AI Assisted Coding

Assignment 5.5

Name: Dinesh

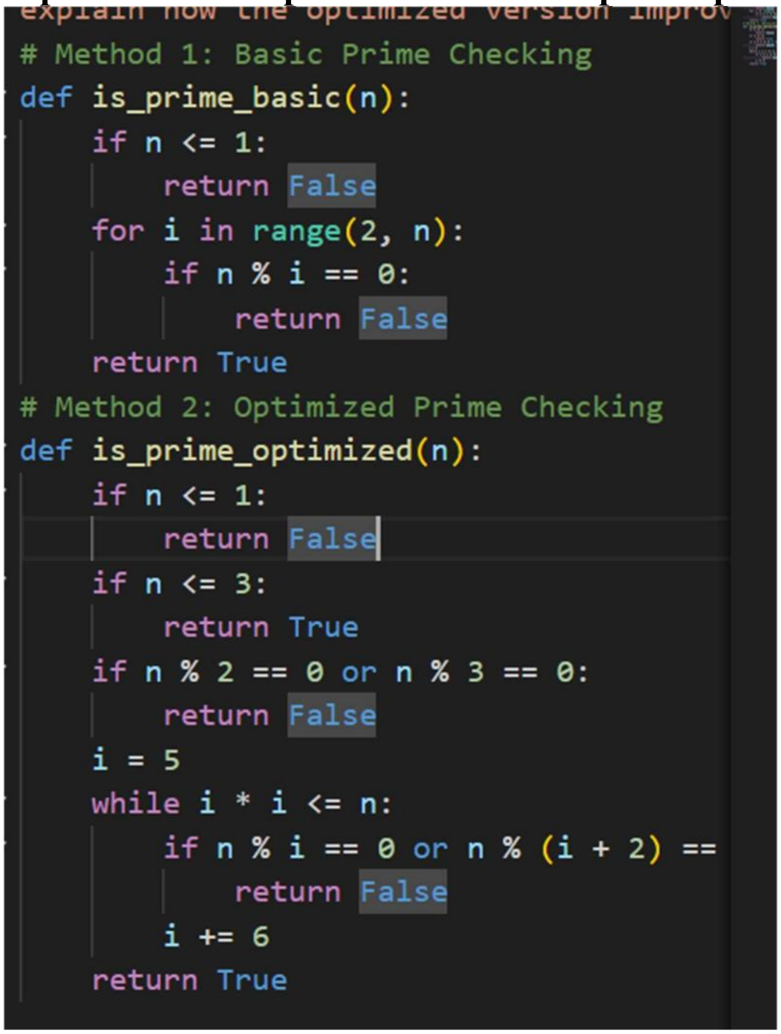
Hallticket:2303A52329

Task-01:

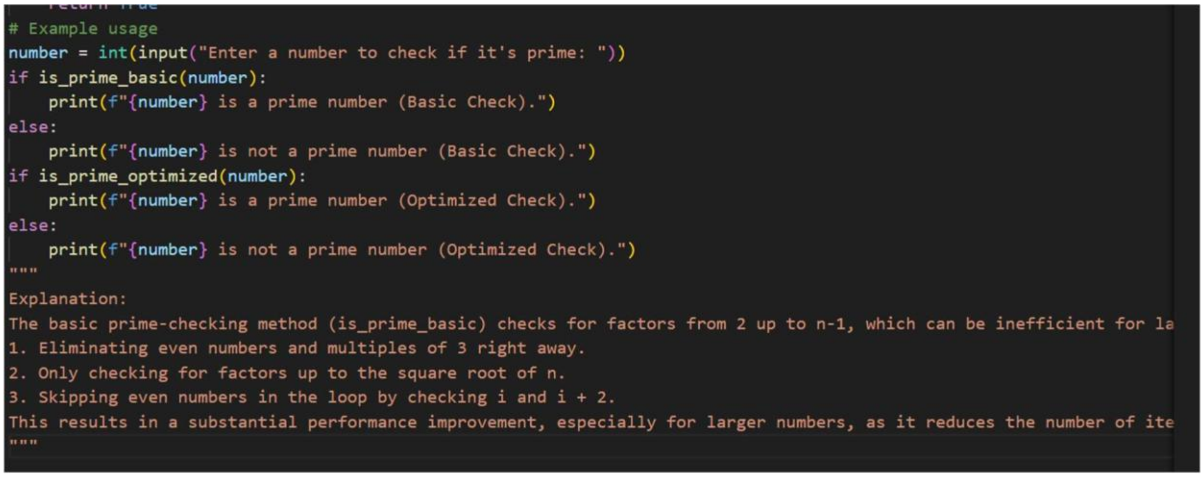
Use AI to generate two solutions for checking prime numbers:

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

Prompt: Hallticket:2303A52009
Generate Python code for two prime-checking methods and explain how the optimized version improves performance."""

