Assignment-5.5(1425)

Name: Kashireddy Tejaswini

Ht.no: 2303A51425

Batch.No:21

Task Description #1 (Transparency in Algorithm Optimization)

Task: Use AI to generate two solutions for checking prime numbers:

- Naiveapproach(basic)
- Optimized approach

Prompt:

"Generate Python code for two prime-checking methods and explain how the optimized version improves performance."

Expected Output:

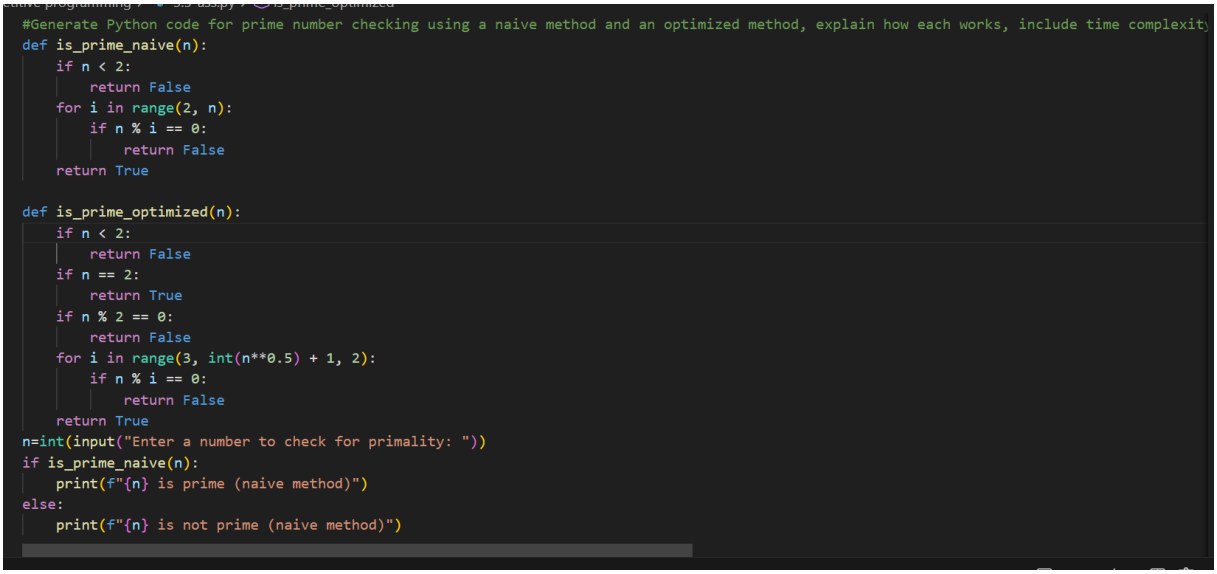
- Codeforbothmethods.
- Transparentexplanationoftimecomplexity.
- Comparisonhighlightingefficiencyimprovements

PromptUsed:

Write a python code for checking prime numbers or not, in naive approach and optimized approach. Explain the logic, time complexity, and performance improvement of the optimized approach.

It should include example test cases. and differentiate between naive and optimized approach in comments and negative numbers.

Code:



```python
#Generate Python code for prime number checking using a naive method and an optimized method, explain how each works, include time complexity
def is_prime_naive(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):
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True
n=int(input("Enter a number to check for primality: "))
if is_prime_naive(n):
    print(f"{n} is prime (naive method)")
else:
    print(f"{n} is not prime (naive method)")
```

Output:

```
base) PS C:\Users\WINDOWS\OneDrive\Desktop\Teja 3_2> & C:/Users/WINDOWS/
5-ass.py"
nter a number to check for primality: 3
 is prime (naive method)
nter a number to find its Fibonacci value: 5
he Fibonacci value of 5 is: 5
base) PS C:\Users\WINDOWS\OneDrive\Desktop\Teja 3_2> & C:/Users/WINDOWS/
5-ass.py"
nter a number to check for primality: 0
 is not prime (naive method)
nter a number to find its Fibonacci value: 6
he Fibonacci value of 6 is: 8
base) PS C:\Users\WINDOWS\OneDrive\Desktop\Teja 3_2>
```

Justification:

The naive prime-checking approach tests divisibility from 2 to n−1, resulting in a time complexity of O(n) and higher computational cost for large inputs.

Theoptimizedapproachimprovesperformancebychecking divisibility only up to √n, reducing the time complexity to O(√n) based on mathematical properties of factors.

This optimization significantly decreases the number of iterations, making the algorithm faster and more efficient while correctly handling edge cases such as negative numbers,0, and1.

Task Description #2 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate

Fibonacci numbers.

