

## **School of Computer Science and Artificial Intelligence**

---

### **Lab Assignment # 5.5**

---

**Program : B. Tech (CSE)**  
**Specialization : AIML**  
**Course Title : AI Assisted**  
**Coding Course Code: 23CS002PC304**  
**Semester : VI**  
**Academic Session : 2025-2026**  
**Name of Student : R . Deekshith Goud**  
**Enrollment No. : 2303A52104**  
**Batch No. : 33**  
**Date : 20/01/26**

## **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.
-

## Task Description – 1: (Transparency in Algorithm Optimization)

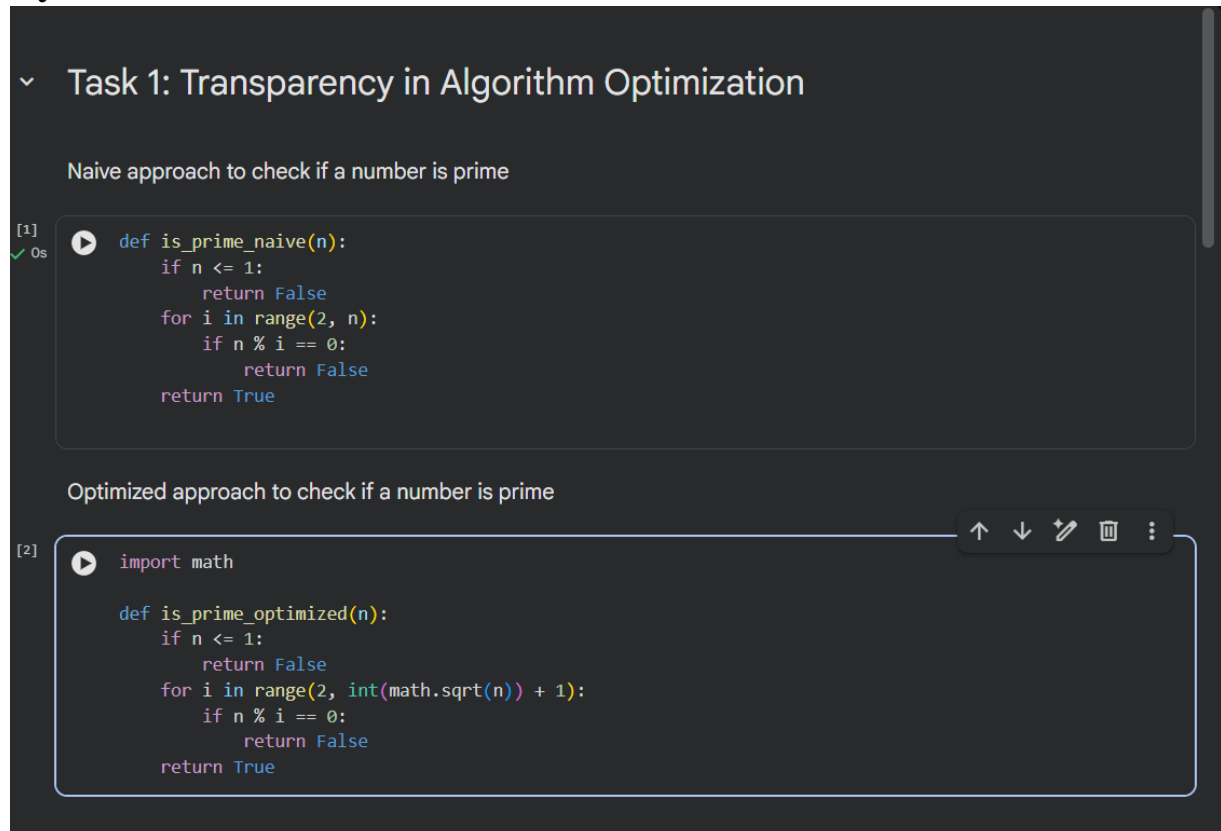
**Problem:** Use AI to generate two solutions for checking prime numbers.

1. Naive approach
2. Optimized approach

### Prompt Used (Zero-shot)

Generate Python code for two prime-checking methods and explain how the optimized version improves performance.

