ASSIGNMENT - 5.5

Batch No. : 2303A510F7
H.T.No. : 03

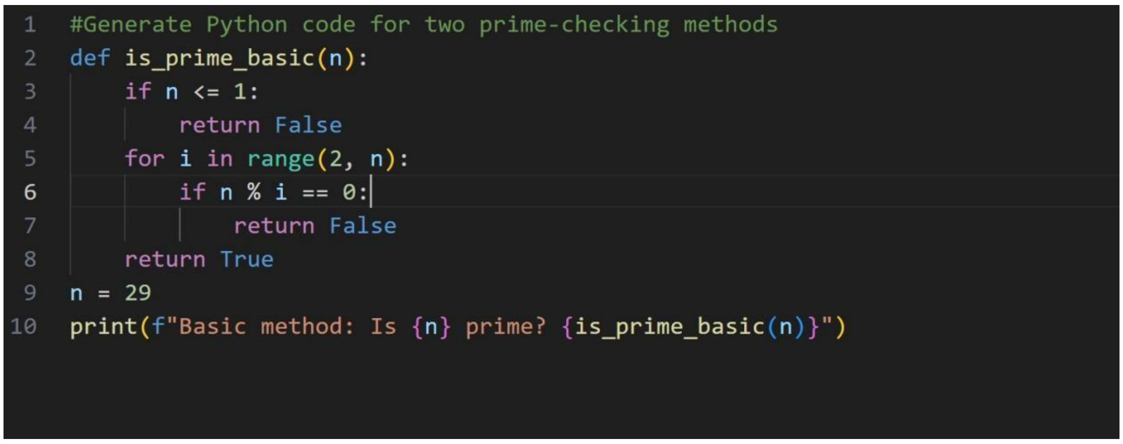
Task Description #1 (Transparency in Algorithm Optimization)
Task: Use AI to generate two solutions for checking prime numbers:

- **Naive approach(basic)**
- **Optimized approach**

Prompt:
"Generate Python code for two prime-checking methods and explain how the optimized version improves performance."

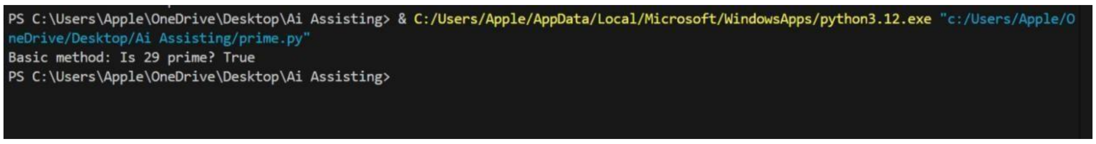
Approach - 1 : Naive approach(basic)

Code :



```python
#Generate Python code for two prime-checking methods
def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
n = 29
print(f"Basic method: Is {n} prime? {is_prime_basic(n)}")
```

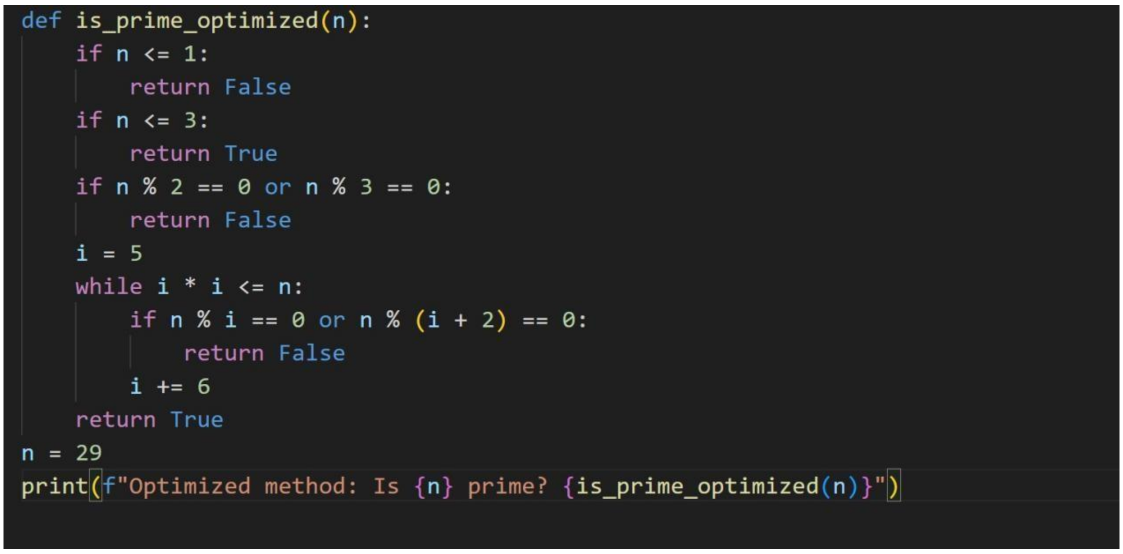
Output :



```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/O
neDrive/Desktop/Ai Assisting/prime.py"
Basic method: Is 29 prime? True
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting>
```

