ASSIGNMENT – 5.5

HT.NO: 2303A52283

BATCH: 45

TASK-01:

Question:

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.(print output)

Generated Code:

```
# Basic prime-checking method
def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
# Optimized prime-checking method
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
# Example usage
number = 29
print(f"Basic method: Is {number} prime? {is_prime_basic(number)}")
print(f"Optimized method: Is {number} prime? {is_prime_optimized(number)}")
# Explanation of performance improvement
# The optimized version improves performance by reducing the number of checks needed to determine if a number is prime.
# It eliminates even numbers and multiples of 3 right away, and only checks for factors up
# to the square root of n, which significantly decreases the number of iterations for larger numbers.
```

Explanation:

This code checks whether a number is prime using two methods.
The basic method tries dividing the number by all smaller numbers, which is slow.
The optimized method skips unnecessary checks and only tests divisors up to √n, making it much faster.

TASK-02:

Question:

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.

Prompt:

generate a python code to calculate fibonacci numbers using recursion explain the code clearly using comments explain base classes and recursive calls in the code(identify it)

Generated Code:

```
def fibonacci(n):
    # Base case: if n is 0 or 1, return n
    if n <= 1:
        return n
    # Recursive case: return the sum of the two preceding Fibonacci numbers
    return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage
num = 10
print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
# Explanation:
# In this code, we define a function 'fibonacci' that calculates the nth Fibonacci number using recursion.
# The base cases are when n is 0 or 1, where we simply return n
# because the Fibonacci sequence starts with 0 and 1.
# For any other value of n, the function calls itself twice:
# once with (n-1) and once with (n-2), and adds the results of these two calls together.
# These recursive calls continue until they reach the base cases.
# This method effectively breaks down the problem into smaller subproblems,
# which is a key characteristic of recursive algorithms.
```

Explanation:

This function finds the nth Fibonacci number using recursion.
For n = 0 or 1, it directly returns n as the base case.
For larger n, it adds the results of the two previous Fibonacci numbers by calling itself.

TASK-03:

Question:

Use AI to generate a Python program that reads a file and processes data.

Prompt:

Generate a Python program that reads data from a file and processes it.
Include proper error handling using try–except blocks.
Handle errors such as file not found, invalid data, and unexpected exceptions.
Add clear comments explaining each error and how the program responds to it.

Generated Code:

```
def process_file(file_path):
    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            data = file.readlines()

        processed_data = []
        for line in data:
            try:
                # Attempt to convert each line to an integer
                number = int(line.strip())
                processed_data.append(number * 2)  # Example processing: doubling the number
            except ValueError:
                # Handle invalid data that cannot be converted to an integer
                print(f"Invalid data encountered: '{line.strip()}'. Skipping this line.")

        return processed_data

    except FileNotFoundError:
        # Handle the case where the file does not exist
        print(f"Error: The file '{file_path}' was not found.")
    except Exception as e:
        # Handle any other unexpected exceptions
```

```
        print(f"An unexpected error occurred: {e}")

# Example usage

file_path = 'data.txt'

result = process_file(file_path)

if result is not None:

    print("Processed data:", result)
```

Explanation:

This code reads numbers from a file and doubles each valid number.
If a line is not a number, it skips it and shows a warning.
It also safely handles errors like missing files or unexpected issues.

TASK-04:

Question:

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

Analyze: Check whether the AI uses secure password handling

practices.

Prompt:

Generate a Python login system. Identify security issues in it. Rewrite the code using password hashing and input validation.Add a short note on secure authentication best practices.

Generated Code:

```
import hashlib

# Insecure login system

users_db = {

    'user1': 'password123',  # Plain-text password storage

    'user2': 'mypassword'

}

def login(username, password):

    if username in users_db and users_db[username] == password:

        return "Login successful!"
```

```
    else:
        return "Invalid username or password."
# Secure login system with password hashing
secure_users_db = {
    'user1': hashlib.sha256('password123'.encode()).hexdigest(),
    'user2': hashlib.sha256('mypassword'.encode()).hexdigest()
}
def secure_login(username, password):
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    if username in secure_users_db and secure_users_db[username] == hashed_password:
        return "Login successful!"
    else:
        return "Invalid username or password."
# Best practices for secure user authentication:
# 1. Always store passwords using strong hashing algorithms (e.g., bcrypt, Argon
# 2. Implement input validation to prevent injection attacks.
# 3. Use secure authentication protocols (e.g., OAuth, JWT).
# Example usage
print(login('user1', 'password123'))  # Insecure login
print(secure_login('user1', 'password123'))  # Secure login
```

Explanation:

This code compares an insecure login system with a secure one.
The insecure version stores and checks passwords in plain text, which is unsafe.
The secure version hashes passwords before storing and comparing them, making it much safer.

TASK-05:

Question:

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.

Prompt:

Generate a Python script that logs user activity.Identify privacy risks in the logging.Improve the script using masked or anonymized data.Explain privacy-aware logging briefly.

Generated Code:

```
import logging

import hashlib

# Configure logging

logging.basicConfig(filename='user_activity.log', level=logging.INFO,
format='%(asctime)s  - %(message)s')

# Function to anonymize user data using hashing

def anonymize_user_data(user_id):

    return hashlib.sha256(user_id.encode()).hexdigest()

# Log user activity with anonymized data

def log_user_activity(user_id, activity):

    anonymized_id = anonymize_user_data(user_id)

    logging.info(f'User: {anonymized_id} - Activity: {activity}')

# Example usage

log_user_activity('user123', 'Logged in')

# Explanation of privacy-aware logging:

# Privacy-aware logging involves minimizing the amount of personally identifiable
information (PII) stored in logs.

# By anonymizing or masking user data, we reduce the risk of exposing sensitive information
in case

# of a data breach. This approach helps protect user privacy while still allowing for effective
monitoring and analysis of user activity.
```

Explanation:

This code logs user activity while protecting privacy.
It hashes the user ID so real identities are not stored in the log file.
This helps track actions safely without exposing personal information.

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SPELL CHECKER 2                Python  +

● priyamvadasingh.b@Priyamvadas-MacBook-Pro AAC % /usr/bin/python3 /Users/priyamvadasingh.b/Desktop/AAC/Lab-5.py
  Basic method: Is 29 prime? True
  Optimized method: Is 29 prime? True
● priyamvadasingh.b@Priyamvadas-MacBook-Pro AAC % /usr/bin/python3 /Users/priyamvadasingh.b/Desktop/AAC/Lab-5.py
  The 10th Fibonacci number is: 55
● priyamvadasingh.b@Priyamvadas-MacBook-Pro AAC % /usr/bin/python3 /Users/priyamvadasingh.b/Desktop/AAC/Lab-5.py
  Error: The file 'data.txt' was not found.
● priyamvadasingh.b@Priyamvadas-MacBook-Pro AAC % /usr/bin/python3 /Users/priyamvadasingh.b/Desktop/AAC/Lab-5.py
  Login successful!
  Login successful!
● priyamvadasingh.b@Priyamvadas-MacBook-Pro AAC % /usr/bin/python3 /Users/priyamvadasingh.b/Desktop/AAC/Lab-5.py
```