

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasank Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
CourseCode	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week3 –	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number: 5.5(Present assignment number)/24(Total number of assignments)			

Q.No.	Question	Expected Time to complete

1	<p>Lab 5: Ethical Foundations – Responsible AI Coding Practices</p> <p>Lab Objectives:</p> <ul style="list-style-type: none"> • 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. <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> • 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. <p>Task Description #1 (Transparency in Algorithm Optimization)</p> <p>Task: Use AI to generate two solutions for checking prime numbers:</p> <ul style="list-style-type: none"> • Naive approach(basic) • Optimized approach <p>Prompt:</p> <p>“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”</p> <p>Expected Output:</p> <ul style="list-style-type: none"> • Code for both methods. • Transparent explanation of time complexity. • Comparison highlighting efficiency improvements. <hr/> <p>Task Description #2 (Transparency in Recursive Algorithms)</p> <p>Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.</p> <p>Instructions:</p> <ol style="list-style-type: none"> 1. Ask AI to add clear comments explaining recursion. 	Week3 -
---	--	---------

Task 1:
Prompt :

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

Suggested code Naive Prime Checking Approach :

```
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 Prime-Checking Approach :

```
def is_prime_optimized(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False

    return True
```

Time Complexity :

- $O(n)$
- The loop may run up to $n-2$ iterations in the worst case

Explanation:

- The function checks divisibility from 2 to $n-1$
- If any divisor is found, the number is not prime
- If no divisor is found, the number is prime

Performance Comparison :

Aspect	Naive Approach	Optimized Approach
Loop range	2 to $n-1$	2 to \sqrt{n}
Time complexity	$O(n)$	$O(\sqrt{n})$
Even number handling	Not optimized	Skipped
Efficiency for large n	Poor	High
Practical usability	Low	High

Task 2 : Transparency in Recursive Algorithms

Prompt :

Write a recursive Python function to calculate Fibonacci numbers.
 Add clear comments explaining how recursion works.
 Explain the base cases and recursive calls, and ensure the explanation matches the actual execution of the code.

Suggested Code Recursive Fibonacci Function :

```
def fibonacci(n):
    # Base case 1:
    # If n is 0, the Fibonacci value is 0
    if n == 0:
        return 0

    # Base case 2:
    # If n is 1, the Fibonacci value is 1
    if n == 1:
        return 1

    # Recursive case:
    # The function calls itself to compute
    # fibonacci(n-1) and fibonacci(n-2)
    return fibonacci(n - 1) + fibonacci(n - 2)

# User input
num = int(input("Enter a number: "))
print("Fibonacci number:", fibonacci(num))
```

Sample output:

Enter a number: 4

Fibonacci number: 3

Explanation of Recursive Calls :

For values of n greater than 1:

- The function computes
Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
- Each call reduces the problem size until a base case is reached
- Results are combined while returning from recursive calls

Task 3 : Transparency in Error Handling

Prompt:

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

Suggested Code :

```

def read_and_process_file(filename):
    try:
        # Attempt to open the file in read mode
        with open(filename, 'r') as file:
            data = file.read()

        # Simple data processing: count number of lines
        lines = data.splitlines()
        print("Number of lines in file:", len(lines))

    except FileNotFoundError:
        # Raised when the file does not exist
        print("Error: The specified file was not found.")

    except PermissionError:
        # Raised when the program does not have permission to read the file
        print("Error: You do not have permission to access this file.")

    except UnicodeDecodeError:
        # Raised when file encoding is incompatible
        print("Error: File encoding is not supported.")

    except Exception as e:
        # Catches any other unexpected errors
        print("An unexpected error occurred:", e)

    else:
        # Executes if no exception occurs
        print("File processed successfully.")

    finally:
        # Executes regardless of whether an exception occurs
        print("File operation completed.")