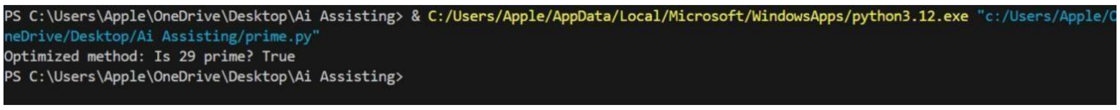
Approach - 2 : Optimized approach

Code :



```python
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
n = 29
print(f"Optimized method: Is {n} prime? {is_prime_optimized(n)}")
```

Output :



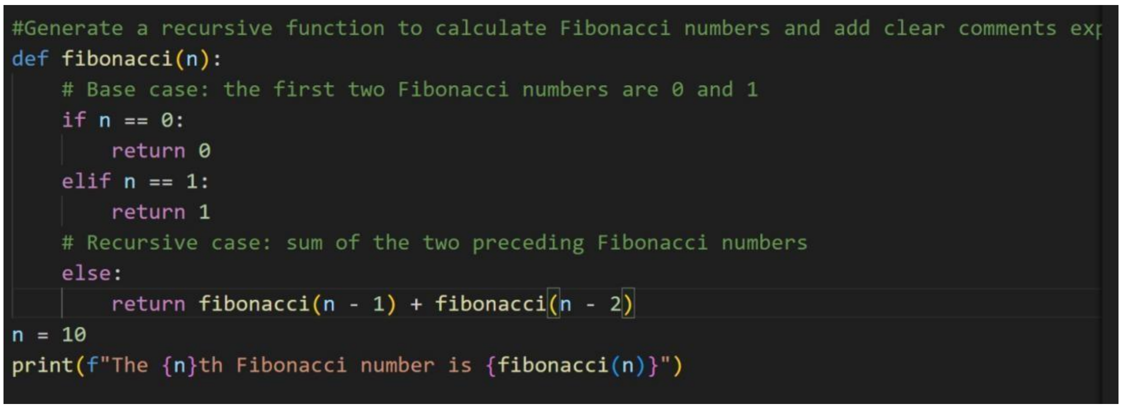
```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/O
neDrive/Desktop/Ai Assisting/prime.py"
Optimized method: Is 29 prime? True
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting>
```

Code Explanation:

The basic prime checking method tests whether a number n is divisible by any integer from 2 to n−1. If it finds a divisor, the number is not prime; otherwise, it is prime. This approach is simple but inefficient because it performs many unnecessary checks, especially for large numbers, resulting in a time complexity of O(n). The optimized prime checking method improves efficiency by first handling small cases, eliminating even numbers and multiples of 3, and then checking divisibility only up to the square root of n. It further reduces checks by testing only numbers of the form 6k ± 1, which are the only possible candidates for primes greater than 3. This significantly reduces the number of iterations, giving a faster time complexity of O(√n), making it suitable for larger inputs.

