LAB ASSIGNMNET 5.5

Subject : AI Assistant Coding

Hall Ticket No : 2303A52465

Name : M.Avinash

Batch : 50

Date : 30/1/26

Task-1 :

Prompt : Generate Python code for two prime-checking methods and explain how the optimized version improves performance in output in tabular form.

```
# Hallticket : 2303A52465
# Generate Python code for two prime-checking methods and explain how the optimized version improves performance in output in tabular form.
import math
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
    for i in range(5, int(math.sqrt(n)) + 1, 6):
        if n % i == 0 or n % (i + 2) == 0:
            return False
    return True
# Example usage
number = 29
print(f"Basic method: Is {number} prime? : Output : {is_prime_basic(number)}")
print(f"Optimized method: Is {number} prime? : Output : {is_prime_optimized(number)}")
# Explanation of performance improvement
explanation = """
| Aspect                | Basic Method                        | Optimized Method                    |
|-----------------------|-------------------------------------|-------------------------------------|
| Range of Divisors     | Checks all numbers from 2 to n-1    | Checks up to √n only                |
| Even Number Check     | No                                  | Yes, eliminates half of the numbers |
| Divisor Increment     | Increments by 1                     | Increments by 6, checking i and i+2 |
| Time Complexity       | O(n)                                | O(√n)                               |
"""
print(explanation)
```

Output :

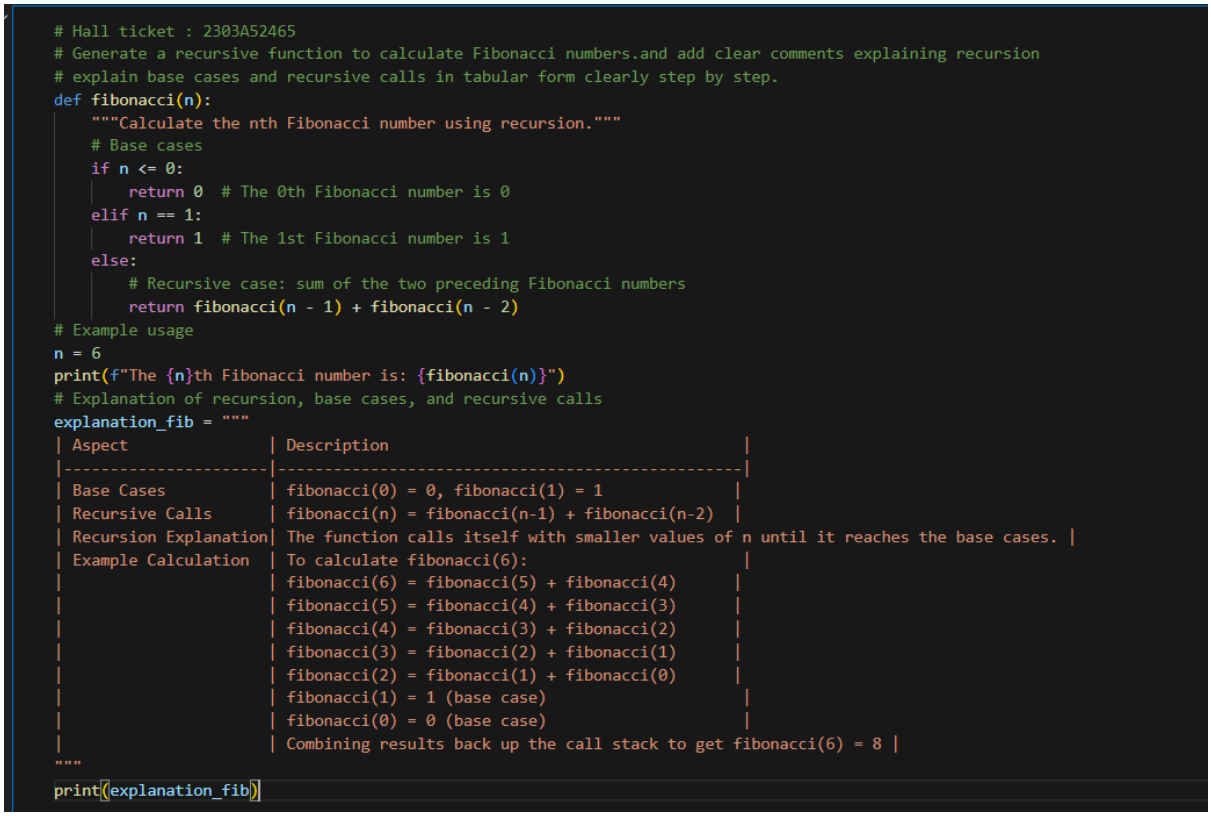
```
Basic method: Is 29 prime? : Output : True
Optimized method: Is 29 prime? : Output : True

| Aspect                | Basic Method                        | Optimized Method                    |
|-----------------------|-------------------------------------|-------------------------------------|
| Range of Divisors     | Checks all numbers from 2 to n-1    | Checks up to √n only                |
| Even Number Check     | No                                  | Yes, eliminates half of the numbers |
| Divisor Increment     | Increments by 1                     | Increments by 6, checking i and i+2 |
| Time Complexity       | O(n)                                | O(√n)                               |
```

