
Lab Assignment # 5

Name of Student : T.SAITEJA
Enrollment No. : 2303A51535
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 code for two prime-checking methods and explain how the optimized version improves performance.

Code:-

```
"""Generate Python code for two prime-checking methods and
#explain how the optimized version improves performance.''
def is_prime_basic(n):
    """Check if a number is prime using basic method."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """Check if a number is prime using optimized method."""
    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
# Example usage:
if __name__ == "__main__":
    test_numbers = [29, 15, 97, 100, 1, 2, 3, 4, 25]
    for number in test_numbers:
        print(f"Basic check: Is {number} prime? {is_prime_basic(number)}")
        print(f"Optimized check: Is {number} prime? {is_prime_optimized(number)}")
```

Output:-

```

Basic check: Is 29 prime? True
Optimized check: Is 29 prime? True
Basic check: Is 15 prime? False
Optimized check: Is 15 prime? False
Basic check: Is 97 prime? True
Optimized check: Is 97 prime? True
Basic check: Is 100 prime? False
Optimized check: Is 100 prime? False
Basic check: Is 1 prime? False
Optimized check: Is 1 prime? False
Basic check: Is 2 prime? True
Optimized check: Is 2 prime? True
Optimized check: Is 1 prime? False
Basic check: Is 2 prime? True
Optimized check: Is 2 prime? True
Basic check: Is 2 prime? True
Optimized check: Is 2 prime? True
Optimized check: Is 2 prime? True
Basic check: Is 3 prime? True
Optimized check: Is 3 prime? True
Optimized check: Is 3 prime? True
Basic check: Is 4 prime? False
Basic check: Is 4 prime? False
Optimized check: Is 4 prime? False
Optimized check: Is 4 prime? False
Basic check: Is 25 prime? False
Optimized check: Is 25 prime? False

```

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 $6k \pm 1$ 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:-

```

31
32     # Task-2
33     #write a python program to print to calculate fibonacci series using recursion
34     def fibonacci(n):
35         # Base case: the first two Fibonacci numbers are 0 and 1
36         if n <= 0:
37             return 0
38         elif n == 1:
39             return 1
40         else:
41             # Recursive call: sum of the two preceding Fibonacci numbers
42             return fibonacci(n - 1) + fibonacci(n - 2)
43     num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
44     print("Fibonacci series:")
45     for i in range(num_terms):
46         print(fibonacci(i), end=" ")
47     print()  # for a new line after the series
48

```

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:-

```
50  #Task-3
51  # write a python program to read file and processes data using error handling and clear e
52  try:
53      file_name = input("Enter the file name to read: ")
54      with open(file_name, 'r') as file:
55          data = file.read()
56          print("File content:")
57          print(data)
58    except FileNotFoundError:
59        print("Error: The file was not found. Please check the file name and try again.")
60    except IOError:
61        print("Error: An I/O error occurred while trying to read the file.")
62    except Exception as e:
63        print(f"An unexpected error occurred: {e}")
64
```

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:

FileNotFoundException: 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 system and secure password handling practices.

Code:

```
# Task-4
#write a python program to build a login system 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 = {
    "Saiteja": hash_password("Sai@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: Saiteja
Enter your password: Sai@123
Login successful!
```

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).

Analyse: 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: -

```
90
91 # Task-5
92 #write a python program to script the logs user activities Example: username,
93 import logging
94 from datetime import datetime
95 import socket
96 # Configure logging
97 logging.basicConfig(
98     filename='user_activities.log',
99     level=logging.INFO,
100    format='%(asctime)s - %(message)s',
101 )
102 def log_user_activity(username):
103     # Get the current timestamp
104     timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
105     # Get the user's IP address
106     ip_address = socket.gethostname()
107     # Log the activity with anonymized data
108     logging.info(f'User: {username}, IP: {ip_address}, Time: {timestamp}')
109 # Example usage
110 username = input("Enter your username: ")
111 log_user_activity(username)
112 print("User activity logged successfully.")
113
```

Output:-

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

Justification:-

The code handles various exceptions that might occur during file reading:

FileNotFoundException: 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.