AI FOR ASSISTED CODING

ASSIGNMENT – 13.1

Name: Mahesh Dasi

H.No: 2303A54021

Batch: 47_A

Task Description #1 (Refactoring – Removing Code Duplication)

Prompt:

```
You are a Python code refactoring assistant. Analyze the following legacy Python script and identify duplicated logic. Refactor the code to remove repetition by creating reusable functions. Ensure the output remains exactly the same. Add proper docstrings to all functions.
Legacy Code:
print("Area of Rectangle:", 5 * 10)
print("Perimeter of Rectangle:", 2 * (5 + 10))
print("Area of Rectangle:", 7 * 12)
print("Perimeter of Rectangle:", 2 * (7 + 12))
print("Area of Rectangle:", 10 * 15)
```

Code:

```python
def calculate_area(length, width):
    """
    Calculate the area of a rectangle.

    Args:
        length: The length of the rectangle.
        width: The width of the rectangle.

    Returns:
        The area of the rectangle.
    """
    return length * width


def calculate_perimeter(length, width):
    """
    Calculate the perimeter of a rectangle.

    Args:
        length: The length of the rectangle.
        width: The width of the rectangle.

    Returns:
        The perimeter of the rectangle.
    """
    return 2 * (length + width)


def print_rectangle_info(length, width):
    """
    Print the area and perimeter of a rectangle.

    Args:
        length: The length of the rectangle.
        width: The width of the rectangle.
```

```
    print("Area of Rectangle:", calculate_area(length, width))
    print("Perimeter of Rectangle:", calculate_perimeter(length, width))


# Process rectangles
print_rectangle_info(5, 10)
print_rectangle_info(7, 12)
print_rectangle_info(10, 15)
```

Output:

```
Area of Rectangle: 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
```

Task Description #2 (Refactoring – Optimizing Loops and Conditionals)

Prompt:

```
You are an AI code optimization assistant. Analyze the following Python script and identify inefficiencies,
especially nested loops and unnecessary condition checks. Suggest algorithmic improvements (such as using sets or
Python built-in functions) to improve performance. Keep the logic and output unchanged.
Compare time complexity before and after optimization. Provide a performance comparison table.
```

Code:

```
names = {"Alice", "Bob", "Charlie", "David"}
search_names = ["Charlie", "Eve", "Bob"]

for s in search_names:
    if s in names:
        print(f"{s} is in the list")
    else:
        print(f"{s} is not in the list")
```

Output:

```
Charlie is in the list
Eve is not in the list
Bob is in the list
```

Task Description #3 (Refactoring – Extracting Reusable Functions)

Prompt:

```
Act as a Python refactoring expert. Analyze the following script where calculations are embedded directly
inside the main code block. Identify repeated or related logic and extract it into reusable functions. Ensure the refactored version is modular, readable, and
includes proper docstrings. Do not change the output.
```

Code:

```python
def calculate_total_price(price, tax_rate=0.18):
    """
    Calculate the total price including tax.

    Args:
        price: The base price.
        tax_rate: The tax rate as a decimal (default: 0.18 for 18%

    Returns:
        The total price including tax.
    """
    tax = price * tax_rate
    return price + tax


# Process prices
print("Total Price:", calculate_total_price(250))
print("Total Price:", calculate_total_price(500))
```

Output:

```
Total Price: 295.0
Total Price: 590.0
```

Task Description #4 (Refactoring – Replacing Hardcoded Values with Constants)

Prompt:

```
You are a Python code improvement assistant. Analyze the following script and identify all hardcoded "magic numbers." Refactor the code by defining named constants at the top of the file. Replace all occurrences of hardcoded values with those constants. Ensure functionality remains unchanged and add appropriate comments.
```

Code:

```python
# Constants
PI = 3.14159
CIRCLE_RADIUS = 7

# Calculate and print circle area and circumference
print("Area of Circle:", PI * (CIRCLE_RADIUS ** 2))
print("Circumference of Circle:", 2 * PI * CIRCLE_RADIUS)
```

Output:

```
Area of Circle: 153.93791
Circumference of Circle: 43.98226
```

Task Description #5 (Refactoring – Improving Variable Naming and Readability)

Prompt:

```
Act as a Python clean code expert.
Refactor the following script to improve readability: Replace unclear variable names with meaningful names.  Add inline comments explaining calculations.
Keep the functionality exactly the same.
```

Code:

```
# Calculate triangle area with base and height
base = 10
height = 20
triangle_area = base * height / 2
print(triangle_area)
```

Output:

```
100.0
```

Task Description #6 (Refactoring – Removing Redundant Conditional Logic)

Prompt:

```
Refactor this code by extracting repeated grading logic into a reusable function.
Keep grading rules unchanged and add a proper docstring.
```

Code:

```
def calculate_grade(marks):
    """
    Calculate the grade based on marks.

    Args:
        marks: The marks obtained (0-100).

    Returns:
        The grade as a string ('A', 'B', or 'C').
    """
    if marks >= 90:
        return "Grade A"
    elif marks >= 75:
        return "Grade B"
    else:
        return "Grade C"
```

Output:

```
Grade B
Grade C
```

Task Description #7 (Refactoring – Converting Procedural Code to Functions)

Prompt:

```
Refactor this procedural code into separate functions for input, processing, and output.
Improve modularity and add docstrings without changing behavior.
```

Code:

```python
def get_number_input():
    """
    Get a number from user input.

    Returns:
        An integer entered by the user.
    """
    return int(input("Enter number: "))


def calculate_square(num):
    """
    Calculate the square of a number.

    Args:
        num: The number to square.

    Returns:
        The square of the number.
    """
    return num * num


def display_result(num, square):
    """
    Display the square result.

    Args:
        num: The original number.
        square: The squared value.
    """
    print("Square:", square)
```

```python
# Main execution
number = get_number_input()
result = calculate_square(number)
display_result(number, result)
```

Output:

```
Enter number: 25
Square: 625
```

Task Description #8 (Refactoring – Optimizing List Processing)

Prompt:

```
Refactor this list processing code using list comprehensions or Python built-in functions.
Keep the output identical and improve readability.
```

Code:

```python
nums = [1, 2, 3, 4, 5]
squares = [n * n for n in nums]
print(squares)
```

Output:

```
[1, 4, 9, 16, 25]
```