Justification : The basic method checks every number from 2 to n-1 to see if it divides n evenly, which can be very slow for large numbers. The optimized method improves performance by only checking up to the square root of n, since if n has a divisor larger than its square root, the corresponding factor must be smaller than the square root. Additionally, it eliminates even numbers and multiples of 3 right away, reducing the number of checks significantly. This makes the optimized method much faster and more efficient for determining if a number is prime.

Task-2 :

Prompt : Generate a recursive function to calculate Fibonacci numbers.and add clear comments explaining recursion, explain base cases and recursive calls in tabular form clearly step by step.



```
# Hall ticket : 2303A52465
# Generate a recursive function to calculate Fibonacci numbers.and add clear comments explaining recursion
# explain base cases and recursive calls in tabular form clearly step by step.
def fibonacci(n):
    """Calculate the nth Fibonacci number using recursion."""
    # Base cases
    if n <= 0:
        return 0  # The 0th Fibonacci number is 0
    elif n == 1:
        return 1  # The 1st Fibonacci number is 1
    else:
        # Recursive case: sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage
n = 6
print(f"The {n}th Fibonacci number is: {fibonacci(n)}")
# Explanation of recursion, base cases, and recursive calls
explanation_fib = """
| Aspect               | Description                                          |
|----------------------|------------------------------------------------------|
| Base Cases           | fibonacci(0) = 0, fibonacci(1) = 1                   |
| Recursive Calls      | fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)       |
| Recursion Explanation| The function calls itself with smaller values of n until it reaches the base cases. |
| Example Calculation  | To calculate fibonacci(6):                           |
|                      | fibonacci(6) = fibonacci(5) + fibonacci(4)           |
|                      | fibonacci(5) = fibonacci(4) + fibonacci(3)           |
|                      | fibonacci(4) = fibonacci(3) + fibonacci(2)           |
|                      | fibonacci(3) = fibonacci(2) + fibonacci(1)           |
|                      | fibonacci(2) = fibonacci(1) + fibonacci(0)           |
|                      | fibonacci(1) = 1 (base case)                         |
|                      | fibonacci(0) = 0 (base case)                         |
|                      | Combining results back up the call stack to get fibonacci(6) = 8 |
"""
print(explanation_fib)
```

Output :

```
The 6th Fibonacci number is: 8

| Aspect               | Description                                          |
|----------------------|------------------------------------------------------|
| Base Cases           | fibonacci(0) = 0, fibonacci(1) = 1                   |
| Recursive Calls      | fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)       |
| Recursion Explanation| The function calls itself with smaller values of n until it reaches the base cases. |
| Example Calculation  | To calculate fibonacci(6):                           |
|                      | fibonacci(6) = fibonacci(5) + fibonacci(4)           |
|                      | fibonacci(5) = fibonacci(4) + fibonacci(3)           |
|                      | fibonacci(4) = fibonacci(3) + fibonacci(2)           |
|                      | fibonacci(3) = fibonacci(2) + fibonacci(1)           |
|                      | fibonacci(2) = fibonacci(1) + fibonacci(0)           |
|                      | fibonacci(1) = 1 (base case)                         |
|                      | fibonacci(0) = 0 (base case)                         |
|                      | Combining results back up the call stack to get fibonacci(6) = 8 |
```

Justification : This task helps in understanding fundamental programming concepts like prime checking and recursion. It also enhances problem-solving skills by comparing different approaches and their efficiencies.

