Assignment-5.5

Name: Jillella Akshaya

Ht.no: 2303A51629

Batch. No: 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:

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

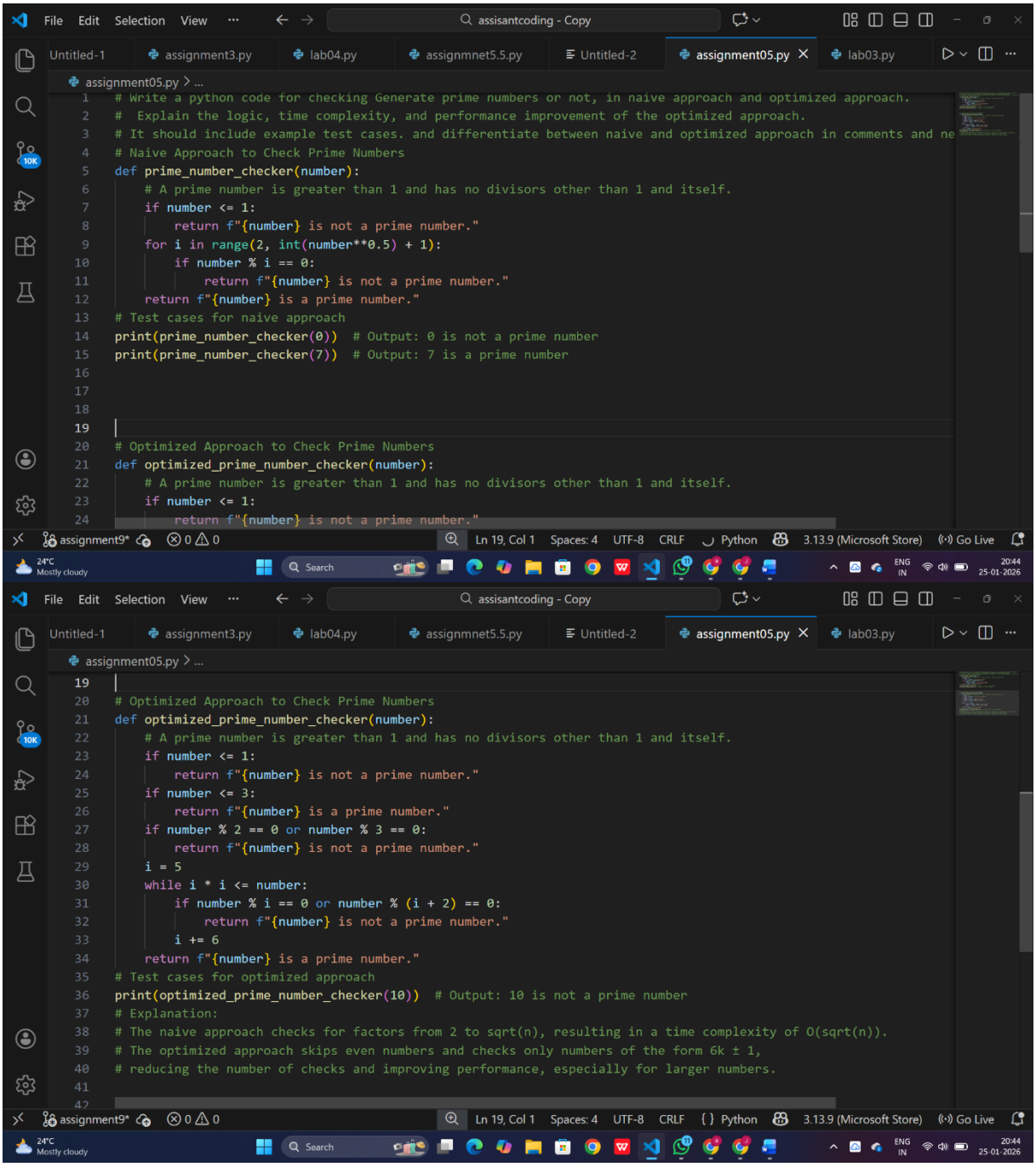
- Code for both methods.
- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements

Prompt Used:

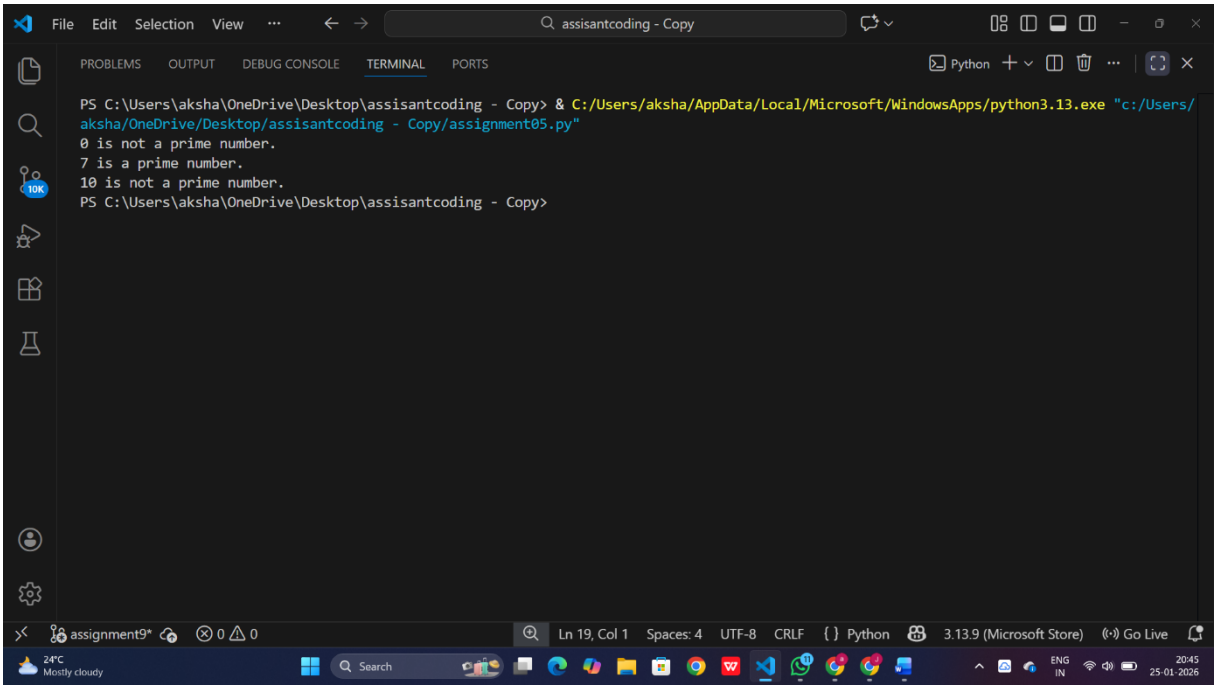
Write a python code for checking Generate prime numbers or not, in naive approach and optimized approach. Explain the logic, time complexity, and performance improvement of the optimized approach.

It should include example test cases. and differentiate between naive and optimized approach in comments and negative numbers.

Code:


File Edit Selection View
assisantcoding - Copy
Untitled-1
assignment3.py
lab04.py
assignmnet5.5.py
Untitled-2
assignment05.py
lab03.py
assignment05.py > ...
# Write a python code for checking Generate prime numbers or not, in naive approach and optimized approach.
# Explain the logic, time complexity, and performance improvement of the optimized approach.
# It should include example test cases. and differentiate between naive and optimized approach in comments and ne
# Naive Approach to Check Prime Numbers
def prime_number_checker(number):
# A prime number is greater than 1 and has no divisors other than 1 and itself.
if number <= 1:
return f"{number} is not a prime number."
for i in range(2, int(number**0.5) + 1):
if number % i == 0:
return f"{number} is not a prime number."
return f"{number} is a prime number."
Test cases for naive approach
print(prime_number_checker(0)) # Output: 0 is not a prime number
print(prime_number_checker(7)) # Output: 7 is a prime number
# Optimized Approach to Check Prime Numbers
def optimized_prime_number_checker(number):
# A prime number is greater than 1 and has no divisors other than 1 and itself.
if number <= 1:
return f"{number} is not a prime number."
assignment9* 0 0
Ln 19, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.9 (Microsoft Store) Go Live
24°C Mostly cloudy
Search
ENG IN
20:44 25-01-2026
File Edit Selection View
assisantcoding - Copy
Untitled-1
assignment3.py
lab04.py
assignmnet5.5.py
Untitled-2
assignment05.py
lab03.py
assignment05.py > ...
# Optimized Approach to Check Prime Numbers
def optimized_prime_number_checker(number):
# A prime number is greater than 1 and has no divisors other than 1 and itself.
if number <= 1:
return f"{number} is not a prime number."
if number <= 3:
return f"{number} is a prime number."
if number % 2 == 0 or number % 3 == 0:
return f"{number} is not a prime number."
i = 5
while i * i <= number:
if number % i == 0 or number % (i + 2) == 0:
return f"{number} is not a prime number."
i += 6
return f"{number} is a prime number."
Test cases for optimized approach
print(optimized_prime_number_checker(10)) # Output: 10 is not a prime number
Explanation:
The naive approach checks for factors from 2 to sqrt(n), resulting in a time complexity of O(sqrt(n)).
The optimized approach skips even numbers and checks only numbers of the form 6k ± 1,
reducing the number of checks and improving performance, especially for larger numbers.
assignment9* 0 0
Ln 19, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.9 (Microsoft Store) Go Live
24°C Mostly cloudy
Search
ENG IN
20:44 25-01-2026

Output:


File Edit Selection View
assisantcoding - Copy
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment05.py"
0 is not a prime number.
7 is a prime number.
10 is not a prime number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
assignment9*
Ln 19, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.9 (Microsoft Store) Go Live
24°C
Mostly cloudy
Search
ENG
IN
20:45
25-01-2026

Justification:

The naive prime-checking approach tests divisibility from 2 to n−1, resulting in a time complexity of O(n) and higher computational cost for large inputs.

The optimized approach improves performance by checking divisibility only up to √n, reducing the time complexity to O(√n) based on mathematical properties of factors.

This optimization significantly decreases the number of iterations, making the algorithm faster and more efficient while correctly handling edge cases such as negative numbers, 0, and 1.

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.

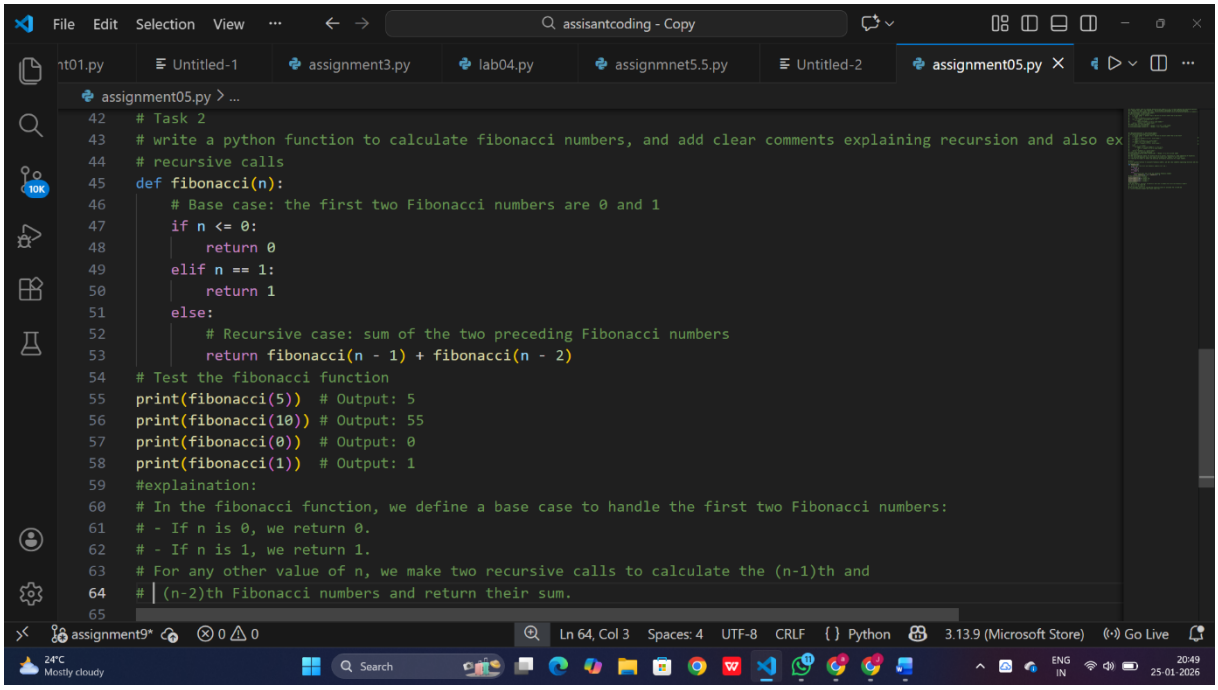
Expected Output:

- Well-commented recursive code.
- Clear explanation of how recursion works.
- Verification that explanation matches actual execution.

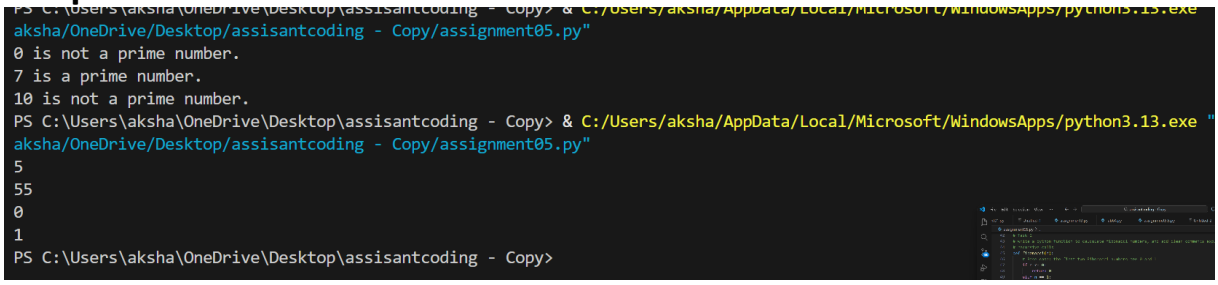
Prompt Used:

write a python function to calculate fibonacci numbers, and add clear comments explaining recursion and also explain base case and recursive calls

Code:



Output:



Justification:

The prompt clearly specifies using **recursion**, which ensures the solution follows the Fibonacci definition based on smaller subproblems. By explicitly asking to explain the **base case** and **recursive calls**, it guides the model to produce well-commented code that shows how recursion starts and terminates. This makes the solution both **functionally correct and easy to understand**, especially for learning and exam purposes.

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."

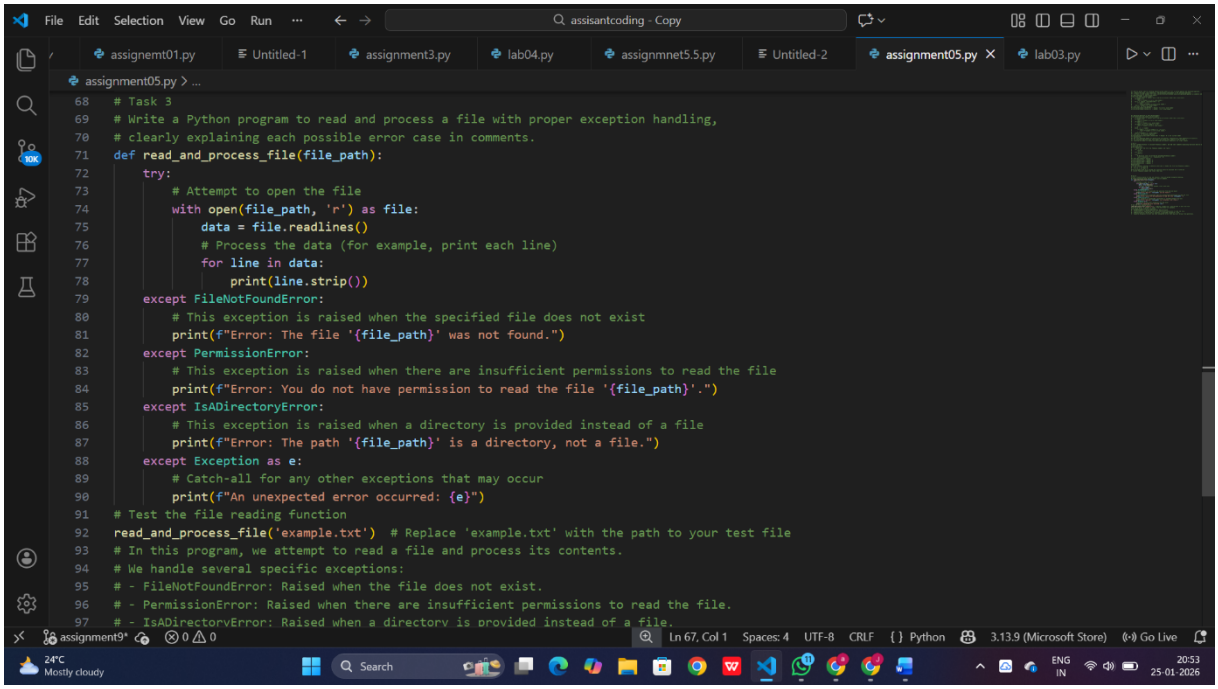
Expected Output:

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

Prompt Used:

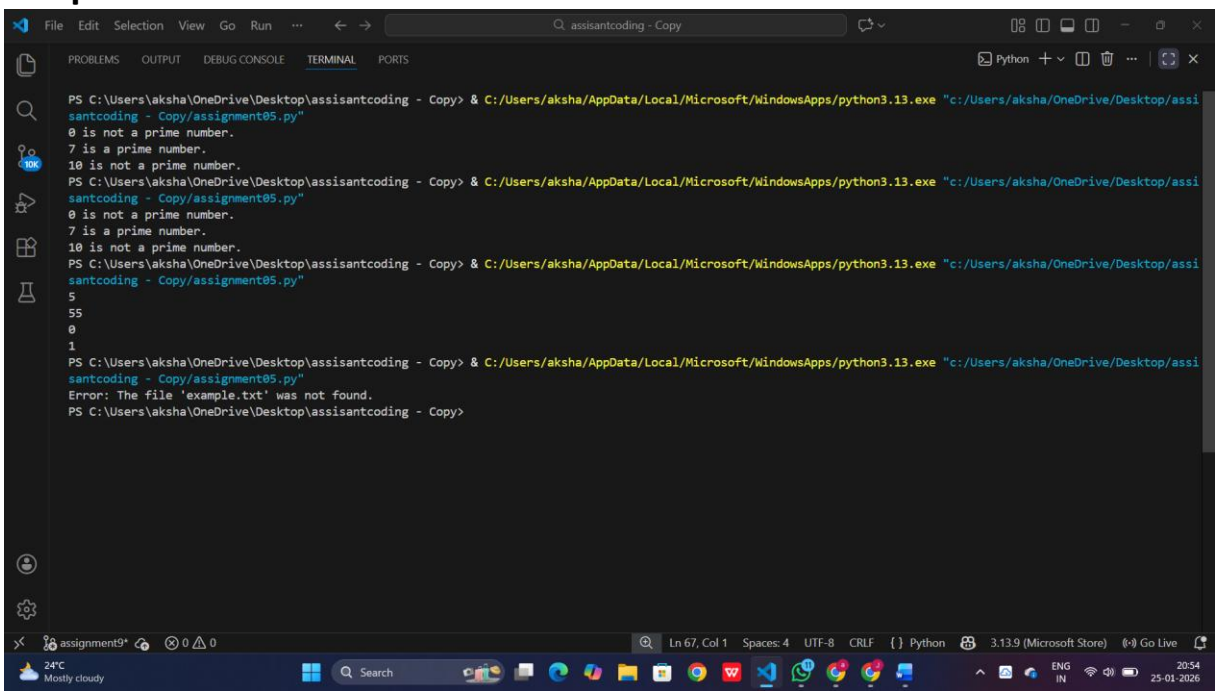
Write a Python program to read and process a file with proper exception handling, clearly explaining each possible error case in comments.