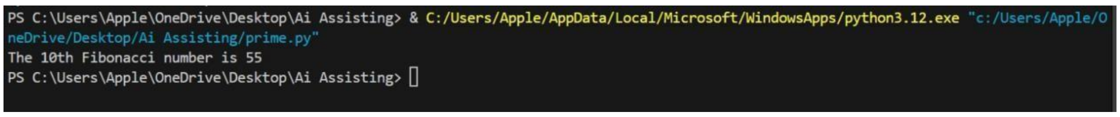
Task Description #2 (Transparency in Recursive Algorithms)
Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Code :



```
#Generate a recursive function to calculate Fibonacci numbers and add clear comments exp
def fibonacci(n):
    # Base case: the first two Fibonacci numbers are 0 and 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # Recursive case: sum of the two preceding Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
n = 10
print(f"The {n}th Fibonacci number is {fibonacci(n)}")
```

Output :



```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/O
neDrive/Desktop/Ai Assisting/prime.py"
The 10th Fibonacci number is 55
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> []
```

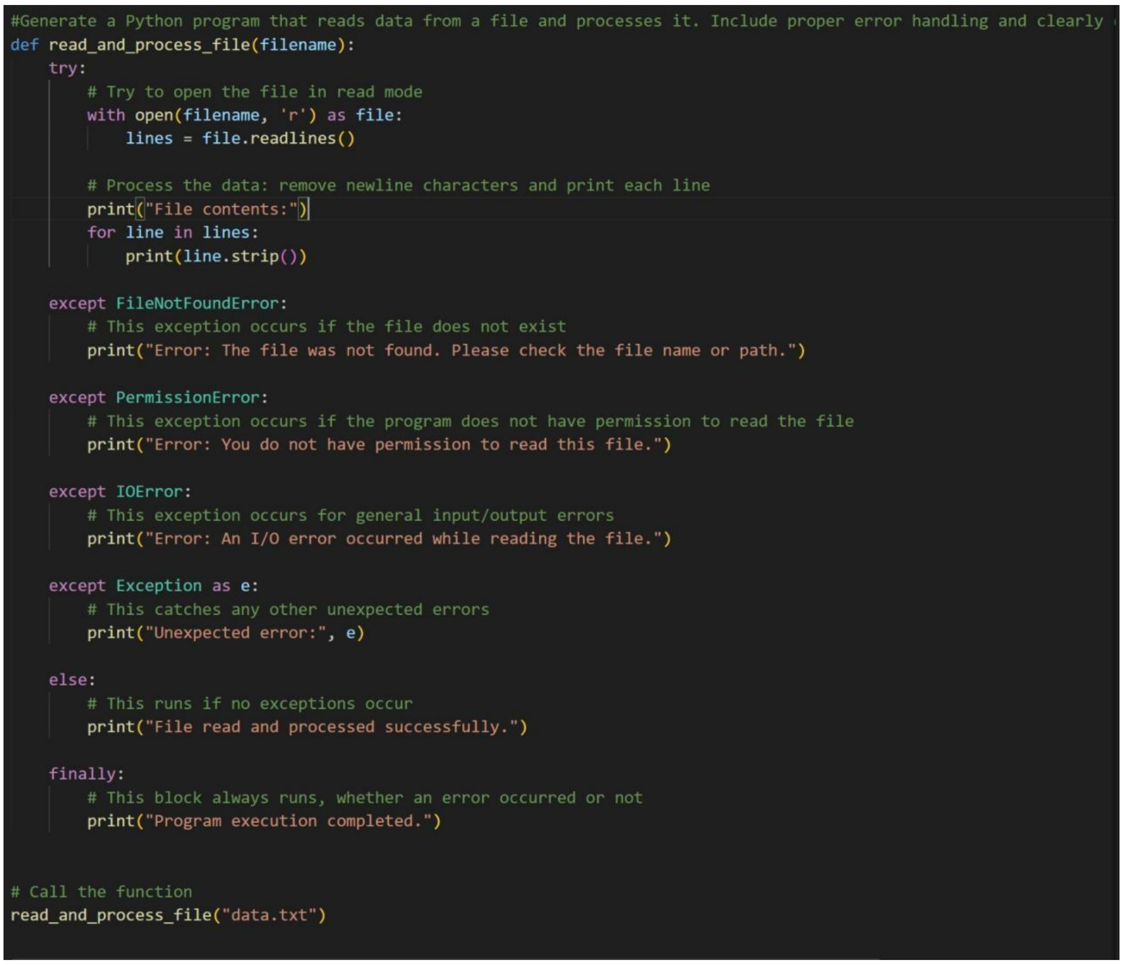
Code Explanation :

