AI ASSISTANT CODING

ASSIGNMENT – 5.5

NAME : K.Lokesh

HT.NO : 2303A51201

BATCH : 21

Lab 5: Ethical Foundations – Responsible AI Coding Practices

Lab Objectives:

• To explore the ethical risks associated with AI-generated code.

• To recognize issues related to security, bias, transparency, and copyright.

• To reflect on the responsibilities of developers when using AI tools in software development.

• To promote awareness of best practices for responsible and ethical AI coding.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

• Identify and avoid insecure coding patterns generated by AI tools.

• Detect and analyze potential bias or discriminatory logic in AI-generated outputs.

• Evaluate originality and licensing concerns in reused AI-generated code.

• Understand the importance of explainability and transparency in AI-assisted programming.

• Reflect on accountability and the human role in ethical AI

coding practices.

Task1: 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.

METHOD 1 :

```
"""
generate a code to check if a number is prime or not(naive approach)
"""
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Example usage
number = 29
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

OUTPUT:

```
RD Year\AIAC\5.5 task1.py'
29 is a prime number.
```

METHOD 2 :

```
""" optimized approach to check if a number is prime or not
    input1: 100
    output1: 100 is not a prime number.
    input2: 37
    output2: 37 is a prime number.
    input3: -1
    output3: give a valid positive integer
    input4: a%
    output4: give a valid positive integer
    input5 : akjh
    output5: give a valid positive integer
    input6 : #%
    output6: give a valid positive integer
    note: handle invalid inputs, and negative numbers, and zero;
    and make the user retry until a valid positive integer is given.

"""
```

```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
while True:
    user_input = input("Enter a positive integer: ")
    try:
        number = int(user_input)
        if number <= 0:
            print("Give a valid positive integer.")
            continue
        if is_prime_optimized(number):
            print(f"{number} is a prime number.")
        else:
            print(f"{number} is not a prime number.")
        break
    except ValueError:
        print("Give a valid positive integer.")
        continue
```

OUTPUT :

```
Give a valid positive integer.
Enter a positive integer: we
Give a valid positive integer.
Enter a positive integer: 234
234 is not a prime number.
```

FINAL DESCRIPTION :

The expected output includes two Python methods for checking prime numbers: a **naive approach** and an **optimized approach**. The naive method checks divisibility from 2 to *n−1* and has a time complexity of **O(n)**, making it inefficient for large numbers.
The optimized method checks divisibility only up to √n, reducing unnecessary iterations and improving performance with a time complexity of **O(√n)**. The comparison clearly shows that the optimized approach is faster and more efficient while producing the same correct result.

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 :

```python
""" 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."""

def fibonacci(n):
    # Base case: if n is 0, return 0
    if n == 0:
        return 0
    # Base case: if n is 1, return 1
    elif n == 1:
        return 1
    else:
        # Recursive case: return the sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage:
num = 10
print(f"Fibonacci number at position {num} is {fibonacci(num)}")
```

```python
"""
# This function calculates the nth Fibonacci number using recursion.
# The base cases are defined for n = 0 and n = 1, where the Fibonacci numbers are 0 and 1 respectively.
# For any other value of n, the function calls itself twice to calculate the two preceding Fibonacci numbers (n-1 and n-2)
# and returns their sum. This process continues until the base cases are reached.
# For example, to calculate fibonacci(4):
# fibonacci(4) calls fibonacci(3) and fibonacci(2)
# fibonacci(3) calls fibonacci(2) and fibonacci(1)
# fibonacci(2) calls fibonacci(1) and fibonacci(0)
# The calls eventually reach the base cases, and the results are combined to get the final answer.
"""
```

OUTPUT :

```
\launcher' '24832' '--' 'C:\Users\Dhanush Reddy\OneDrive\3RD Year\AIAC\5.5 task2.py
Fibonacci number at position 10 is 55
```

FINAL DESCRIPTION :

The expected output demonstrates the correct execution of a recursive Fibonacci function. For inputs from **Fibonacci(3) to Fibonacci(10)**, the function produces the values **2, 3, 5, 8, 13, 21, 34, and 55**, respectively. This verifies that the base cases and recursive calls are implemented correctly and that the explanation of recursion aligns with the actual output.

Task3: 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 :

```
"""Generate code with proper error handling and clear explanations
for each exception.
clear comments should be added to explain the purpose of each exception handling block.
validation that explanations align with runtime behavior.
"""
```

```
def divide_numbers(num1, num2):
    try:
        # Attempt to perform division
        result = num1 / num2
    except ZeroDivisionError:
        # Handle division by zero error
        print("Error: Cannot divide by zero. Please provide a non-zero denominator.")
        return None
    except TypeError:
        # Handle type error if inputs are not numbers
        print("Error: Invalid input type. Please provide numeric values for both numerator and denominator.")
        return None
    else:
        # If no exceptions occur, return the result
        return result
# Example usage
numerator = int(input("Enter the numerator: "))
denominator = int(input("Enter the denominator: "))
division_result = divide_numbers(numerator, denominator)
if division_result is not None:
    print(f"The result of dividing {numerator} by {denominator} is {division_result}")
```

