN SYSTEM' and then prompts the user for their username and password. When 'akira' is entered as the username and '12344' as the password, it prints an error message: 'Login failed! Invalid username or password.'

```
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:\use
eDrive\Desktop\AI assisted coding\5.5.py\task4.py
=====
SIMPLE LOGIN SYSTEM
=====
Enter your username: akira
Enter your password: 12344
Login failed! Invalid username or password.
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>
```

Explanation: This program implements a basic login system using Python. It verifies a user by checking the entered username and password against stored credentials.

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).

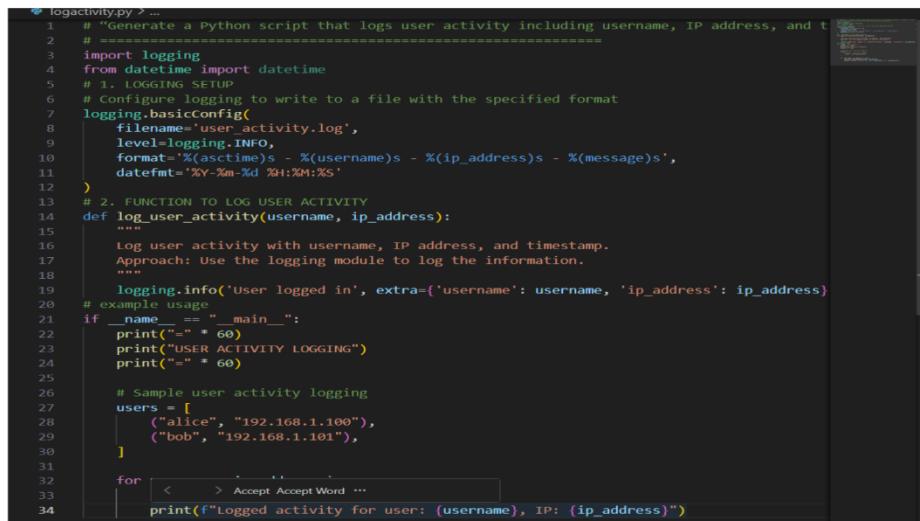
Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.

Expected Output:

- Identified privacy risks in logging.
- Improved version with minimal, anonymized, or masked logging.

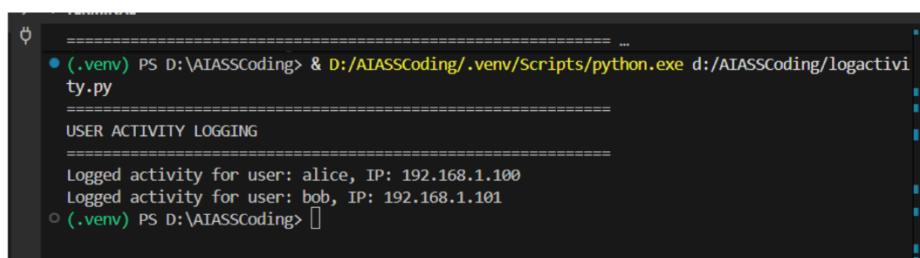
- Explanation of privacy-aware logging principles.

CODE :



```
# logactivity.py >...
1 # "Generate a Python script that logs user activity including username, IP address, and timestamp.
2 # =====
3 import logging
4 from datetime import datetime
5 # 1. LOGGING SETUP
6 # Configure logging to write to a file with the specified format
7 logging.basicConfig(
8     filename='user_activity.log',
9     level=logging.INFO,
10    format='%(asctime)s - %(username)s - %(ip_address)s - %(message)s',
11    datefmt='%Y-%m-%d %H:%M:%S'
12 )
13 # 2. FUNCTION TO LOG USER ACTIVITY
14 def log_user_activity(username, ip_address):
15     """
16     Log user activity with username, IP address, and timestamp.
17     Approach: Use the logging module to log the information.
18     """
19     logging.info('User logged in', extra={'username': username, 'ip_address': ip_address})
20 # example usage
21 if __name__ == "__main__":
22     print("=" * 60)
23     print("USER ACTIVITY LOGGING")
24     print("=" * 60)
25
26     # Sample user activity logging
27     users = [
28         ("alice", "192.168.1.100"),
29         ("bob", "192.168.1.101"),
30     ]
31
32     for [ <   > Accept Accept Word ... ]:
33         print(f"Logged activity for user: {username}, IP: {ip_address}")
34
```

OUTPUT :



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