SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

Name of Student : Kesoju saikiran
Enrollment No. : 2303A51539
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**

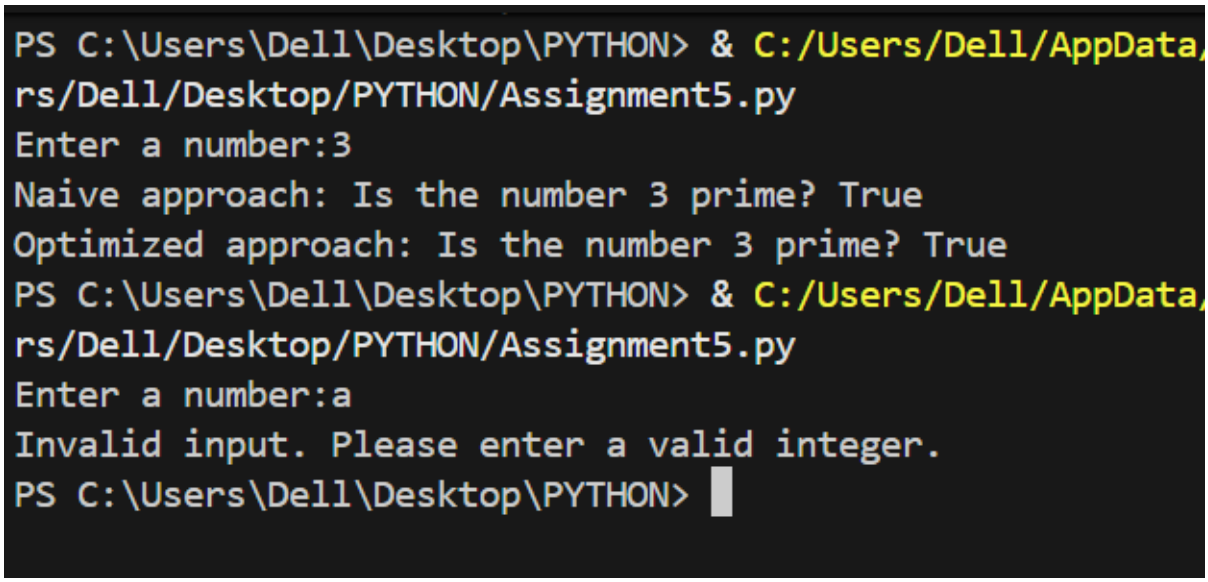
Prompt:-
write a python program of a prime number checking in two methods navie and optimised approach. and add when in input letters it gives invalid input

Code:

```
#Task 1
#write a python program for two prime checking methods: one is Naive approach and optimized approach.

# Add when given a input in letters the output should be invalid input and give negative number outpu
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
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
#Example usage:
number = input("Enter a number:")
try:
    number = int(number)
    result_naive = is_prime_naive(number)
    result_optimized = is_prime_optimized(number)
    print(f'Naive approach: Is the number {number} prime? {result_naive}')
    print(f'Optimized approach: Is the number {number} prime? {result_optimized}')
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Output:-


PS C:\Users\Dell\Desktop\PYTHON> & C:/Users/Dell/AppData/
rs/Dell/Desktop/PYTHON/Assignment5.py
Enter a number:3
Naive approach: Is the number 3 prime? True
Optimized approach: Is the number 3 prime? True
PS C:\Users\Dell\Desktop\PYTHON> & C:/Users/Dell/AppData/
rs/Dell/Desktop/PYTHON/Assignment5.py
Enter a number:a
Invalid input. Please enter a valid integer.
PS C:\Users\Dell\Desktop\PYTHON>

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 generate a recursive function to calculate a fibonacci numbers.
def fibonacci(n):
    if n <= 0:
        return "Invalid input. Please enter a positive integer."
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
#Example usage:
num_terms = input("Enter the number of terms in the Fibonacci sequence:")
try:
    num_terms = int(num_terms)
    print(f'Fibonacci sequence with {num_terms} terms:')
    for i in range(1, num_terms + 1):
        print(fibonacci(i), end=' ')
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Output:-

```
rs/Dell/Desktop/PYTHON/Assignment5.py
Enter the number of terms in the Fibonacci sequence:5
Enter the number of terms in the Fibonacci sequence:5
Fibonacci sequence with 5 terms:
Fibonacci sequence with 5 terms:
0 1 1 2 3
PS C:\Users\Dell\Desktop\PYTHON>
```

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:-

```
#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 that builds a simple login system, takes user input with prompts, and securely handles passwords using hashing and salt without storing plain-text passwords.

.

Code:-

```python
#Write a Python program that builds a simple login system, takes user input with prompts, and
import hashlib
import os
def hash_password(password, salt):
    return hashlib.sha256(salt + password.encode()).hexdigest()
def create_user(username, password):
    salt = os.urandom(16)
    hashed_password = hash_password(password, salt)
    return {'username': username, 'salt': salt, 'hashed_password': hashed_password}
def verify_user(user, password):
    hashed_password = hash_password(password, user['salt'])
    return hashed_password == user['hashed_password']
#Example usage:
username = input("Enter username: ")
password = input("Enter password: ")
user = create_user(username, password)
print("User created successfully!")
login_username = input("Enter username to login: ")
login_password = input("Enter password to login: ")
if login_username == user['username'] and verify_user(user, login_password):
    print("Login successful!")
else:
    print("Login failed! Invalid username or password.")
```

Output:-

```
Enter username: sai
Enter password: 5678
User created successfully!
Enter username to login:
```

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.

SR UNIVERSITY

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 that logs user activities (username, timestamp, IP address), identifies insecure or unnecessary sensitive logging, and then provides an improved version with minimal, masked, or anonymized secure logging.

Code:-

```
#Write a Python program that logs user activities (username, timestamp, IP address), identifies insec
import logging
from datetime import datetime
import socket
# Configure logging
logging.basicConfig(filename='user_activity.log', level=logging.INFO, format='%(asctime)s - %(message
def log_user_activity(username):
    try:
        ip_address = socket.gethostbyname(socket.gethostname())
    except socket.gaierror:
        ip_address = 'Unknown'
    masked_ip = mask_ip(ip_address)
    logging.info(f'User: {username}, IP: {masked_ip}')
def mask_ip(ip):
    parts = ip.split('.')
    if len(parts) == 4:
        return f'{parts[0]}.{parts[1]}.***.***'
    return '***.***.***.***'
#Example usage:
username = input("Enter username: ")
log_user_activity(username)
print("User activity logged securely.")
```

Output:-

```
rs/Dell/Desktop/PYTHON/Assignment4.py
Enter username: sai
User activity logged securely.
PS C:\Users\Dell\Desktop\PYTHON>
```

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.