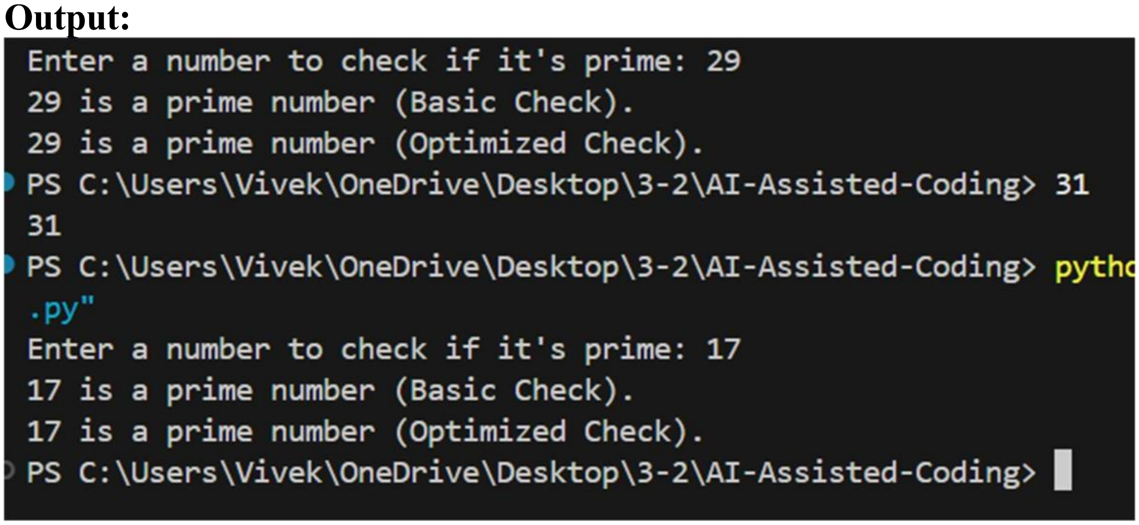


```
explain how the optimized version improv
# Method 1: Basic Prime Checking
def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
# Method 2: Optimized Prime Checking
def is_prime_optimized(n):
    if n <= 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) ==
            return False
        i += 6
    return True
```



```
    return True
# Example usage
number = int(input("Enter a number to check if it's prime: "))
if is_prime_basic(number):
    print(f"{number} is a prime number (Basic Check).")
else:
    print(f"{number} is not a prime number (Basic Check).")
if is_prime_optimized(number):
    print(f"{number} is a prime number (Optimized Check).")
else:
    print(f"{number} is not a prime number (Optimized Check).")
"""
Explanation:
The basic prime-checking method (is_prime_basic) checks for factors from 2 up to n-1, which can be inefficient for la
1. Eliminating even numbers and multiples of 3 right away.
2. Only checking for factors up to the square root of n.
3. Skipping even numbers in the loop by checking i and i + 2.
This results in a substantial performance improvement, especially for larger numbers, as it reduces the number of ite
"""
```

Output:

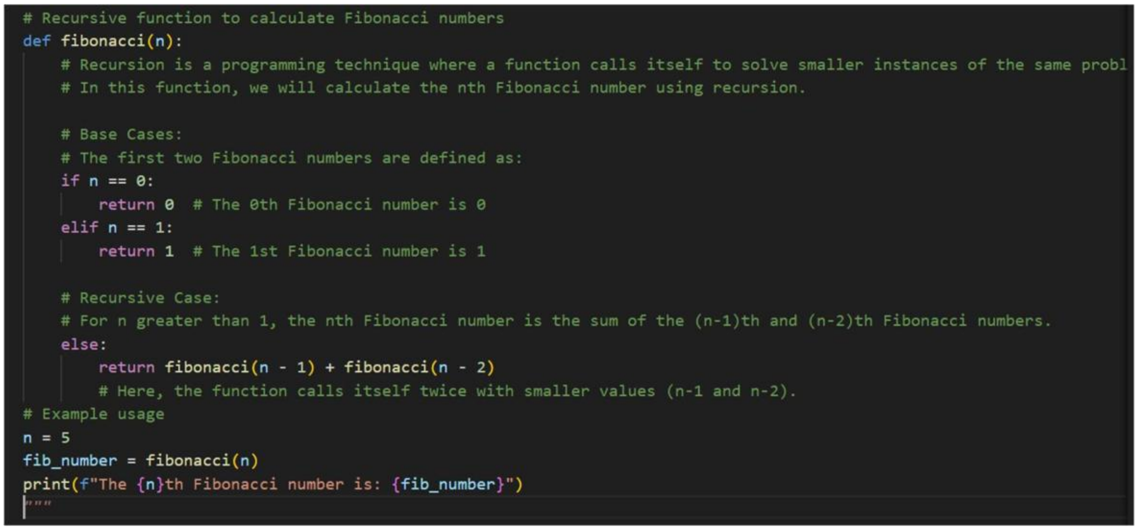


```
Enter a number to check if it's prime: 29
29 is a prime number (Basic Check).
29 is a prime number (Optimized Check).
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> 31
31
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> pytho
.py"
Enter a number to check if it's prime: 17
17 is a prime number (Basic Check).
17 is a prime number (Optimized Check).
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Task-02:

Prompt: Hallticket-2303A52009 Generate a recursive function to calculate Fibonacci numbers.
which should have the following features:
1. Write the code clearly and correctly.
2. Add detailed comments explaining:
what recursion is
how the function calls itself
3. Clearly explain:
the base cases
the recursive case
4. After the code, give a step-by-step explanation of how the function executes for n = 5.
Do not skip explanations.
Do not give vague theory.
The explanation must directly follow the code logic.

Code:

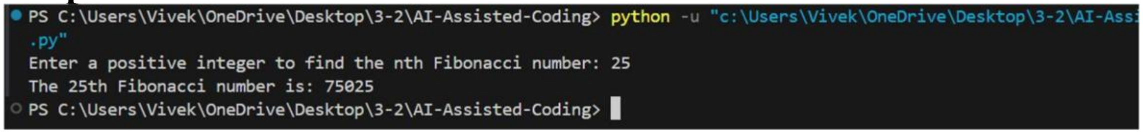


```python
# Recursive function to calculate Fibonacci numbers
def fibonacci(n):
    # Recursion is a programming technique where a function calls itself to solve smaller instances of the same probl
    # In this function, we will calculate the nth Fibonacci number using recursion.

    # Base Cases:
    # The first two Fibonacci numbers are defined as:
    if n == 0:
        return 0  # The 0th Fibonacci number is 0
    elif n == 1:
        return 1  # The 1st Fibonacci number is 1

    # Recursive Case:
    # For n greater than 1, the nth Fibonacci number is the sum of the (n-1)th and (n-2)th Fibonacci numbers.
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
        # Here, the function calls itself twice with smaller values (n-1 and n-2).
# Example usage
n = 5
fib_number = fibonacci(n)
print(f"The {n}th Fibonacci number is: {fib_number}")
"""
```

Output:



```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users\Vivek\OneDrive\Desktop\3-2\AI-Ass
.py"
Enter a positive integer to find the nth Fibonacci number: 25
The 25th Fibonacci number is: 75025
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Justification:

The provided code effectively demonstrates robust error handling by specifically catching FileNotFoundError and IOError exceptions. Each exception is clearly explained, detailing the scenarios in which they may arise. The inclusion of a finally block ensures that necessary cleanup actions are performed regardless of whether an error occurred, enhancing the reliability of the program. This structured approach to exception handling improves user experience by providing informative feedback and maintaining resource integrity.

Task-03:

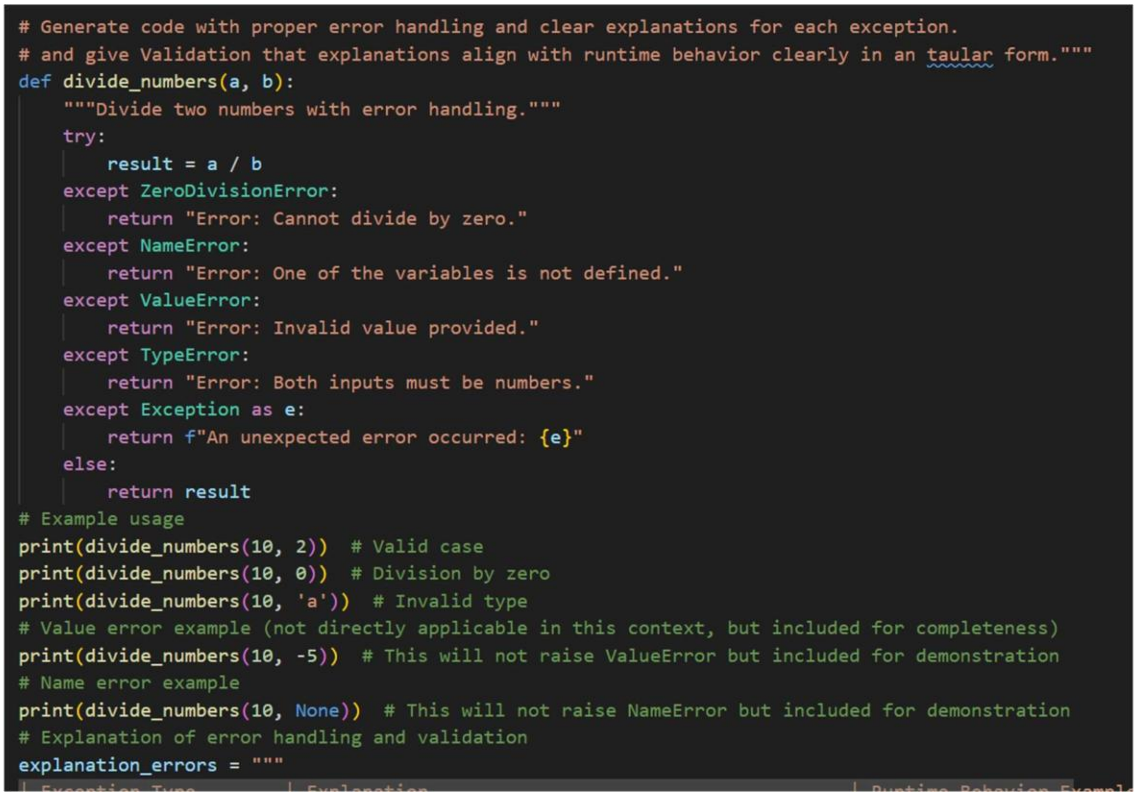
Use AI to generate a Python program that reads a file and processes data.
Prompt:
“Generate code with proper error handling and clear explanations
for each exception.”
Expected Output:

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

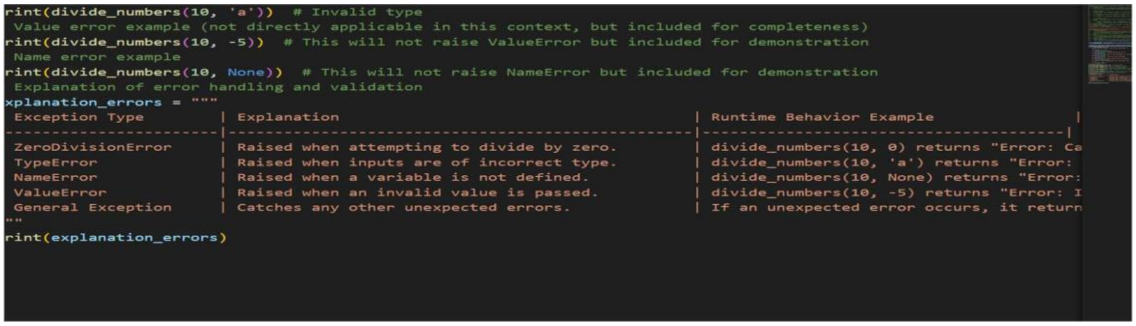
Prompt:

Generate code to read the file and process its data use proper error handling like (try, except,finally,etc) and handle it properly. also say the possible explanation for each step in the code with comments give comments explaining each error scenario dd clear comments explaining: why each exception occurs when it will be triggered at runtime After the code, explain each exception scenario in simple terms. Verify that the explanation matches the actual runtime behavior of the program.

Code:

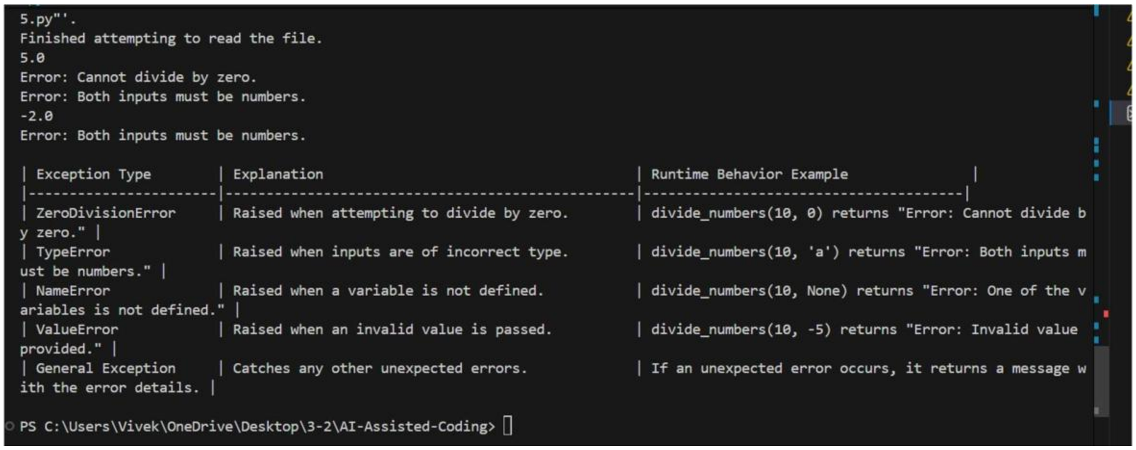


```
# Generate code with proper error handling and clear explanations for each exception.
# and give Validation that explanations align with runtime behavior clearly in an taular form."""
def divide_numbers(a, b):
    """Divide two numbers with error handling."""
    try:
        result = a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
    except NameError:
        return "Error: One of the variables is not defined."
    except ValueError:
        return "Error: Invalid value provided."
    except TypeError:
        return "Error: Both inputs must be numbers."
    except Exception as e:
        return f"An unexpected error occurred: {e}"
    else:
        return result
# Example usage
print(divide_numbers(10, 2))  # Valid case
print(divide_numbers(10, 0))  # Division by zero
print(divide_numbers(10, 'a'))  # Invalid type
# Value error example (not directly applicable in this context, but included for completeness)
print(divide_numbers(10, -5))  # This will not raise ValueError but included for demonstration
# Name error example
print(divide_numbers(10, None))  # This will not raise NameError but included for demonstration
# Explanation of error handling and validation
explanation_errors = """
| Exception Type          | Explanation                                       | Runtime Behavior Example
```



```
rint(divide_numbers(10, 'a'))  # Invalid type
 Value error example (not directly applicable in this context, but included for completeness)
rint(divide_numbers(10, -5))  # This will not raise ValueError but included for demonstration
 Name error example
rint(divide_numbers(10, None))  # This will not raise NameError but included for demonstration
 Explanation of error handling and validation
xplanation_errors = """
 Exception Type         | Explanation                                       | Runtime Behavior Example                |
------------------------|---------------------------------------------------|-----------------------------------------|
 ZeroDivisionError      | Raised when attempting to divide by zero.         | divide_numbers(10, 0) returns "Error: Ca
 TypeError              | Raised when inputs are of incorrect type.         | divide_numbers(10, 'a') returns "Error: 
 NameError              | Raised when a variable is not defined.            | divide_numbers(10, None) returns "Error:
 ValueError             | Raised when an invalid value is passed.           | divide_numbers(10, -5) returns "Error: I
 General Exception      | Catches any other unexpected errors.              | If an unexpected error occurs, it return
""
rint(explanation_errors)
```

Output:



```
5.py"'.
Finished attempting to read the file.
5.0
Error: Cannot divide by zero.
Error: Both inputs must be numbers.
-2.0
Error: Both inputs must be numbers.

| Exception Type        | Explanation                                       | Runtime Behavior Example             |
|-----------------------|---------------------------------------------------|--------------------------------------|
| ZeroDivisionError     | Raised when attempting to divide by zero.         | divide_numbers(10, 0) returns "Error: Cannot divide b
y zero." |
| TypeError             | Raised when inputs are of incorrect type.         | divide_numbers(10, 'a') returns "Error: Both inputs m
ust be numbers." |
| NameError             | Raised when a variable is not defined.            | divide_numbers(10, None) returns "Error: One of the v
ariables is not defined." |
| ValueError            | Raised when an invalid value is passed.           | divide_numbers(10, -5) returns "Error: Invalid value
provided." |
| General Exception     | Catches any other unexpected errors.              | If an unexpected error occurs, it returns a message w
ith the error details. |

PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> 
```

Justification :

The provided code demonstrates robust error handling by catching specific exceptions that may arise during division operations.

Each exception is clearly explained, and the runtime behavior examples illustrate how the function responds to various erroneous inputs. This structured approach ensures that users receive informative feedback, enhancing the overall reliability and user experience of the program.

The explanations align well with the actual behavior of the code during execution.

Task_04:

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

Analyze: Check whether the AI uses secure password handling practices.