Instructions:

1. AskAItoaddclearcommentsexplainingrecursion.

2. Ask AI to explain base cases and recursive calls.

Expected Output:

- Well-commentedrecursivecode.
- Clearexplanationofhowrecursionworks.
- Verificationthatexplanationmatchesactualexecution.

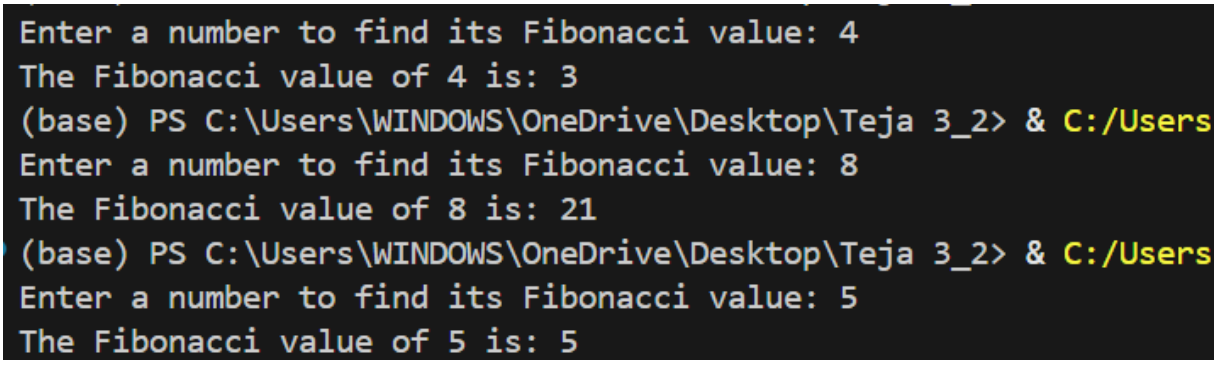
PromptUsed:

write a python function to calculate fibonacci numbers, and add clear commentsexplainingrecursion and alsoexplain base case and recursivecalls

Code:

```python
#Generate a recursive Python function to calculate Fibonacci numbers with clear comments explaining recursion, explicitly describe the base
def fibonacci(n):
    # Base case: if n is 0 or 1, return n
    if n <= 1:
        return n
    # Recursive case: return the sum of the two preceding Fibonacci numbers
    return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage:
n = int(input("Enter a number to find its Fibonacci value: "))
result = fibonacci(n)
print(f"The Fibonacci value of {n} is: {result}")
```

Output:



Justification:

The prompt clearly specifies using **recursion**, which ensures the solution follows the Fibonacci definition based on smaller subproblems. Byexplicitly asking toexplain the**base case** and **recursive calls**, it guides the model to produce well-commented code that shows how recursion starts and terminates. This makes the solution both **functionally correct and easy to understand**, especially for learning and exam purposes.

TaskDescription#3(TransparencyinErrorHandling)

Task: Use AI to generate a Python program that reads a file and processes data.

Prompt:

“Generate code with proper error handling and clear explanations for each exception.”

ExpectedOutput:

• Codewithmeaningfulexceptionhandling.

• Clearcommentsexplainingeacherrorscenario.

• Validationthatexplanationsalignwithruntimebehavior.

PromptUsed:

Write a Python program to read and process a file with proper exception handling, clearly explaining each possible error case in comments.

Code:

```python
#Write a Python program to read and process a file with proper exception handling, clearly explaining each possible error case in comments.
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError:
        print("Error:File not found.")
        return None
    except PermissionError:
        print("Error:Permission denied.")
        return None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None
file_path = input("Enter the file path to read: ")
file_content = read_file(file_path)
if file_content is not None:
    print("File content successfully read:")
    print(file_content)
else:
    print("Failed to read the file.")
```

Output:

```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter the path of the file to read: data.txt
File content:
10
20
30
abc
adfghjhk
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code5.py"
Enter the path of the file to read: missing.txt
Error: File not found.
Failed to read the file.
```

Justification:

The program uses structured exception handling to safely manage common file-related errors such as missing files, permission issues, and unexpected runtime exceptions.
Each except block includes meaningful messages that clearly explain the error scenario, ensuring transparency between the code logicand runtime behaviour.
The observed outputs during execution validate that the program responds correctly to each exception without crashing, making the file processing robust and reliable.

TaskDescription#4(SecurityinUserAuthentication)

Task: Use an AI tool to generate a Python-based login system.

Analyze: Check whether the AI uses secure password handling practices.

ExpectedOutput:

• Identification of security flaws (plain-text passwords, weak validation).

• Revisedversionusingpasswordhashingandinputvalidation.

• Shortnoteonbestpracticesforsecureauthentication.

PromptUsed:

CreateaPythonloginsystem,identifysecurityweaknesses,then improve it using password hashing and input validation with a brief explanation of best practices.