This Fibonacci function uses recursion to find the nth Fibonacci number. The function first checks the base cases: if `n` is `0`, it returns `0`, and if `n` is `1`, it returns `1`, since these are the first two Fibonacci numbers. For any other value of `n`, the function calls itself recursively to compute the sum of the previous two Fibonacci numbers, `fibonacci(n−1)` and `fibonacci(n−2)`. This follows the Fibonacci definition where each number is the sum of the two before it. Finally, when `n = 10`, the function calculates and prints the 10th Fibonacci number.

Task Description #3 (Transparency in Error Handling)
Task: Use AI to generate a Python program that reads a file and processes data.

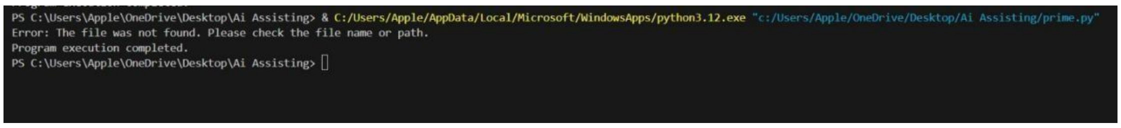
Prompt:
“Generate code with proper error handling and clear explanations for each exception.”

Code :



```python
#Generate a Python program that reads data from a file and processes it. Include proper error handling and clearly
def read_and_process_file(filename):
    try:
        # Try to open the file in read mode
        with open(filename, 'r') as file:
            lines = file.readlines()

        # Process the data: remove newline characters and print each line
        print("File contents:")
        for line in lines:
            print(line.strip())

    except FileNotFoundError:
        # This exception occurs if the file does not exist
        print("Error: The file was not found. Please check the file name or path.")

    except PermissionError:
        # This exception occurs if the program does not have permission to read the file
        print("Error: You do not have permission to read this file.")

    except IOError:
        # This exception occurs for general input/output errors
        print("Error: An I/O error occurred while reading the file.")

    except Exception as e:
        # This catches any other unexpected errors
        print("Unexpected error:", e)

    else:
        # This runs if no exceptions occur
        print("File read and processed successfully.")

    finally:
        # This block always runs, whether an error occurred or not
        print("Program execution completed.")


# Call the function
read_and_process_file("data.txt")
```

Output :



```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/OneDrive/Desktop/Ai Assisting/prime.py"
Error: The file was not found. Please check the file name or path.
Program execution completed.
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> 
```