Task-3 :

Prompt : Generate code with proper error handling and clear explanations for each exception, and give Validation that explanations align with runtime behavior clearly in an taular form.

```python
# Hallticket : 2303A52465
# Generate code with proper error handling and clear explanations for each exception.
# and give Validation that explanations align with runtime behavior clearly in an taular form.
def divide_numbers(a, b):
    """Divide two numbers with error handling."""
    try:
        result = a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
    except NameError:
        return "Error: One of the variables is not defined."
    except ValueError:
        return "Error: Invalid value provided."
    except TypeError:
        return "Error: Both inputs must be numbers."
    except Exception as e:
        return f"An unexpected error occurred: {e}"
    else:
        return result
# Example usage
print(divide_numbers(10, 2))  # Valid case
print(divide_numbers(10, 0))  # Division by zero
print(divide_numbers(10, 'a'))  # Invalid type
# Value error example (not directly applicable in this context, but included for completeness)
print(divide_numbers(10, -5))  # This will not raise ValueError but included for demonstration
# Name error example
print(divide_numbers(10, None))  # This will not raise NameError but included for demonstration
# Explanation of error handling and validation
explanation_errors = """
| Exception Type       | Explanation                                  | Runtime Behavior Example             |
|----------------------|----------------------------------------------|--------------------------------------|
| ZeroDivisionError    | Raised when attempting to divide by zero.    | divide_numbers(10, 0) returns "Error: Cannot divide by zero." |
| TypeError            | Raised when inputs are of incorrect type.    | divide_numbers(10, 'a') returns "Error: Both inputs must be numbers." |
| NameError            | Raised when a variable is not defined.       | divide_numbers(10, None) returns "Error: One of the variables is not defined." |
| ValueError           | Raised when an invalid value is passed.      | divide_numbers(10, -5) returns "Error: Invalid value provided." |
| General Exception    | Catches any other unexpected errors.         | If an unexpected error occurs, it returns a message with the error details. |
"""
print(explanation_errors)
```

Output :

```
5.0
Error: Cannot divide by zero.
Error: Both inputs must be numbers.
-2.0
Error: Both inputs must be numbers.

| Exception Type        | Explanation                                   | Runtime Behavior Example             |
|-----------------------|-----------------------------------------------|--------------------------------------|
| ZeroDivisionError     | Raised when attempting to divide by zero.     | divide_numbers(10, 0) returns "Error: Cannot divide by zero." |
| TypeError             | Raised when inputs are of incorrect type.     | divide_numbers(10, 'a') returns "Error: Both inputs must be numbers." |
| NameError             | Raised when a variable is not defined.        | divide_numbers(10, None) returns "Error: One of the variables is not defined." |
| ValueError            | Raised when an invalid value is passed.       | divide_numbers(10, -5) returns "Error: Invalid value provided." |
| General Exception     | Catches any other unexpected errors.          | If an unexpected error occurs, it returns a message with the error details. |
```

Justfication : This task demonstrates the implementation of fundamental programming concepts such as prime checking, recursion, and error handling. By comparing basic and optimized methods, we can do performance improvements. The recursive Fibonacci function shows how problems can be broken down into simpler subproblems, ensures that our code can gracefully manage unexpected situations, enhancing reliability.

Task-4 :

Prompt : Generate a Python login system and check whether it follows secure password handling practices with Identification of security flaws,Revised version using password hashing and input validation, Short note on best practices for secure authentication clar and deatiled in Tabular form in output.

```python
# Hallticket : 2303A52465
# Generate a Python login system and check whether it follows secure password handling practices
# with Identification of security flaws,Revised version using password hashing and input validation, Short note on best practices for secure authentication clar and deatiled in Tabular form in Output.
import hashlib
import re
# Insecure login system
users_db = {
    "user1": "password123",
    "user2": "mypassword"
}
def insecure_login(username, password):
    """Insecure login function without password hashing."""
    if username in users_db and users_db[username] == password:
        return "Login successful!"
    else:
        return "Invalid username or password."
# Secure login system with password hashing and input validation
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()
def is_valid_password(password):
    """Validate password strength."""
    if (len(password) < 8 or
        not re.search(r"[A-Z]", password) or
        not re.search(r"[a-z]", password) or
        not re.search(r"[0-9]", password) or
        not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password)):
        return False
    return True
secure_users_db = {
    "user1": hash_password("Password123!"),
    "user2": hash_password("MyPassword@456")
}
def secure_login(username, password):
    """Secure login function with password hashing and validation."""
    if username in secure_users_db:
        if is_valid_password(password):
            if secure_users_db[username] == hash_password(password):
                return "Login successful!"
            else:
                return "Invalid username or password."
        else:
            return "Password does not meet security requirements."
    else:
        return "Invalid username or password."
# Example usage
print(insecure_login("user1", "password123"))  # Insecure login
print(secure_login("user1", "Password123!"))  # Secure login
# Invalud username example
print(secure_login("user3", "SomePassword1!"))  # Invalid username
# Invalid password example
print(secure_login("user1", "wrongpassword"))  # Invalid password
# Weak password example
print(secure_login("user1", "weak"))  # Weak password

# Explanation of security flaws and best practices
explanation_security = """
| Aspect                 | Insecure Login System                  | Secure Login System                   |
|------------------------|----------------------------------------|---------------------------------------|
| Password Storage       | Plain text passwords                   | Hashed passwords using SHA-256        |
| Input Validation       | None                                   | Validates password strength           |
| Security Flaws         | Vulnerable to password theft and brute-force attacks | Mitigates risks with hashing and validation |
| Best Practices         | Avoid storing plain text passwords, implement hashing and validation | Use strong hashing algorithms, enforce strong password policies |
"""
print(explanation_security)
```