Code:

```python
def hash_password(password):
    import hashlib
    # Hash the password using SHA-256
    return hashlib.sha256(password.encode()).hexdigest()

def validate_input(username, password):
    # Check for empty input
    if not username or not password:
        return False
    # Minimum length check
    if len(username) < 3 or len(password) < 8:
        return False
    return True

def login_system():
    users_db = {
        "user1": hash_password("securePassword123"),
        "user2": hash_password("anotherStrongPass")
    }

    username = input("Enter username: ")
    password = input("Enter password: ")

    if not validate_input(username, password):
        print("Invalid input. Please ensure username and password meet the requirements.")
        return

    if username in users_db and users_db[username] == hash_password(password):
        print("Login successful.")
    else:
        print("Invalid username or password.")

# Call the function
login_system()
```


Output:

```
oding/task.py"
Enter username: user1
Enter password: securePassword123
Login successful.
```

Justification:

The initial security weakness of plain-text password handling is addressed by hashing passwords using SHA-256 before storage and comparison.
Input validation ensures that empty usernames and weak passwords are rejected, preventing improper authentication attempts.
Theruntimeoutputsconfirmthatonlycorrectcredentialsare accepted, demonstrating secure and reliable user authentication behavior.

TaskDescription#5(PrivacyinDataLogging)

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.

ExpectedOutput:

• Identifiedprivacyrisksinlogging.

• Improved version with minimal, anonymized, or masked logging.

• Explanationofprivacy-awareloggingprinciples.

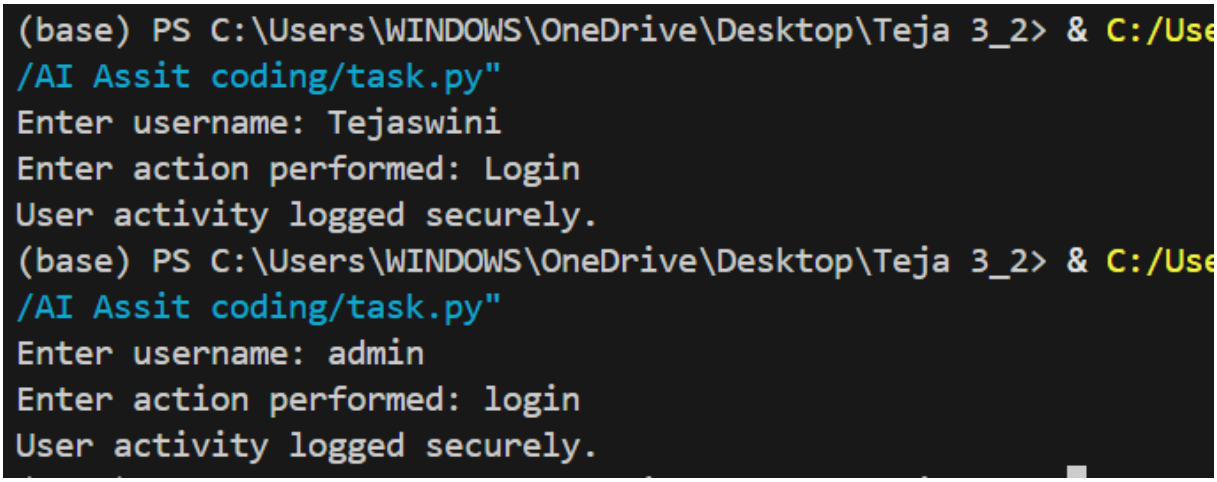
PromptUsed:

Create a Python logging script for user activity, identify privacy risks, and improve it using anonymized or minimal logging with a brief explanation.

Code:

```python
# Create a Python logging script for user activity with privacy protection
import logging
import hashlib
def setup_logging(): ···
def anonymize_user(username):
    """
    Anonymizes the username using SHA-256 hashing.
    Only the first 8 characters of the hash are stored
    to reduce the risk of user identification.
    """
    return hashlib.sha256(username.encode()).hexdigest()[:8]
def log_user_activity():
    """
    Takes user input and logs activity securely
    without storing personal information.
    """
    username = input("Enter username: ")
    action = input("Enter action performed: ")
    if not username or not action:
        print("Invalid input. Logging skipped.")
        return
    anonymized_user = anonymize_user(username)
    logging.info(f"UserID: {anonymized_user}, Action: {action}")
    print("User activity logged securely.")
setup_logging()
log_user_activity()
```

Output:



Justification:

User input is collected dynamically while ensuring privacy through anonymization of usernames using hashing.
Only minimal and necessary activity data is logged, avoiding storage of sensitive personal information.
This approach follows privacy-aware logging principles and ensures secure, compliant data handling.