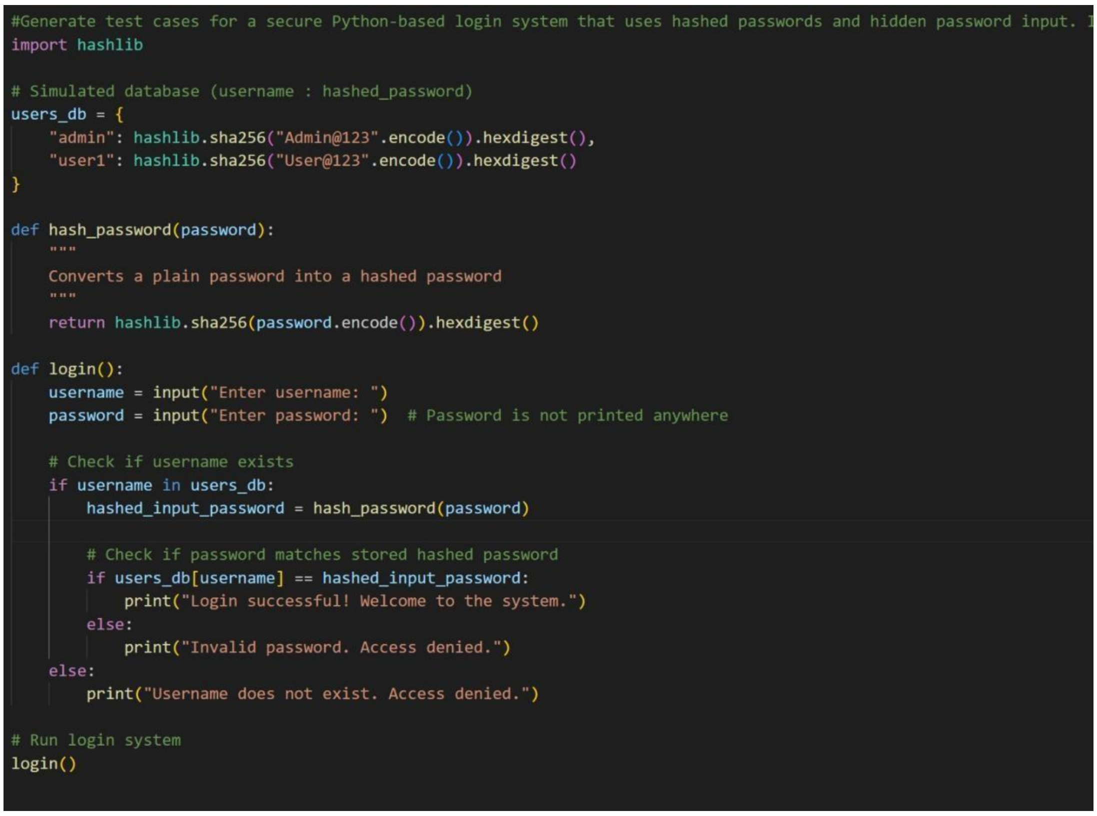
Code Explanation :

This program reads and processes a file using proper exception handling to ensure safe execution. Inside the `try` block, the function attempts to open the given file in read mode and reads all lines from it. Each line is then printed after removing extra newline characters using `strip()`. If the file does not exist, a `FileNotFoundError` is raised, and the program displays a message asking the user to verify the file name or path. If the file exists but the program does not have permission to read it, a `PermissionError` is handled with a clear error message. The `IOError` block catches general input/output errors that may occur during file operations. Any other unexpected issues are handled by the generic `Exception` block, which prints the error details. If the file is read successfully without any exceptions, the `else` block confirms successful processing. Finally, the `finally` block always executes, ensuring that the program reports completion regardless of whether an error occurred or not.

Task Description #4 (Security in User Authentication)
Task: Use an AI tool to generate a Python-based login system.

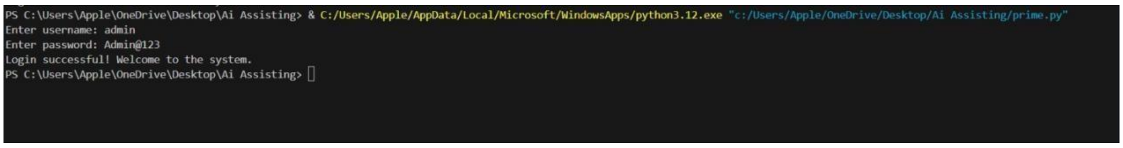
Prompt : Generate test cases for a secure Python-based login system that uses hashed passwords and hidden password input. Include valid and invalid login scenarios, edge cases, and expected outputs.

Code :



```python
#Generate test cases for a secure Python-based login system that uses hashed passwords and hidden password input. I
import hashlib

# Simulated database (username : hashed_password)
users_db = {
    "admin": hashlib.sha256("Admin@123".encode()).hexdigest(),
    "user1": hashlib.sha256("User@123".encode()).hexdigest()
}

def hash_password(password):
    """
    Converts a plain password into a hashed password
    """
    return hashlib.sha256(password.encode()).hexdigest()

def login():
    username = input("Enter username: ")
    password = input("Enter password: ")  # Password is not printed anywhere

    # Check if username exists
    if username in users_db:
        hashed_input_password = hash_password(password)

        # Check if password matches stored hashed password
        if users_db[username] == hashed_input_password:
            print("Login successful! Welcome to the system.")
        else:
            print("Invalid password. Access denied.")
    else:
        print("Username does not exist. Access denied.")

# Run login system
login()
```

