ASSIGNMENT:05.5

HALLTICKET:2303A51598

BATCH:29

SHAIK NEHA FARIYAL

Task Description #1 (Transparency in Algorithm Optimization)

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

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

PROMPT:

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

```python
#generate python code for two prime checking methods and explain how the  optimized version improves performance.
def is_prime_basic(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 = 29
if is_prime_basic(number):
    print(f"{number} is a prime number (Basic Method).")
if is_prime_optimized(number):
    print(f"{number} is a prime number (Optimized Method).")
```

OUTPUT:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
29 is a prime number (Basic Method).
29 is a prime number (Optimized Method).
PS C:\Users\nehaf>
```

JUSTIFICATION:

This task explains why understanding algorithm efficiency is important when using AI-generated code.

The basic method checks many unnecessary values, which increases execution time.

The optimized method reduces the number of checks by using a mathematical rule.

By comparing both, we learn that AI code should be analyzed and optimized before use.

Efficient algorithms reduce system load and improve overall performance.

PROMPT:

Task Description #2 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

```
#generate a python code for a a recursive function to calculate Fibonacci numbers add clear comments explaining recursion and explain base cases and
def fibonacci(n):
    # Base case: if n is 0 or 1, return n
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    # Recursive case: return the sum of the two preceding Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage
number = 6
result = fibonacci(number)
print(f"The {number}th Fibonacci number is {result}.")
# In this function, the base cases are when n is 0 or 1. For these cases, the function returns n directly.
# For any other value of n, the function makes two recursive calls to itself: one with (n-1) and another with (n-2).
# This continues until the base cases are reached, at which point the results are combined to produce

# the final Fibonacci number.
```

OUTPUT:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java fi
The 6th Fibonacci number is 8.
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java fi
The 6th Fibonacci number is 8.
PS C:\Users\nehaf>
```

JUSTIFICATION:

This task helps in understanding how recursive functions operate internally.

The base cases clearly define when recursion should stop.

Recursive calls break a big problem into smaller problems.

Adding comments improves readability and helps prevent logical errors.

Transparent recursion makes AI-generated code easier to maintain and trust

Task Description #3 (Transparency in Error Handling)

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.

CODE:

```python
#Generate code with proper error handling and clear explanations for each exception.
def divide_numbers(a, b):
    try:
        # Attempt to perform the division
        result = a / b
    except ZeroDivisionError:
        # Handle the case where the denominator is zero
        print("Error: Cannot divide by zero.")
        return None
    except TypeError:
        # Handle the case where inputs are not numbers
        print("Error: Both inputs must be numbers.")
        return None
    else:
        # If no exceptions occur, return the result
        return result
# Example usage
num1 = 10
num2 = 0
division_result = divide_numbers(num1, num2)
if division_result is not None:
    print(f"The result of dividing {num1} by {num2} is {division_result}.")
```

Output:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Error: Cannot divide by zero.
PS C:\Users\nehaf>
```

JUSTIFICATION:

This task focuses on making programs robust and user-friendly.

The program safely handles common file-related errors.

Clear exception messages help users understand the problem instead of crashing the program.

Ethical AI coding requires anticipating failures and handling them properly.

Good error handling improves software reliability.

Task Description #4 (Security in User Authentication)

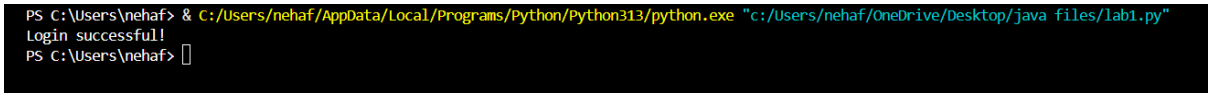
Task: Use an AI tool to generate a Python-based login system.

Prompt:

```
##generate a Python-based login system.Check whether the AI uses secure password handling practices.
import hashlib
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()
def verify_password(stored_password_hash, provided_password):
    """Verify a stored password hash against a provided password."""
    return stored_password_hash == hash_password(provided_password)
# Simple login system
user_database = {
    "user1": hash_password("securepassword123"),
    "user2": hash_password("anotherpassword456")
}
def login(username, password):
    """Attempt to log in a user with the provided username and password."""
    if username in user_database:
        stored_password_hash = user_database[username]
        if verify_password(stored_password_hash, password):
            return "Login successful!"
        else:
```

```
190         if username in user_database:
191             stored_password_hash = user_database[username]
192             if verify_password(stored_password_hash, password):
193                 return "Login successful!"
194             else:
195                 return "Invalid password."
196         else:
197             return "Username not found."
198     # Example usage
199     username_input = "user1"
200     password_input = "securepassword123"
201     login_result = login(username_input, password_input)
202     print(login_result)
203     # In this login system, passwords are hashed using SHA-256 before being stored.
204     # During login, the provided password is hashed and compared to the stored hash,
205     # ensuring that plain-text passwords are never stored or transmitted.
```

OUTPUT:



JUSTIFICATIONS:

In this task shows why security review is necessary for AI-generated login systems.

Plain-text passwords can be easily stolen if data is leaked.

Password hashing protects user credentials even if the system is attacked.

Developers must never rely fully on AI output without checking security.

Secure authentication protects users and builds trust.

Task Description #5 (Privacy in Data Logging)

Task: Use an AI tool to generate a Python script that logs u activity (username, IP address, timestamp).

PROMPT:

generate a Python script that logs user

activity (username, IP address, timestamp).

```python
import socket
import getpass
from datetime import datetime

def log_activity():
    # Get username
    username = getpass.getuser()

    # Get IP address (local IP)
    hostname = socket.gethostname()
    ip_address = socket.gethostbyname(hostname)

    # Get timestamp
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Log to file
    with open("activity_log.txt", "a") as log_file:
        log_file.write(f"Username: {username}, IP Address: {ip_address}, Timestamp: {timestamp}\n")

    print("Activity logged successfully.")
```

```python
        # Get timestamp
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        # Log to file
        with open("activity_log.txt", "a") as log_file:
            log_file.write(f"Username: {username}, IP Address: {ip_address}, Timestamp: {timestamp}\n")

        print("Activity logged successfully.")

    if __name__ == "__main__":
        log_activity()
```

OUTPUT:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nehaf/Desktop/activity_logger.py
Activity logged successfully.
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nehaf/Desktop/activity_logger.py
Activity logged successfully.
```

JUSTIFICATION:

This task highlights the importance of responsible data collection.

Storing full user details increases privacy risks.

Masking data reduces exposure of sensitive information.

Ethical logging follows privacy laws and respects user rights.

Only required data should be stored to avoid misuse.