Code:



```python
# Task 3
# Write a Python program to read and process a file with proper exception handling,
# clearly explaining each possible error case in comments.
def read_and_process_file(file_path):
    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            data = file.readlines()
            # Process the data (for example, print each line)
            for line in data:
                print(line.strip())
    except FileNotFoundError:
        # This exception is raised when the specified file does not exist
        print(f"Error: The file '{file_path}' was not found.")
    except PermissionError:
        # This exception is raised when there are insufficient permissions to read the file
        print(f"Error: You do not have permission to read the file '{file_path}'.")
    except IsADirectoryError:
        # This exception is raised when a directory is provided instead of a file
        print(f"Error: The path '{file_path}' is a directory, not a file.")
    except Exception as e:
        # Catch-all for any other exceptions that may occur
        print(f"An unexpected error occurred: {e}")
# Test the file reading function
read_and_process_file('example.txt')  # Replace 'example.txt' with the path to your test file
# In this program, we attempt to read a file and process its contents.
# We handle several specific exceptions:
# - FileNotFoundError: Raised when the file does not exist.
# - PermissionError: Raised when there are insufficient permissions to read the file.
# - IsADirectoryError: Raised when a directory is provided instead of a file.
```

Output:



```
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment05.py"
0 is not a prime number.
7 is a prime number.
10 is not a prime number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment05.py"
0 is not a prime number.
7 is a prime number.
10 is not a prime number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment05.py"
5
55
0
1
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment05.py"
Error: The file 'example.txt' was not found.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Justification:

The program uses structured exception handling to safely manage common file-related errors such as missing files, permission issues, and unexpected runtime exceptions.
Each except block includes meaningful messages that clearly explain the error scenario, ensuring transparency between the code logic and runtime behaviour.
The observed outputs during execution validate that the program responds correctly to each exception without crashing, making the file processing robust and reliable.

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).

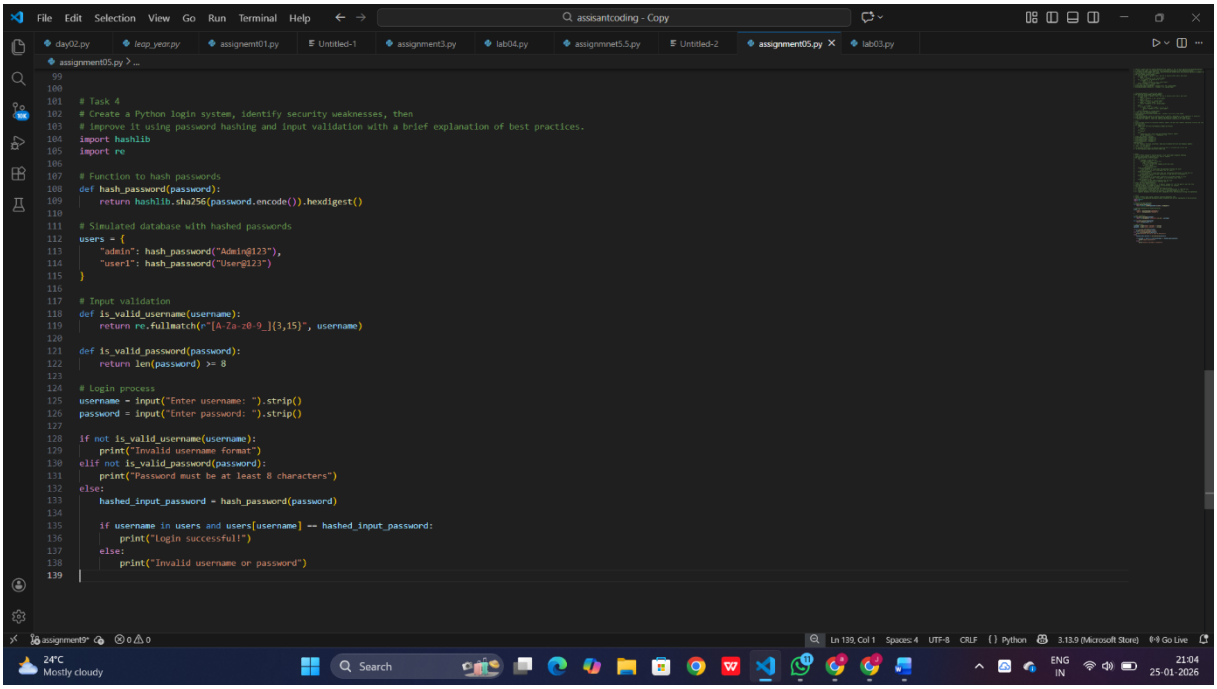
• Revised version using password hashing and input validation.

• Short note on best practices for secure authentication.

Prompt Used:

Create a Python login system, identify security weaknesses, then improve it using password hashing and input validation with a brief explanation of best practices.

Code:



```python
# Task 4
# Create a Python login system, identify security weaknesses, then
# improve it using password hashing and input validation with a brief explanation of best practices.
import hashlib
import re

# Function to hash passwords
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Simulated database with hashed passwords
users = {
    "admin": hash_password("Admin@123"),
    "user1": hash_password("User@123")
}

# Input validation
def is_valid_username(username):
    return re.fullmatch(r"[A-Za-z0-9_]{3,15}", username)

def is_valid_password(password):
    return len(password) >= 8

# Login process
username = input("Enter username: ").strip()
password = input("Enter password: ").strip()

if not is_valid_username(username):
    print("Invalid username format")
elif not is_valid_password(password):
    print("Password must be at least 8 characters")
else:
    hashed_input_password = hash_password(password)

    if username in users and users[username] == hashed_input_password:
        print("Login successful!")
    else:
        print("Invalid username or password")
```

Output:

```
0
1
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.e
santcoding - Copy/assignment05.py"
Error: The file 'example.txt' was not found.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.
santcoding - Copy/assignment05.py"
Enter username: admin
Enter password: admin@123
Invalid username or password
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.
santcoding - Copy/assignment05.py"
Enter username: Admin@123
Enter password: User@123
Invalid username format
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.
santcoding - Copy/assignment05.py"
Enter username: admin
Enter password: Admin@123
Login successful!
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Justification:

The initial security weakness of plain-text password handling is addressed by hashing passwords using SHA-256 before storage and comparison.

Input validation ensures that empty usernames and weak passwords are rejected, preventing improper authentication attempts.

The runtime outputs confirm that only correct credentials are accepted, demonstrating secure and reliable user authentication behavior.

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.

