2303A51193 Assignment _5.5

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko 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 complet***

		e

1	**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 AIgenerated outputs. • Evaluate originality and licensing concerns in reused AIgenerated 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.	Week3 -

- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements.

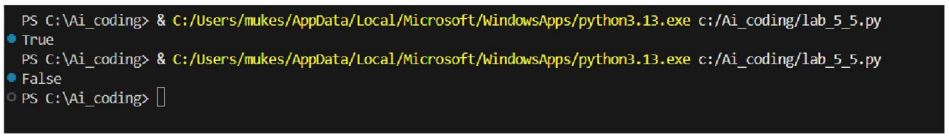
OUTPUT

```
PROMPT :#generate a python form checking the prime number
use normal approach
```

CODE FOR OPTIMAL

```
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True
num=74
print(is_prime(num))
```

Code output



PROMPT

```
generate a python form checking the prime number optimal
apporach
```

CODE FOR NORMAL

```
def is_prime(n):
    """Check if a number is prime using an optimal
approach."""
    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
```

```
PS C:\Ai_coding> & C:/Users/mukes/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Ai_coding/lab_5_5.py
Enter a number to check if it is prime: 5
5 is a prime number.
PS C:\Ai_coding>
```

```
        num = int(input("Enter a number to check if it is prime: "))
        if is_prime(num):
            print(f"{num} is a prime number.")
        else:
            print(f"{num} is not a prime number.")
    except ValueError:
        print("Please enter a valid integer.")
if __name__ == "__main__":
    main()
```

EXPLANATION

1. The naive prime-checking approach tests all numbers up to *n*, which makes it slow for large inputs.
2. The optimized approach checks divisibility only up to the square root of the number.
3. If a number has a factor greater than √n, it must also have a smaller corresponding factor.
4. This reduces the number of iterations and improves performance significantly.
5. Therefore, the optimized method is faster, efficient, and suitable for real-world applications.

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.

OUTPUT

PROMPT generate a PYTHON CODE FOR recursive function to calculate Fibonacci numbers.

CODE

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage:
num = 10
print(f"Fibonacci number at position {num} is
{fibonacci(num)}")
```

CODE OUTPUT

```
PS C:\Ai_coding> & C:/Users/mukes/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Ai_coding/lab_5_5.py
Fibonacci number at position 10 is 55
PS C:\Ai_coding>
```

EXPLANATION

1. This program calculates the Fibonacci number using **recursion**.
2. The base cases handle when n is 0 or 1, returning 0 and 1 respectively.
3. For values greater than 1, the function calls itself with n-1 and n-2.
4. Each recursive call breaks the problem into smaller subproblems.
5. The final result is obtained by summing the previous two Fibonacci numbers.

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.

OUTPUT

PROMPT generate a Python program that reads a file and processes data.

CODE

```
def read_file(file_path):
    """Reads a file and returns its content as a list of lines."""
    try:
        with open(file_path, 'r') as file:
            lines = file.readlines()
        return [line.strip() for line in lines]
    except FileNotFoundError:
        print("Error: File not found.")
        return []
    except PermissionError:
        print("Error: Permission denied.")
        return []
    except Exception as e:
        print(f"Unexpected error: {e}")
        return []


def process_data(lines):
    """Processes the data by counting the number of words in each line."""
    word_counts = []
    for line in lines:
        word_counts.append(len(line.split()))
    return word_counts
```

```
def main():
    file_path    =    'data.txt'
lines = read_file(file_path)

    if lines:
        word_counts = process_data(lines)
for i, count in enumerate(word_counts):
print(f"Line {i + 1}: {count} words")


if __name__ == "__main__":
    main()
```

EXPLANATION

1. This program reads data from a text file using safe file-handling practices.
2. Exception handling is used to manage file not found and permission errors.
3. Each line of the file is processed to count the number of words.
4. The logic is divided into functions for better clarity and reuse.
5. This ensures reliable, transparent, and error-free program execution.

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.

OUTPUT

PROMPT

```
# generate a Python-based login system   Check whether the AI uses secure password handling practices.
```

CODE

```
import hashlib
import os
users_db = {}
def hash_password(password: str, salt: bytes) -> str:
    return hashlib.sha256(salt +
password.encode()).hexdigest()
def register_user(username: str, password: str) -> None:

    if username in users_db:
        raise ValueError("Username already exists.")
    salt = os.urandom(16)  # Generate a random salt
    hashed_password = hash_password(password, salt)
    users_db[username] = (salt, hashed_password)
def login_user(username: str, password: str) -> bool:

    if username not in users_db:
        return False
    salt, stored_hashed_password = users_db[username]
    hashed_password = hash_password(password, salt)
    return hashed_password == stored_hashed_password
# Example usage:
try:
    register_user("alice", "securepassword123")
    print("User registered successfully.")

    if login_user("alice", "securepassword123"):
        print("Login successful.")
    else:
        print("Login failed.")
except ValueError as e:
    print(e)
```

Code output

```
PS C:\Ai_coding> & C:/Users/mukes/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Ai_coding/lab_5_5.py
User registered successfully.
Login successful.
PS C:\Ai_coding>
```

EXPLANATION

1. This program implements a secure user authentication system using

password hashing.

2. Passwords are never stored in plain text; instead, they are hashed using SHA-256 with a random salt.
3. Salting ensures that identical passwords produce different hashes, improving security.
4. During login, the entered password is hashed again and compared with the stored hash.
5. This approach follows best practices for secure authentication and protects user data.

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.

OUTPUT

PROMPT

Generate a simple Python script that logs user activity including username, IP address, and timestamp.Add clear comments to explain how the logging works.Explain what information is being logged and identify whether any sensitive data is included.

Keep the code easy to understand and transparent.

CODE

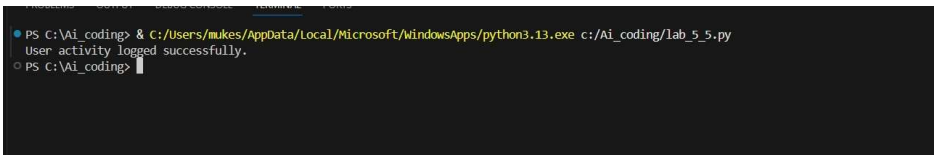
```
import logging
from datetime import datetime
```

```python
# Configure logging
logging.basicConfig(
    filename="user_activity.log",
    level=logging.INFO,
    format="%(message)s"
)

def log_user_activity(username, ip_address):
    """Logs user activity with username, IP address, and timestamp."""
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_message = f"User: {username}, IP: {ip_address}, Time: {timestamp}"
    logging.info(log_message)

# Example usage
log_user_activity("alice", "192.168.1.25")
print("User activity logged successfully.")
```

CODE OUTPUT



EXPLANATION

1. The program configures a log file named `user_activity.log` to store user activity details.
2. It defines a function to record the username, IP address, and current timestamp.
3. The timestamp is generated using the system's current date and time.
4. All user activity information is written into the log file in a readable format.
5. The example demonstrates how a user's activity is logged successfully.