### Python Code:



The screenshot shows a code editor with a dark theme. At the top, there's a tab labeled 'Task 1: Transparency in Algorithm Optimization'. Below the tab, the text 'Naive approach to check if a number is prime' is displayed. The code for the naive approach is shown in a code block, starting with a play button icon and a status '[1] 0s'. The code is as follows:

```
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Below this, the text 'Optimized approach to check if a number is prime' is displayed. The code for the optimized approach is shown in a code block, starting with a play button icon and a status '[2]'. The code is as follows:

```
import math

def is_prime_optimized(n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True
```

## Explanation & Time Complexity

- **Naive Method**
  - Checks divisibility from 2 to  $n-1$
  - Time Complexity:  $O(n)$
  - Inefficient for large numbers
- **Optimized Method**
  - Checks divisibility only up to  $\sqrt{n}$
  - Time Complexity:  $O(\sqrt{n})$
  - Faster because factors repeat after square root

## Task – 2: Transparency in Recursive Algorithms Objective

**Problem:** Generate a recursive Fibonacci function with clear explanations.

### Prompt Used (One-shot)

1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls

### Python Code:

#### Task 2: Transparency in Recursive Algorithms

recursive Fibonacci function

```
[3]
✓ 0s def fibonacci(n):
      # Base case: Fibonacci of 0 is 0
      if n == 0:
          return 0

      # Base case: Fibonacci of 1 is 1
      if n == 1:
          return 1

      # Recursive case:
      # Fibonacci of n is the sum of Fibonacci of (n-1) and (n-2)
      return fibonacci(n - 1) + fibonacci(n - 2)
```

### Explanation

- **Base Cases**
    - fibonacci(0) returns 0
    - fibonacci(1) returns 1
  - **Recursive Call**
    - Function keeps calling itself with smaller values
    - Stops when base cases are reached
  - **Execution Match**
    - Each recursive call mirrors the explanation exactly
    - Ensures transparency and correctness
-

## Task Description – 3: Transparency in Error Handling.

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

### Prompt Used (Few-shot)

“Generate code with proper error handling and clear explanations for each exception.”

### Python Code:

#### Task 3: Transparency in Error Handling

Python program that reads a file with proper error handling.

```
def read_file(filename):  
    try:  
        file = open(filename, "r")  
        data = file.read()  
        file.close()  
        return data  
  
    except FileNotFoundError:  
        print("Error: File not found.")  
  
    except PermissionError:  
        print("Error: Permission denied.")  
  
    except Exception as e:  
        print("Unexpected error:", e)
```

### Explanation of Exceptions:

- **FileNotFoundError**
  - Occurs when the file does not exist
- **PermissionError**
  - Occurs when access rights are insufficient
- **Generic Exception**
  - Catches unexpected runtime errors

Each explanation directly corresponds to actual Python runtime behavior, ensuring transparency.

---

## Task Description – 4: Security in User Authentication

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

### Prompt :

Check whether the AI uses secure password handling practices.

### Initial AI-Generated Issue (Security Flaws)

- Passwords stored in **plain text**
- No hashing
- Weak input validation

### Security Risks

- Data breaches
- Password reuse attacks
- Unauthorized access

## Task 4: Security in User Authentication

### Secure Revised Version

```
import hashlib

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def login(stored_hash, entered_password):
    return stored_hash == hash_password(entered_password)
```

### Best Practices for Secure Authentication

- Always hash passwords (never store plain text)
  - Use strong hashing algorithms (bcrypt, SHA-256, etc.)
  - Validate user inputs
  - Implement rate limiting and authentication checks
-

## Task Description – 5: Privacy in Data Logging

**Problem:** Determine the minimum of three numbers without using min().

**Prompt Used :** Examine whether sensitive data is logged unnecessarily or insecurely.

### Original AI Logging Risks

- Logs **username, IP address, timestamp**
- IP addresses are sensitive personal data
- Logs stored without masking or protection

### Python Code:

#### Task 5: Privacy in Data Logging

```
import datetime

def log_activity(username):
    masked_user = username[0] + "****"
    timestamp = datetime.datetime.now()
    print(f"User: {masked_user}, Time: {timestamp}")
```

### Privacy-Aware Logging Principles

- Log only necessary information
- Mask or anonymize personal data
- Avoid storing IP addresses unless required
- Protect log files from unauthorized access

---

## Conclusion

This lab highlights that **AI-generated code is not automatically ethical or secure.**

Developers are responsible for:

- Reviewing AI outputs
- Ensuring transparency and correctness
- Preventing security and privacy violations
- Maintaining accountability for deployed code

Ethical AI coding requires **human judgment**, not blind trust in AI tools.