SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 5

Name of Student : U MANOJ
Enrollment No. : 2303A51524
Batch No. : 22

Task Description #1 (Transparency in Algorithm Optimization)
Task: Use AI to generate two solutions for checking prime numbers:-

- **Naive approach(basic),**
- **Optimized approach**

Prompt:-
write a python program of a prime number checking in two methods navie and optimised approach. and add when in input letters it gives invalid input

Code:-

Assignments Codes > Assignment5.py > ...

```python
# Task-1
#write a python program of a prime number checking in two methods navie and optimised approach.
# and add when in input letters it gives invalid input
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    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
    return True
try:
    number = int(input("Enter a number to check if it is prime: "))
    print("Naive approach:", is_prime_naive(number))
    print("Optimized approach:", is_prime_optimized(number))
except ValueError:
    print("Invalid input. Please enter a valid integer.")

```

Output:-

```
Enter a number to check if it is prime: 7
Naive approach: True
Optimized approach: True
```

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:-
write a python program to print to calculate fibonacci series using recursion function and clear explanation of recursion, base case and recursive calls.

Code:-

```
    # Task-2
#write a python program to print to calculate fibonacci series using recursion
def fibonacci(n):
    # Base case: the first two Fibonacci numbers are 0 and 1
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive call: sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
print("Fibonacci series:")
for i in range(num_terms):
    print(fibonacci(i), end=" ")
print()  # for a new line after the series

```

Output:-

```
Enter the number of terms in the Fibonacci series: 5
Fibonacci series:
0 1 1 2 3
PS C:\AIAC LAB>
```

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.
Prompt:-
write a python program to read file and processes data using error handling and clear explanations for each exception.

Code:-

```python
#Task-3
# write a python program to read file and  processes data using error handling and clear e
try:
    file_name = input("Enter the file name to read: ")
    with open(file_name, 'r') as file:
        data = file.read()
        print("File content:")
        print(data)
except FileNotFoundError:
    print("Error: The file was not found. Please check the file name and try again.")
except IOError:
    print("Error: An I/O error occurred while trying to read the file.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

```

Output:-

```
Enter the file name to read: AIAC_Lab Assignment_1.docx
Error: The file was not found. Please check the file name and try again.
```

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:-

write a python program to build a login sytem and secure password handling practices.

Code:-

```python
# Task-4
#write a python program to build a login sytem and secure password handling practices.
import hashlib
def hash_password(password):
    # Hash the password using SHA-256
    return hashlib.sha256(password.encode()).hexdigest()
def verify_password(stored_password_hash, provided_password):
    # Verify the provided password against the stored hash
    return stored_password_hash == hash_password(provided_password)
# Simulated user database
user_db = {
    "PhaniKumar": hash_password("Phani@123"),
    "kirito": hash_password("SwordArtOnline"),
}
username = input("Enter your username: ")
password = input("Enter your password: ")
if username in user_db:
    if verify_password(user_db[username], password):
        print("Login successful!")
    else:
        print("Invalid password.")
else:
    print("Username not found.")
```

Output:-

```
Enter your username: usilla manoj
Enter your password: MANOJ@2005
```

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:-
write a python program to script the logs user activities Example: username, time stamp, IP address etc Examine sensitive data is logged unnecessarily or insecurely and Improved version with minimal, anonymized, or masked logging system.

Code:-

```
# Task-5
#write a python program to script the logs user activities Example: username,
import logging
from datetime import datetime
import socket
# Configure logging
logging.basicConfig(
    filename='user_activities.log',
    level=logging.INFO,
    format='%(asctime)s - %(message)s',
)
def log_user_activity(username):
    # Get the current timestamp
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    # Get the user's IP address
    ip_address = socket.gethostbyname(socket.gethostname())
    # Log the activity with anonymized data
    logging.info(f'User: {username}, IP: {ip_address}, Time: {timestamp}')
# Example usage
username = input("Enter your username: ")
log_user_activity(username)
print("User activity logged successfully.")
```

Output:-

```
Enter your username: manoj
User activity logged successfully.
```

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.