

**Lab Assignment # 5**

**Name of Student : Gunturu Devi Prasad**

**Enrollment No. : 2303A51538**

**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

```
Assignments Codes > Assignments5.py > ...
1  # Task-1
2  #write a python program of a prime number checking in two methods naive and optimised approach.
3  # and add when in input letters it gives invalid input
4  def is_prime_naive(n):
5      if n <= 1:
6          return False
7      for i in range(2, n):
8          if n % i == 0:
9              return False
10     return True
11 def is_prime_optimized(n):
12     if n <= 1:
13         return False
14     if n <= 3:
15         return True
16     if n % 2 == 0 or n % 3 == 0:
17         return False
18     i = 5
19     while i * i <= n:
20         if n % i == 0 or n % (i + 2) == 0:
21             return False
22         i += 6
23     return True
24 try:
25     number = int(input("Enter a number to check if it is prime: "))
26     print("Naive approach:", is_prime_naive(number))
27     print("Optimized approach:", is_prime_optimized(number))
28 except ValueError:
29     print("Invalid input. Please enter a valid integer.")
30
```

**Code:Output:-**

```
1570 WIN32_X64_Banded(1103.debugpy_launcher) 05002 C:\Users\9191...
```

```
p\Ai Assistant coding\Assignment5.5.py'
Enter a number to check if it is prime: 21
Naive approach: False
Optimized approach: 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: 7
Fibonacci series:
0 1 1 2 3 5 8
```

**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:-**

```

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.")

```

**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 = {
    "Deviprasad": hash_password("Devi@2390"),
    "Chinnu": hash_password("Mynameischinnu@123"),
}
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: Deviprasad
Enter your password: Devi@2390
Login successful!
Enter your username: Chinnu
User activity logged successfully.
PS C:\Users\Devi Prasad Gunturu\Desktop\Ai Assistant coding> █
```

**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:-**

```
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: Chinnu
User activity logged successfully.
PS C:\Users\Devi Prasad Gunturu\Desktop\Ai Assistant coding>
```

**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.