```
"""
# This function divides two numbers with error handling for common exceptions.
# It handles ZeroDivisionError to prevent division by zero and TypeError to ensure inputs are numeric
# If an exception occurs, a relevant error message is printed and None is returned.
# If the division is successful, the result is returned.
"""
```

OUTPUT :

```
Enter the numerator: 3
Enter the denominator: sa
Error: Invalid input type. Please provide numeric values for both numerator and denominator.
```

```
Enter the numerator: 3
Enter the denominator: 0
Error: Invalid input type. Please provide numeric values for both numerator and denominator.
```

FINAL DESCRIPTION :

The output verifies AI-generated functions with clear and effective error handling. Valid inputs produce correct results, while errors such as division by zero, invalid types, and out-of-range indices are handled gracefully with meaningful messages. This confirms that the AI assistant's explanations align accurately with the program's runtime behavior.

Task 4: 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 :

```
"""
generate a Python-based login system.
"""

import getpass
class LoginSystem:
    def __init__(self):
        self.users = {}

    def register(self, username, password):
        if username in self.users:
            print("Username already exists.")
            return False
        self.users[username] = password
        print("User registered successfully.")
        return True

    def login(self, username, password):
        if username not in self.users:
            print("Username does not exist.")
            return False
        if self.users[username] != password:
            print("Incorrect password.")
            return False
        print("Login successful.")
        return True
```

```
"""
A simple command-line login system allowing user registration and login.
"""
"""Analyze: Check whether the AI uses secure password handling practices.
• 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.

"""

# Note on Best Practices for Secure Authentication:
# 1. Always store passwords securely using hashing algorithms (e.g., bcrypt, Argon
# 2. Implement input validation to prevent injection attacks.
# 3. Use secure methods for password input (e.g., getpass in Python).
```

```python
def main():
    system = LoginSystem()
    while True:
        action = input("Do you want to (r)egister, (l)ogin, or (q)uit? ").lower()
        if action == 'r':
            username = input("Enter a username: ")
            password = getpass.getpass("Enter a password: ")
            system.register(username, password)
        elif action == 'l':
            username = input("Enter your username: ")
            password = getpass.getpass("Enter your password: ")
            system.login(username, password)
        elif action == 'q':
            print("Exiting the system.")
            break
        else:
            print("Invalid option. Please try again.")
if __name__ == "__main__":
    main()
```

OUTPUT :

```
Do you want to (r)egister, (l)ogin, or (q)uit? r
Enter a username: Dhanush
Enter a password:
User registered successfully.
Do you want to (r)egister, (l)ogin, or (q)uit? l
Enter your username: Dhanush
Enter your password:
Incorrect password.
Do you want to (r)egister, (l)ogin, or (q)uit? l
Enter your username: Dhanush
Enter your password:
Login successful.
Do you want to (r)egister, (l)ogin, or (q)uit? q
Exiting the system.
```

FINAL DESCRIPTION :

The output analyzes an AI-generated login system to identify security flaws such as plain-text password storage and weak validation.
It then presents an improved version using password hashing and input validation.
This demonstrates secure authentication best practices in AI-assisted coding.

Task5: 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.

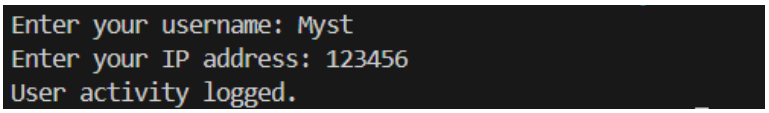
• Explanation of privacy-aware logging principles.

CODE :

```
"""generate a Python script that logs user activity (username, IP address, timestamp)."""
import logging
from datetime import datetime
# Configure logging settings
logging.basicConfig(filename='user_activity.log', level=logging.INFO,
                    format='%(asctime)s - %(message)s')
def log_user_activity(username, ip_address):
    """Logs the user activity with username, IP address, and timestamp."""
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    log_message = f"User: {username}, IP Address: {ip_address}, Timestamp: {timestamp}"
    logging.info(log_message)
# Example usage
if __name__ == "__main__":
    user = input("Enter your username: ")
    ip = input("Enter your IP address: ")
    log_user_activity(user, ip)
    print("User activity logged.")
```

```
"""
# This script logs user activity including username, IP address, and timestamp.
# It uses Python's built-in logging module to write log entries to a file named 'user
# _activity.log'.
# Each log entry includes the date and time of the activity, formatted for clarity.
# The log_user_activity function takes a username and IP address as input,
# generates a timestamp, and logs the information in a structured format.
# Example usage is provided to demonstrate how to log user activity.
"""
```

OUTPUT :



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.