AI-Assisted Coding

Name: P. Archith Roll Number: 2303A52258 Batch: 45

Assignment Number: 5.5

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.

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."

Expected Output:

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

Prompt:

Generate two Python Code to check whether a given number is prime:

A basic (naïve) method that checks divisibility up to n − 1.

An optimized method that checks divisibility only up to √n.

For each method:

Provide clear, correct Python code.

Explain the working logic step by step.

Analyze and state the time complexity using Big-O notation.

Code:

```
"""2303A52324"""
import math
# Naïve method
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

# Optimized method (check up to sqrt(n))
def is_prime_sqrt(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return n == 2
    r = math.isqrt(n)
    for i in range(3, r + 1, 2):
        if n % i == 0:
            return False
    return True
# Example quick test
if __name__ == "__main__":
    test_values = [1, 2, 3, 4, 17, 18, 19, 97, 100, 101]
    print("Naive results:", {v: is_prime_naive(v) for v in test_values})
    print("Sqrt  results:", {v: is_prime_sqrt(v) for v in test_values})
```

Output:

```
py'
Naive results: {1: False, 2: True, 3: True, 4: False, 17: True, 18: False, 19: True, 97: True, 100: False, 101: True}
Sqrt  results: {1: False, 2: True, 3: True, 4: False, 17: True, 18: False, 19: True, 97: True, 100: False, 101: True}
PS D:\3-2\AI Assitant Coding>
```

Justification:

The naïve prime-checking method tests all possible divisors from 2 to $n - 1$, which results in a linear time complexity of **O(n)** and becomes inefficient for large values of n.

The optimized method improves efficiency by observing that any composite number must have a factor less than or equal to $\sqrt{n}$. By limiting the divisor checks up to $\sqrt{n}$ and skipping even numbers, the number of iterations is significantly reduced, achieving a time complexity of **O(√n)**.

Thus, the optimized approach performs faster and scales better while producing the same correct results as the naïve method.

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: 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:

```
"""2303A52324"""
"""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 problem.
    # 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 = int(input("Enter a positive integer to find the nth Fibonacci number: "))
fib_number = fibonacci(n)
print(f"The {n}th Fibonacci number is: {fib_number}")
```

Output:

```
PS D:\3-2\AI Assitant Coding>  d:; cd 'd:\3-2\AI Assitant Coding'; & '
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugp
Enter a positive integer to find the nth Fibonacci number: 6
The 6th Fibonacci number is: 8
PS D:\3-2\AI Assitant Coding>
```

Justification:

The Fibonacci program uses recursion by defining the problem in terms of smaller subproblems. Base cases for n = 0 and n = 1 are necessary to stop infinite recursion and provide correct initial values. For n > 1, the function recursively computes the sum of the previous two Fibonacci numbers, ensuring correctness by following the mathematical definition of the Fibonacci sequence.

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: 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.

Code:

```
"""2303A52324"""
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              |
|-----------------------|------------------------------------------------|-----|---------------------------------------|
| ZeroDivisionError     | Raised when attempting to divide by zero.      | divide_numbers(10, 0) returns "Error: Cannot divide by zero." |
| TypeError             | Raised when inputs are of incorrect type.      | divide_numbers(10, 'a') returns "Error: Both inputs must be numbers." |
| NameError             | Raised when a variable is not defined.         | divide_numbers(10, None) returns "Error: One of the variables 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 with the error details. |
"""
print(explanation_errors)
```

Output:

```
-2.0
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61611' '--' 'D:\3-2\AI Assitant Coding\Task-5.5.py'
5.0
Error: Cannot divide by zero.
Error: Both inputs must be numbers.
-2.0
Error: Both inputs must be numbers.
Error: Cannot divide by zero.
Error: Both inputs must be numbers.
-2.0
Error: Both inputs must be numbers.
-2.0
Error: Both inputs must be numbers.
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 by zero." |
| TypeError             | Raised when inputs are of incorrect type.      | divide_numbers(10, 'a') returns "Error: Both inputs must be numbers." |
| NameError             | Raised when a variable is not defined.         | divide_numbers(10, None) returns "Error: One of the variables is not defined." |
| ValueError            | Raised when an invalid value is passed.        | divide_numbers(10, -5) returns "Error: Invalid value provided." |
| ZeroDivisionError     | Raised when attempting to divide by zero.      | divide_numbers(10, 0) returns "Error: Cannot divide by zero." |
| TypeError             | Raised when inputs are of incorrect type.      | divide_numbers(10, 'a') returns "Error: Both inputs must be numbers." |
| NameError             | Raised when a variable is not defined.         | divide_numbers(10, None) returns "Error: One of the variables is not defined." |
| ValueError            | Raised when an invalid value is passed.        | divide_numbers(10, -5) returns "Error: Invalid value provided." |
| TypeError             | Raised when inputs are of incorrect type.      | divide_numbers(10, 'a') returns "Error: Both inputs must be numbers." |
| NameError             | Raised when a variable is not defined.         | divide_numbers(10, None) returns "Error: One of the variables is not defined." |
| ValueError            | Raised when an invalid value is passed.        | divide_numbers(10, -5) returns "Error: Invalid value provided." |
| NameError             | Raised when a variable is not defined.         | divide_numbers(10, None) returns "Error: One of the variables is not defined." |
| ValueError            | Raised when an invalid value is passed.        | divide_numbers(10, -5) returns "Error: Invalid value provided." |
| 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 with the error details. |

PS D:\3-2\AI Assitant Coding>
```

Justification:

This program is justified because it improves **robustness and reliability** by preventing program crashes due to invalid input or runtime errors. By handling specific exceptions separately, the function provides **clear, user-friendly error messages** instead of abrupt termination. The use of a general exception handler further ensures safety against unforeseen errors, making the function suitable for real-world applications where input validation and error handling are essential.

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: 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:

```
"""2303A52324"""
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:
```

Output:

Justification: This program is justified because it follows **secure authentication practices** by avoiding plain-text password storage and using cryptographic hashing for verification. Hashing ensures confidentiality of user credentials, while input masking enhances privacy during login. The modular design improves readability, reusability, and maintainability, making the system safer and more reliable for real-world applications.

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: Generate a python script that logs users activities with username,IP address timestamps to a log file.

Code:

```
#generate a python script that logs users activities with username,IP address timestamps to a log file.
import socket,datetime

u=input("Username: ")
ip=socket.gethostbyname(socket.gethostname())
t=datetime.datetime.now().isoformat()

open("activity.log","a").write(f"{u},{ip},{t}\n")
print("Logged")
```

Output:

```
t=datetime.datetime.now().isoformat()

open("activity.log","a").write(f"{u},{ip},{t}\n")
print("Logged")
```

```
Logged
```

Justification: This task highlights the importance of privacy-aware data logging by critically examining how user activity logs are generated and stored. While logging usernames, IP addresses, and timestamps can be useful for monitoring and security purposes, recording such information in raw form may unnecessarily expose sensitive personal data and increase the risk of misuse or data breaches. By identifying these privacy risks and improving the logging mechanism through techniques such as anonymization, masking, and data minimization, the task demonstrates responsible handling of user information. This approach ensures compliance with ethical data practices and privacy regulations while still maintaining the essential functionality of activity monitoring, thereby promoting secure, transparent, and privacy-conscious system design.