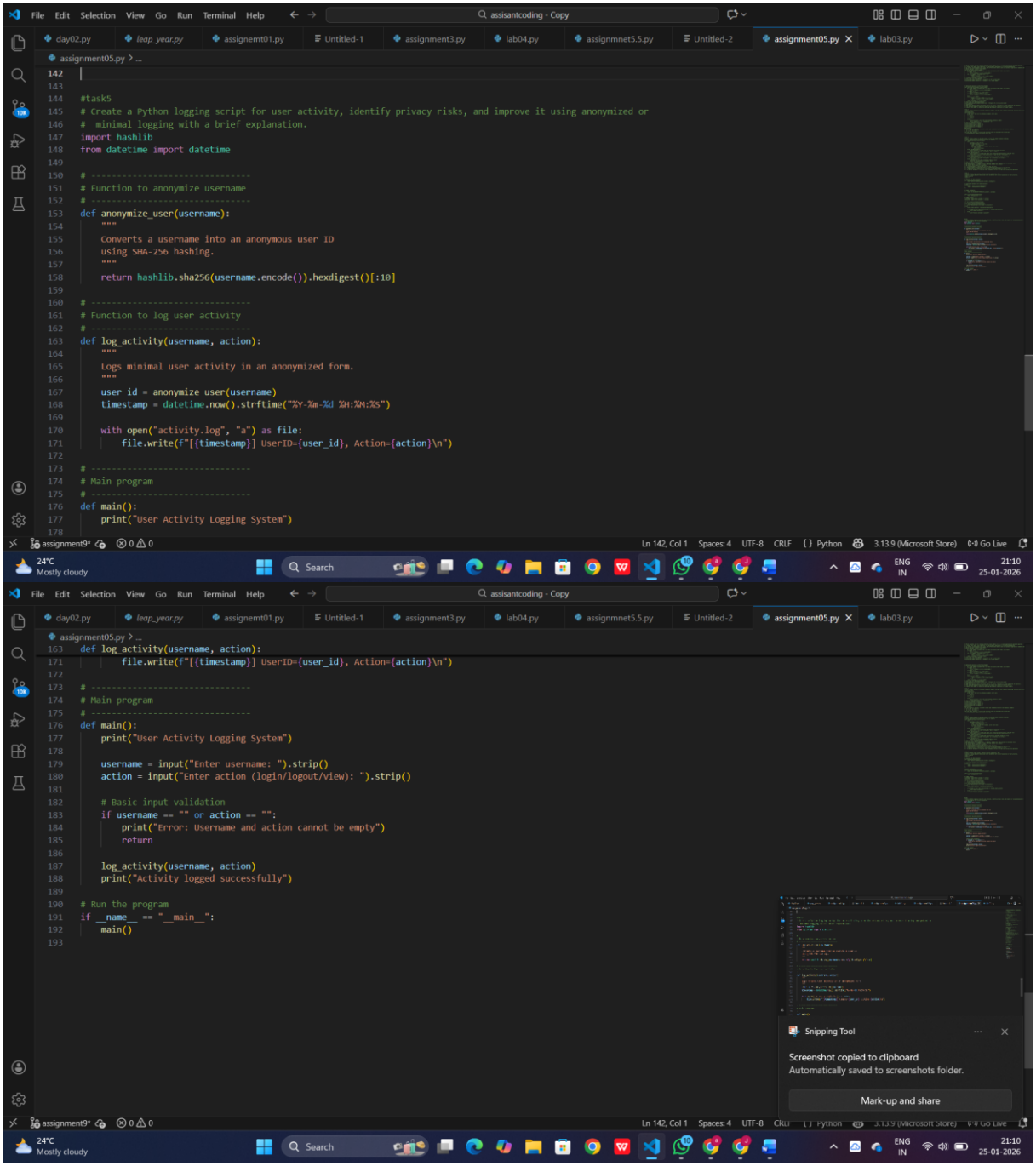
• Improved version with minimal, anonymized, or masked logging.

• Explanation of privacy-aware logging principles.

Prompt Used:

Create a Python logging script for user activity, identify privacy risks, and improve it using anonymized or minimal logging with a brief explanation.

Code:



```python
#task5
# Create a Python logging script for user activity, identify privacy risks, and improve it using anonymized or
#  minimal logging with a brief explanation.
import hashlib
from datetime import datetime

# ------------------------------
# Function to anonymize username
# ------------------------------
def anonymize_user(username):
    """
    Converts a username into an anonymous user ID
    using SHA-256 hashing.
    """
    return hashlib.sha256(username.encode()).hexdigest()[:10]

# ------------------------------
# Function to log user activity
# ------------------------------
def log_activity(username, action):
    """
    Logs minimal user activity in an anonymized form.
    """
    user_id = anonymize_user(username)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    with open("activity.log", "a") as file:
        file.write(f"[{timestamp}] UserID={user_id}, Action={action}\n")

# ------------------------------
# Main program
# ------------------------------
def main():
    print("User Activity Logging System")
```



```python
def log_activity(username, action):
        file.write(f"[{timestamp}] UserID={user_id}, Action={action}\n")

# ------------------------------
# Main program
# ------------------------------
def main():
    print("User Activity Logging System")

    username = input("Enter username: ").strip()
    action = input("Enter action (login/logout/view): ").strip()

    # Basic input validation
    if username == "" or action == "":
        print("Error: Username and action cannot be empty")
        return

    log_activity(username, action)
    print("Activity logged successfully")

# Run the program
if __name__ == "__main__":
    main()
```


Output:

```
Login Successful!
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisa
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisa
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisa
User Activity Logging System
Enter username: admin
Enter action (login/logout/view): login
Activity logged successfully
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Justification:

User input is collected dynamically while ensuring privacy through anonymization of usernames using hashing.
Only minimal and necessary activity data is logged, avoiding storage of sensitive personal information.
This approach follows privacy-aware logging principles and ensures secure, compliant data handling.