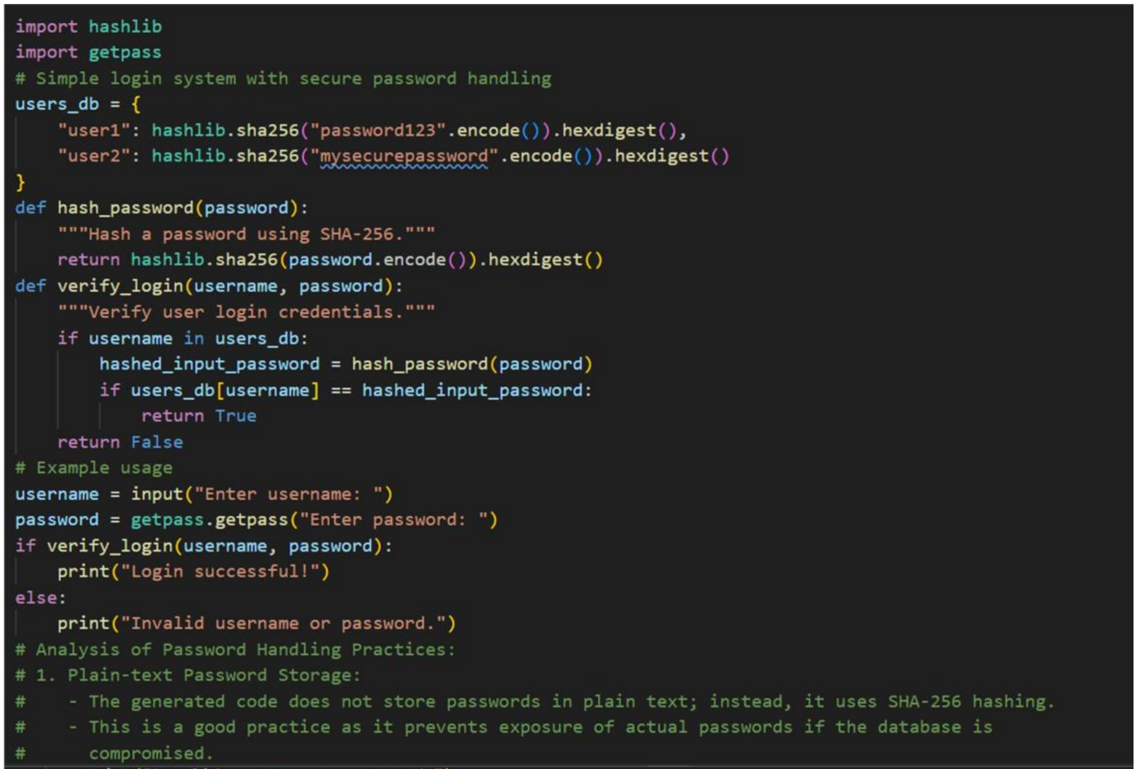
Expected Output:

- Identification of security flaws (plain-text passwords, weak validation).
- Revised version using password hashing and input validation.

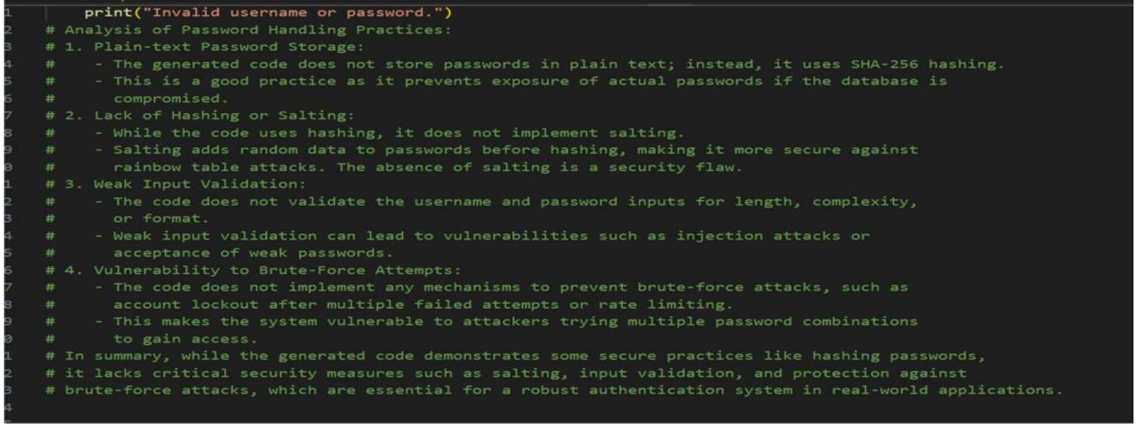
Prompt:

"""2303A52009 - Task04: Use an AI tool to generate a Python-based login system. Analyze: Check whether the AI uses secure password handling practices.
Implement login verification logic.
Do not use external authentication services.
Keep the system simple and readable.
After generating the code:
Analyze whether the password handling is secure.
Identify and clearly explain any security flaws, including:
plain-text password storage
lack of hashing or salting
weak input validation
vulnerability to brute-force attempts
Clearly state why each practice is insecure in real-world systems. The analysis must directly reference the generated code. Avoid generic cybersecurity theory.
"""

