SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 5

Name of Student : Yuvaraj Bandi
Enrollment No. : 2303A51440
Batch No. 21

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 methods to check if a number is prime.
One should be a basic method.
The other should be an optimized method using square root.
Also explain the time complexity and compare both methods.

Code:-

```
'''Generate Python code for two methods to check if a number is prime.
One should be a basic method.
The other should be an optimized method using square root.
Also explain the time complexity and compare both methods'''
import math
def is_prime_basic(n):
    """
    Basic prime checker.
    Time Complexity: O(n)
    """
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """
    Optimized prime checker using square root.
    Time Complexity: O(√n)
    """
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True
if __name__ == "__main__":
    test_numbers = [2, 17, 20, 97, 100]

    for num in test_numbers:
        basic = is_prime_basic(num)
        optimized = is_prime_optimized(num)
        print(f"{num}: Basic={basic}, Optimized={optimized}")
```

Output:-

SR UNIVERSITY

Justification:-
The program includes two methods for checking prime numbers: a naive approach that checks divisibility from 2 to n-1, and an optimized approach that reduces the number of checks by eliminating even numbers and using a 6k ± 1 rule. Additionally, it handles invalid input by catching Value Error exceptions when the user inputs non-integer values. This ensures robustness and user-friendliness.

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.

Prompt:-
Generate a recursive Python function to calculate Fibonacci numbers.
Take the number as input from the user.
Print the Fibonacci result for that input.
Add comments explaining base case and recursive call

Code:-

```
'''Generate a recursive Python function to calculate Fibonacci numbers.
Take the number as input from the user.
Print the Fibonacci result for that input.
Add comments explaining base case and recursive call.'''
# Function to calculate Fibonacci numbers recursively
def fibonacci(n):
    # Base case: if n is 0 or 1, return n (F(0) = 0, F(1) = 1)
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive call: F(n) = F(n-1) + F(n-2)
        return fibonacci(n - 1) + fibonacci(n - 2)
# Take input from the user
num = int(input("Enter a number to calculate its Fibonacci: "))
# Calculate and print the Fibonacci result for the input number
result = fibonacci(num)
print(f"The Fibonacci number for {num} is: {result}")
```

Output:-

```
PS C:\Users\2303A\Downloads\ai assistant coding> & C
/2303A/Downloads/ai assistant coding/assn5.py"
Enter a number to calculate its Fibonacci: 19
The Fibonacci number for 19 is: 4181
```

Justification:-
Recursion is a programming technique where a function calls itself to solve a problem. In this case, the `fibonacci` function calculates the nth Fibonacci number by recursively calling itself with smaller values of n until it reaches the base cases (n=0 or n=1). The base cases are essential because they stop the recursion and provide a direct answer for the simplest subproblems. The recursive calls break down the problem into smaller subproblems, which are then solved by further recursive calls.

Task Description #3 (Transparency in Error Handling)
Task: Use AI to generate a Python program that reads a file and processes data.

SR UNIVERSITY

Prompt:-

Generate a Python program that asks the user to enter a file name.
Read the file and count the number of lines.
Use proper exception handling.
Display appropriate error messages if the file is not found or cannot be accessed

Code:-

```
'''Generate a Python program that asks the user to enter a file name.
Read the file and count the number of lines.
Use proper exception handling.
Display appropriate error messages if the file is not found or cannot be accessed.'''

# Ask the user to enter a file name
file_name = input("Enter the file name: ")

try:
    # Open the file in read mode
    with open(file_name, 'r') as file:
        # Read all lines from the file
        lines = file.readlines()
        # Count the number of lines
        line_count = len(lines)
        print(f"The file '{file_name}' has {line_count} lines.")
except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found.")
except IOError:
    print(f"Error: The file '{file_name}' could not be accessed.")
```

Output:-

```
PS C:\Users\2303A\OneDrive\Desktop\python course> & C:\Users\2303A\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/2303A/OneDrive/Desktop/python course/ai.py"
Enter the file name: yuvaraj.1
Error: File not found.
PS C:\Users\2303A\OneDrive\Desktop\python course>
```

Justification:-

The code handles various exceptions that might occur during file reading:
FileNotFoundError: Raised when the specified file does not exist.
IOError: Raised when an I/O error occurs while trying to read the file.
Exception: Catches any other unexpected errors.

