AI ASSISTANT CODING ASSESSMENT 1.4

T.VIKAS

2303A52281

TASK 1: Prime Number Check Without Functions.

Prompt : AI-Generated Logic Without Modularization for prime Number Check Without Functions.

Code:

```
num = int(input("Enter a number: "))

if num <= 1:
    print("Not a prime number")
else:
    is_prime = True
    for i in range(2, num):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print("Prime number")
    else:
        print("Not a prime number")
```

OUTPUT:

```
Enter a number: 2
Prime number
```

Report: In this task, a Python program was created to check whether a number is prime without using any user-defined functions. The entire logic was written inside the main body of the program. User input was taken using the input() function. A loop was used to check divisibility of the number. The output correctly displays whether the number is prime or not.

TASK 2: Logic Optimization (Cleanup)

Prompt: Generate Efficiency & Logic Optimization (Cleanup) prime number check.

Code:

```
import math

num = int(input("Enter a number: "))

if num <= 1:
    print("Not a prime number")
else:
    is_prime = True
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print("Optimized Prime number")
    else:
        print("Not a prime number")
```

OUTPUT:

```
Enter a number: 2
Optimized Prime number
```

Report:

This task focused on improving the efficiency of the prime number checking program. The loop range was optimized to run only up to the square root of the number. This reduced unnecessary iterations and improved performance. The code was also made more readable and cleaner. Time complexity was reduced compared to the previous version.

TASK3: Modular Design Using AI Assistance (Prime Number Check Using Functions)

Prompt: The prime-checking logic will be reused across multiple modules.

Code:

```python
def is_prime(num):
    if num <= 1:
        return False

    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True
number = int(input("Enter a number: "))

if is_prime(number):
    print("Prime number")
else:
    print("Not a prime number")
```

OUTPUT:

```
Enter a number: 6
Not a prime number
```

Report:

In this task, a modular approach was used by creating a user-defined function to check prime numbers. The function returns a Boolean value based on the result. This makes the code reusable and easier to maintain. Proper comments were added for better understanding. The program worked correctly for different test inputs.

TASK 4: Comparison Between Task 1 and Task 3

Prompt: Comparison table or short analytical report.

Code:

```python
#Task 4
print("\nComparison Between Task 1 and Task 3\n")

print("{:<20} {:<30} {:<30}".format("Feature", "Task 1 (Without Function)", "Task 3 (With Funct
print("-" * 80)

data = [
    ["Code Structure", "Single block of code", "Function-based structure"],
    ["Reusability", "No", "Yes"],
    ["Modularity", "No", "Yes"],
    ["Readability", "Medium", "High"],
    ["Ease of Debugging", "Difficult", "Easy"],
    ["Scalability", "Not Suitable", "Suitable"],
    ["Maintainability", "Low", "High"],
]

for row in data:
    print("{:<20} {:<30} {:<30}".format(row[0], row[1], row[2]))
```

OUTPUT:

```
Comparison Between Task 1 and Task 3

Feature             Task 1 (Without Function)     Task 3 (With Function)
--------------------------------------------------------------------------------
Code Structure      Single block of code          Function-based structure
Reusability         No                            Yes
Modularity          No                            Yes
Readability         Medium                        High
Ease of Debugging   Difficult                     Easy
Scalability         Not Suitable                  Suitable
Maintainability     Low                           High
Enter a number:
Feature             Task 1 (Without Function)     Task 3 (With Function)
--------------------------------------------------------------------------------
Code Structure      Single block of code          Function-based structure
Reusability         No                            Yes
Modularity          No                            Yes
Readability         Medium                        High
Feature             Task 1 (Without Function)     Task 3 (With Function)
--------------------------------------------------------------------------------
Feature             Task 1 (Without Function)     Task 3 (With Function)
Feature             Task 1 (Without Function)     Task 3 (With Function)
Feature             Task 1 (Without Function)     Task 3 (With Function)
--------------------------------------------------------------------------------
Code Structure      Single block of code          Function-based structure
Feature             Task 1 (Without Function)     Task 3 (With Function)
--------------------------------------------------------------------------------
Code Structure      Single block of code          Function-based structure
--------------------------------------------------------------------------------
Code Structure      Single block of code          Function-based structure
Code Structure      Single block of code          Function-based structure
Reusability         No                            Yes
Reusability         No                            Yes
Modularity          No                            Yes
Readability         Medium                        High
Ease of Debugging   Difficult                     Easy
Scalability         Not Suitable                  Suitable
Maintainability     Low                           High
Enter a number:
Readability         Medium                        High
Ease of Debugging   Difficult                     Easy
Scalability         Not Suitable                  Suitable
Readability         Medium                        High
Ease of Debugging   Difficult                     Easy
Readability         Medium                        High
Readability         Medium                        High
Ease of Debugging   Difficult                     Easy
Scalability         Not Suitable                  Suitable
Maintainability     Low                           High
```

Report:

This task involved comparing the programs from Task 1 and Task 3. The comparison was done based on clarity, reusability, debugging ease, and scalability. The function-based approach was found to be more structured. It is more suitable for large applications. A tabular comparison was generated using Python.

TASK 5: Your mentor wants to evaluate how AI handles alternative logical strategies.

Prompt: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking).

Code:

```python
#Task 5
num = int(input("Enter a number: "))

count = 0
for i in range(1, num + 1):
    if num % i == 0:
        count += 1

if count == 2:
    print("Prime number")
else:
    print("Not a prime number")
```

Output:

```
Enter a number: 1
Not a prime number
```

Report:

In this task, two different prime-checking approaches were implemented. The first method checks divisibility by all numbers, while the second checks up to the square root. The optimized method is faster and more efficient. Both implementations were tested with large inputs. Their performance and time complexity were compared.