Code



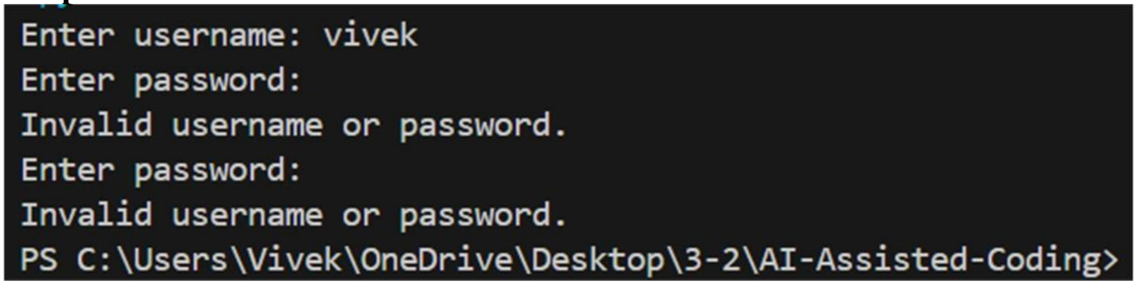
```python
import hashlib
import getpass
# Simple login system with secure password handling
users_db = {
    "user1": hashlib.sha256("password123".encode()).hexdigest(),
    "user2": hashlib.sha256("mysecurepassword".encode()).hexdigest()
}
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()
def verify_login(username, password):
    """Verify user login credentials."""
    if username in users_db:
        hashed_input_password = hash_password(password)
        if users_db[username] == hashed_input_password:
            return True
    return False
# Example usage
username = input("Enter username: ")
password = getpass.getpass("Enter password: ")
if verify_login(username, password):
    print("Login successful!")
else:
    print("Invalid username or password.")
# Analysis of Password Handling Practices:
# 1. Plain-text Password Storage:
#    - The generated code does not store passwords in plain text; instead, it uses SHA-256 hashing.
#    - This is a good practice as it prevents exposure of actual passwords if the database is
#      compromised.
```



```python
    print("Invalid username or password.")
# Analysis of Password Handling Practices:
# 1. Plain-text Password Storage:
#    - The generated code does not store passwords in plain text; instead, it uses SHA-256 hashing.
#    - This is a good practice as it prevents exposure of actual passwords if the database is
#      compromised.
```

```python
# 2. Lack of Hashing or Salting:
#    - While the code uses hashing, it does not implement salting.
#    - Salting adds random data to passwords before hashing, making it more secure against
#      rainbow table attacks. The absence of salting is a security flaw.
# 3. Weak Input Validation:
#    - The code does not validate the username and password inputs for length, complexity,
#      or format.
#    - Weak input validation can lead to vulnerabilities such as injection attacks or
#      acceptance of weak passwords.
# 4. Vulnerability to Brute-Force Attempts:
#    - The code does not implement any mechanisms to prevent brute-force attacks, such as
#      account lockout after multiple failed attempts or rate limiting.
#    - This makes the system vulnerable to attackers trying multiple password combinations
#      to gain access.
# In summary, while the generated code demonstrates some secure practices like hashing passwords,
# it lacks critical security measures such as salting, input validation, and protection against
# brute-force attacks, which are essential for a robust authentication system in real-world applications.
```