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.

Prompt:-

Generate a Python login system that takes username and password from user input.
Validate the credentials.
Then improve it using password hashing.
Keep inputs dynamic.

Code:-

```
'''Generate a Python login system that takes username and password from user input.
Validate the credentials.
Then improve it using password hashing.
Keep inputs dynamic.'''
# Task 4: Secure Login System

import hashlib

# Simple database with hashed passwords
user_db = {
    "yuvaraj": hashlib.sha256("yuvaraj123".encode()).hexdigest(),
    "user2": hashlib.sha256("mypassword".encode()).hexdigest()
}

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

def validate_credentials(username, password):
    if username in user_db:
        return user_db[username] == hash_password(password)
    return False

# Dynamic input
username = input("Enter username: ")
password = input("Enter password: ")
                 (variable) username: str

if validate_credentials(username, password):
    print("Login Successful!")
else:
    print("Invalid Username or Password.")
```

Output:-

```
PS C:\Users\2303A\OneDrive\Desktop\python course> & C:\Users\2303A\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/2303A/OneDrive/Desktop/python course/ai.py"
Enter username: yuvaraj
Enter password: yuvaraj123
Login Successful!
PS C:\Users\2303A\OneDrive\Desktop\python course>
```

Justification:-

Secure password handling is crucial to protect user data. By hashing passwords, we ensure that even if the database is compromised, the actual passwords remain secure. Using a strong hashing algorithm like SHA-256 adds an additional layer of security. This approach prevents storing plain-text passwords, reducing the risk of unauthorized access.

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.

Prompt:-

Generate a Python script that takes username and IP address as user input.
Log the username, IP address, and current timestamp into a file.
Then improve it by masking sensitive data and add clear comments

Code:-

```
'''Generate a Python script that takes username and IP address as user input.
Log the username, IP address, and current timestamp into a file.
Then improve it by masking sensitive data and add clear comments'''
import datetime
def log_user_info(username, ip_address):
    # Get the current timestamp in a readable format
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Mask the username for privacy (show only first and last character)
    masked_username = username[0] + "****" + username[-1] if len(username) > 2 else username

    # Mask the IP address for privacy (show only the first and last octet)
    masked_ip = ip_address.split('.')[0] + ".***.***." + ip_address.split('.')[-1]

    # Create a log entry with the timestamp, masked username, and masked IP address
    log_entry = f"{timestamp} - Username: {masked_username}, IP Address: {masked_ip}\n"

    # Append the log entry to the file 'user_log.txt'
    with open("user_log.txt", "a") as log_file:
        log_file.write(log_entry)
if __name__ == "__main__":
    # Prompt the user to enter their username and IP address
    username = input("Enter your username: ")
    ip_address = input("Enter your IP address: ")

    # Log the user information with masking
    log_user_info(username, ip_address)

    # Inform the user that their information has been logged successfully
    print("User information logged successfully!")
```

Output:-

```
PS C:\Users\2303A\OneDrive\Desktop\python course> & C:\Users\2303A\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/2303A/OneDrive/Desktop/python course/ai.py"
Enter username: yuvaraj
Enter IP address: 192.10.678
User information logged successfully!
PS C:\Users\2303A\OneDrive\Desktop\python course>
```

Justification:-
The code handles various exceptions that might occur during file reading:
FileNotFoundError: Raised when the specified file does not exist.
IOError: Raised when an I/O error occurs while trying to read the file.
Exception: Catches any other unexpected errors.