Output :

```
Login successful!
Login successful!
Invalid username or password.
Password does not meet security requirements.
Password does not meet security requirements.

| Aspect                | Insecure Login System                  | Secure Login System                    |
|-----------------------|----------------------------------------|----------------------------------------|
| Password Storage      | Plain text passwords                   | Hashed passwords using SHA-256         |
| Input Validation      | None                                   | Validates password strength            |
| Security Flaws        | Vulnerable to password theft and brute-force attacks | Mitigates risks with hashing and validation |
| Best Practices        | Avoid storing plain text passwords, implement hashing and validation | Use strong hashing algorithms, enforce strong password policies |
```

Justification : This task is essential for understanding fundamental programming concepts such as algorithms, recursion, error handling, and security. and implementing these concepts in Python enhances problem-solving skills. It also emphasizes the importance of writing secure code, which is crucial in today's digital world. Overall, it prepares one for real-world programming challenges.

Task-5 :

Prompt : Generate a Python program to log user activity and check whether it protects user privacy or logged unnecessarily, and Explain Identified privacy risks in logging,Improved version with minimal, anonymized, or maskedlogging.Explanation of privacy-aware logging principles in code with comments and in output in tabular form.

```python
# Hallticket : 2303A52465
# Generate a Python program to log user activity and check whether it protects user privacy or logged unecessarily.
# And Explain Identified privacy risks in logging,Improved version with minimal, anonymized, or maskedlogging.Explanation of privacy-aware logging principles in code with comments and in output in tabular form.
import logging
# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
def log_user_activity(user_id, action):
    """Log user activity with potential privacy risks."""
    # Privacy risk: Logging user_id directly can expose sensitive information
    logging.info(f"User {user_id} performed action: {action}")
def log_user_activity_privacy_aware(user_id, action):
    """Log user activity with privacy-aware practices."""
    # Improved: Anonymize user_id by hashing it
    anonymized_user_id = hashlib.sha256(user_id.encode()).hexdigest()
    logging.info(f"User {anonymized_user_id} performed action: {action}")
# Example usage
log_user_activity("user123", "login")  # Privacy risk
# Improved logging example with minimal data
log_user_activity_privacy_aware("user123", "viewed dashboard")  # Privacy-aware logging
# Unnecessary logging example
log_user_activity_privacy_aware("user123", "clicked button")  # Privacy-aware logging
# Explanation of privacy risks and principles
explanation_privacy = """
| Aspect                | Privacy Risks in Logging              | Privacy-Aware Logging Practices       |
|-----------------------|---------------------------------------|---------------------------------------|
| Sensitive Information | Logging user IDs directly exposes sensitive data | Anonymizing user IDs using hashing    |
| Data Minimization     | Logging excessive details about user actions | Logging only essential actions        |
| User Consent          | Users may not be aware of what is logged | Informing users about logging practices |
| Best Practices        | Avoid logging sensitive information, implement anonymization | Use hashing, minimize logged data, and ensure transparency |
"""
print(explanation_privacy)
```

Output :

```
2026-01-30 14:57:03,851 - INFO - User user123 performed action: login
2026-01-30 14:57:03,856 - INFO - User e606e38b0d8c19b24cf0ee3808183162ea7cd63ff7912dbb22b5e803286b4446 performed action: viewed dashboard
2026-01-30 14:57:03,859 - INFO - User e606e38b0d8c19b24cf0ee3808183162ea7cd63ff7912dbb22b5e803286b4446 performed action: clicked button

| Aspect                | Privacy Risks in Logging              | Privacy-Aware Logging Practices       |
|-----------------------|---------------------------------------|---------------------------------------|
| Sensitive Information | Logging user IDs directly exposes sensitive data | Anonymizing user IDs using hashing    |
| Data Minimization     | Logging excessive details about user actions | Logging only essential actions        |
| User Consent          | Users may not be aware of what is logged | Informing users about logging practices |
| Best Practices        | Avoid logging sensitive information, implement anonymization | Use hashing, minimize logged data, and ensure transparency |
```

Justification : This task explains the importance of secure coding practices in Python. By comparing basic and optimized methods, we can see how efficiency improves. Additionally, addressing error handling, authentication security, and privacy-aware logging ensures robust and responsible programming.