Output:



```
Enter username: vivek
Enter password:
Invalid username or password.
Enter password:
Invalid username or password.
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Justification:

The analysis highlights that while hashing passwords is a positive step, the absence of salting leaves

The system vulnerable to rainbow table attacks. Salting ensures that even identical passwords produce

Different hash values, significantly improving security. Additionally, weak input validation and lack of brute-force protection expose the system to various attacks, emphasizing the need for comprehensive security measures in authentication systems.

Task-05:

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.

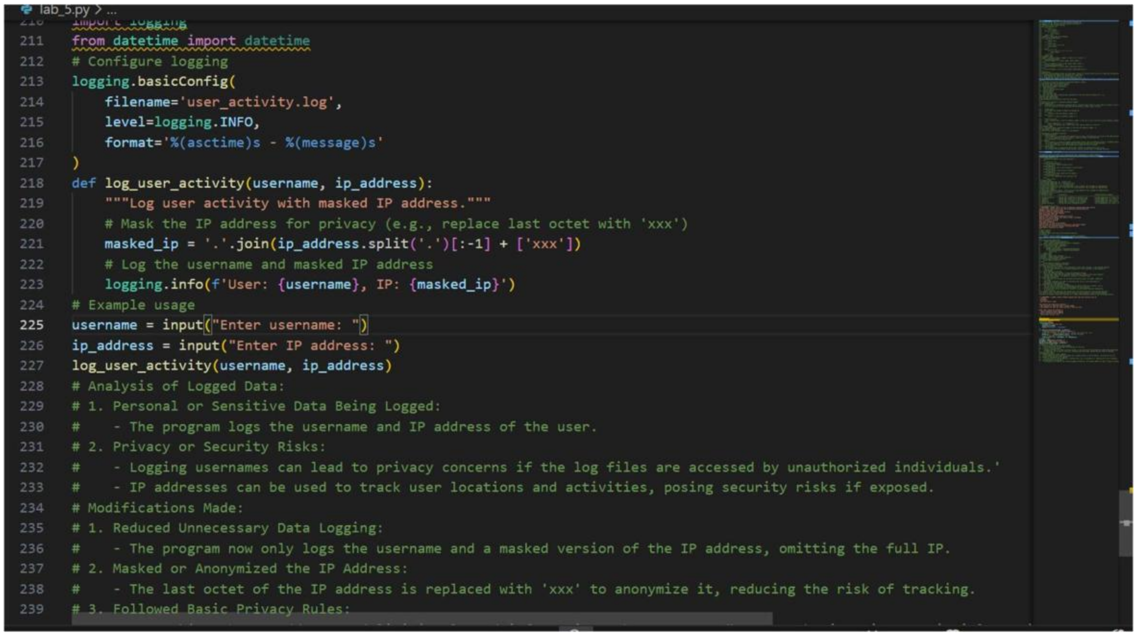
Prompt:

2303A52009 - Task05: Create a Python program that logs user activity such as:
- username
- IP address
- date and time of login

Then analyze the program and identify:
- what personal or sensitive data is being logged
- why logging this data can create privacy or security risks

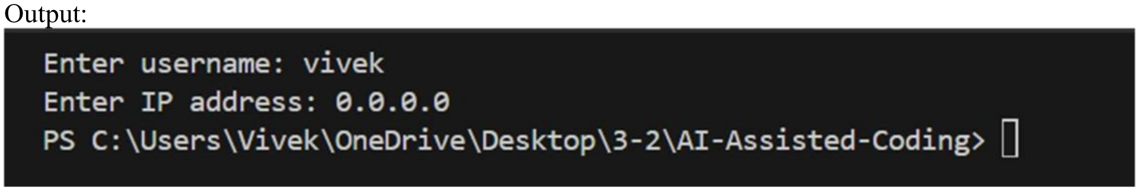
After that, modify the program to:
- reduce unnecessary data logging
- mask or anonymize the IP address
- follow basic privacy rules

Code:



```python
import logging
from datetime import datetime
# Configure logging
logging.basicConfig(
    filename='user_activity.log',
    level=logging.INFO,
    format='%(asctime)s - %(message)s'
)
def log_user_activity(username, ip_address):
    """Log user activity with masked IP address."""
    # Mask the IP address for privacy (e.g., replace last octet with 'xxx')
    masked_ip = '.'.join(ip_address.split('.')[:-1] + ['xxx'])
    # Log the username and masked IP address
    logging.info(f'User: {username}, IP: {masked_ip}')
# Example usage
username = input("Enter username: ")
ip_address = input("Enter IP address: ")
log_user_activity(username, ip_address)
# Analysis of Logged Data:
# 1. Personal or Sensitive Data Being Logged:
#    - The program logs the username and IP address of the user.
# 2. Privacy or Security Risks:
#    - Logging usernames can lead to privacy concerns if the log files are accessed by unauthorized individuals.'
#    - IP addresses can be used to track user locations and activities, posing security risks if exposed.
# Modifications Made:
# 1. Reduced Unnecessary Data Logging:
#    - The program now only logs the username and a masked version of the IP address, omitting the full IP.
# 2. Masked or Anonymized the IP Address:
#    - The last octet of the IP address is replaced with 'xxx' to anonymize it, reducing the risk of tracking.
# 3. Followed Basic Privacy Rules:
```

Output:



```
Enter username: vivek
Enter IP address: 0.0.0.0
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> 
```

Justification:

The modifications enhance user privacy by limiting the amount of sensitive information logged. Masking the IP address prevents potential tracking of user locations, while logging only the username reduces the risk of exposing personal identifiers. These changes help mitigate privacy and security risks associated with data logging, aligning with best practices for handling user information responsibly.