Output :



```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/OneDrive/Desktop/Ai Assisting/prime.py"
Enter username: admin
Enter password: Admin@123
Login successful! Welcome to the system.
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> 
```

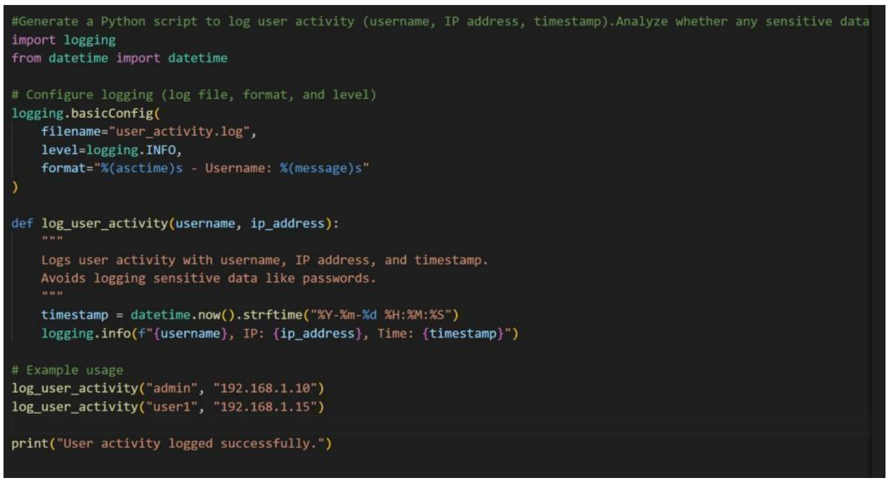
Code Explanation :

This program improves login security by using password hashing instead of plain-text passwords, which prevents attackers from seeing real passwords even if the database is leaked. It also checks whether a username exists before validating the password, avoiding weak validation. The hashed input password is compared with the stored hash, not the original password. As a best practice, secure authentication systems should always hash (and salt) passwords, never display or store passwords in plain text, and validate user input properly to reduce security risks.

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).

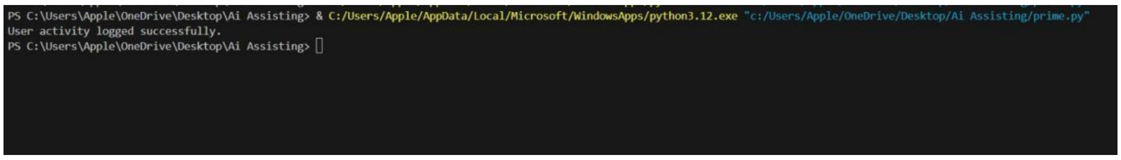
Prompt : Generate a Python script to log user activity (username, IP address, timestamp). Analyze whether any sensitive data is logged unnecessarily or insecurely.

Code :



```python
#Generate a Python script to log user activity (username, IP address, timestamp).Analyze whether any sensitive data
import logging
from datetime import datetime

# Configure logging (log file, format, and level)
logging.basicConfig(
    filename="user_activity.log",
    level=logging.INFO,
    format="%(asctime)s - Username: %(message)s"
)

def log_user_activity(username, ip_address):
    """
    Logs user activity with username, IP address, and timestamp.
    Avoids logging sensitive data like passwords.
    """
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    logging.info(f"{username}, IP: {ip_address}, Time: {timestamp}")

# Example usage
log_user_activity("admin", "192.168.1.10")
log_user_activity("user1", "192.168.1.15")

print("User activity logged successfully.")
```

Output :



```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/OneDrive/Desktop/Ai Assisting/prime.py"
User activity logged successfully.
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> 
```

Code Explanation :

This program logs usernames and IP addresses, which can create privacy risks if log files are exposed. A better approach is to log minimal or masked information (for example, hiding part of the IP address) and avoid personal details. Privacy-aware logging follows the rule of log only what is necessary, never log sensitive data, and protect log files from unauthorized access.