# User input
filename = input("Enter file name: ")
read_and_process_file(filename)

```

Sample Output :

```

Enter file name: data.txt
Error: The specified file was not found.
Error: The specified file was not found.
File operation completed.

```

Validation: Explanation vs Runtime Behavior:

Scenario	Runtime Behavior	Explanation Match
File does not exist	“File not found” message printed	Yes
No read permission	Permission error message	Yes
Unsupported encoding	Encoding error message	Yes
Valid file	Line count + success message	Yes
Any unexpected issue	Generic error message	Yes

Task 4 : Security in User Authentication

Prompt :

Generate a Python-based user login system.

Store and verify passwords securely using hashing and include basic input validation

Suggested Code AI-Generated Login System (Initial / Insecure Version) :

```
def login(username, password):
    stored_username = "admin"
    stored_password = "admin123"    # Plain-text password (INSECURE)

    if username == stored_username and password == stored_password:
        return "Login successful"
    else:
        return "Invalid credentials"

# User input
u = input("Enter username: ")
p = input("Enter password: ")
print(login(u, p))
```

Sample output :

```
Enter username: admin
Enter password: admin123
Login successful
```

Revised Secure Version (With Hashing & Validation):

```

import hashlib
import getpass

# Pre-stored hashed password (hash of "admin123")
STORED_USERNAME = "admin"
STORED_PASSWORD_HASH = hashlib.sha256("admin123".encode()).hexdigest()

def secure_login(username, password):
    if not username or not password:
        return "Invalid input"

    # Hash the entered password
    password_hash = hashlib.sha256(password.encode()).hexdigest()

    if username == STORED_USERNAME and password_hash == STORED_PASSWORD_HASH:
        return "Login successful"
    else:
        return "Invalid credentials"

# User input
username = input("Enter username: ")
password = getpass.getpass("Enter password: ")
print(secure_login(username, password))

```

Sample output :

```

Enter username: admin
Enter password:
Login successful

```

Identification of Security Flaws :

Security Issue	Explanation
Plain-text password storage	Password is stored and compared in readable form
No password hashing	Easily compromised if code or database is leaked
No input validation	Accepts empty or malformed inputs
No protection against brute force	Unlimited login attempts
Password visible while typing	input() exposes password

Task Description 5 : (Privacy in Data Logging)

Prompt:

Suggested code AI-Generated Logging Script (Initial / Privacy-Risky Version) :

```
from datetime import datetime

def log_user_activity(username, ip_address):
    timestamp = datetime.now()
    with open("activity.log", "a") as file:
        file.write(f"{timestamp} | User: {username} | IP: {ip_address}\n")

# Example usage
log_user_activity("alice", "192.168.1.101")
```

Identified Privacy Risks in Logging

Privacy Risk	Explanation
Logging full IP address	Can be used to identify user location
Logging real usernames	Directly identifies individuals
No data minimization	Logs more data than required
Plain-text storage	Log file can be read if accessed
No retention policy	Data stored indefinitely

Improved Privacy-Aware Logging Version :

```

from datetime import datetime
import hashlib

def mask_ip(ip_address):
    # Mask last octet of IP address
    parts = ip_address.split(".")
    parts[-1] = "xxx"
    return ".".join(parts)

def anonymize_username(username):
    # Hash username to anonymize identity
    return hashlib.sha256(username.encode()).hexdigest()[:8]

def log_user_activity(username, ip_address):
    timestamp = datetime.now()
    anon_user = anonymize_username(username)
    masked_ip = mask_ip(ip_address)

    with open("activity.log", "a") as file:
        file.write(f"{timestamp} | UserID: {anon_user} | IP: {masked_ip}\n")

# Example usage
log_user_activity("alice", "192.168.1.101")

```

How Privacy Is Improved:

Improvement	Description
Username anonymization	Prevents direct user identification
IP masking	Reduces location precision
Data minimization	Logs only necessary information
Reduced exposure	Limits sensitive data in plain text

Privacy-Aware Logging Principles:

- Log only what is strictly necessary (data minimization)
- Avoid storing personally identifiable information (PII)
- Mask or anonymize sensitive fields
- Protect log files from unauthorized access
- Define data retention and deletion policies
- Comply with privacy regulations (GDPR, etc.)