LAB ASSIGNMENT-13.1

Name: Akhil
Roll.No: 2303A53024
Batch: 46

Task-1

Analyze the following Python script and refactor it to remove duplicate logic. Create reusable functions, maintain the same output, and add proper docstrings.

CODE:

```
def calculate_rectangle(length: int, width: int) -> None:
    """
    Calculates and prints the area and perimeter of a rectangle.
    Args:
        length (int): Length of the rectangle
        width (int): Width of the rectangle
    """
    area = length * width
    perimeter = 2 * (length + width)
    print("Area of Rectangle:", area)
    print("Perimeter of Rectangle:", perimeter)
# Reusable function calls
calculate_rectangle(5, 10)
calculate_rectangle(7, 12)
calculate_rectangle(10, 15)
```

Task-2

Refactor this nested loop Python code using set lookups to improve performance. Maintain the same logic and compare execution time before and after optimization

CODE:

```
import time
names = ["Alice", "Bob", "Charlie", "David"]
search_names = ["Charlie", "Eve", "Bob"]
start = time.time()
for s in search_names:
    found = False
    for n in names:
        if s == n:
            found = True
    if found:
        print(f"{s} is in the list")
    else:
        print(f"{s} is not in the list")
original_time = time.time() - start
start = time.time()
names_set = set(names)
for s in search_names:
```

```
    if s in names_set:
        print(f"{s} is in the list")
    else:
        print(f"{s} is not in the list")
optimized_time = time.time() - start
print("\nPerformance Comparison")
print("----------------------")
print(f"Original Time: {original_time:.10f}")
print(f"Optimized Time: {optimized_time:.10f}")
```

Task-3

Refactor this script by extracting repeated tax calculation logic into a reusable function called calculate_total(price). Add proper docstrings.

CODE:

```
TAX_RATE = 0.18
def calculate_total(price: float) -> float:
    tax = price * TAX_RATE
    return price + tax
print("Total Price:", calculate_total(250))
print("Total Price:", calculate_total(500))
```

Task-4

Identify magic numbers in this script and replace them with named constants at the top of the file. Maintain same functionality.

CODE:

```
PI = 3.14159
RADIUS = 7
def calculate_circle_area(radius: float) -> float:
    """Returns area of a circle."""
    return PI * (radius ** 2)
def calculate_circumference(radius: float) -> float:
    """Returns circumference of a circle."""
    return 2 * PI * radius
print("Area of Circle:", calculate_circle_area(RADIUS))
print("Circumference of Circle:", calculate_circumference(RADIUS))
```

Task-5

Improve variable names and readability of this script without changing functionality. Add inline comments where necessary.

CODE:

```
base = 10
height = 20

area_of_triangle = base * height / 2
```

```
print("Area of Triangle:", area_of_triangle)
```

Task-6

Refactor repeated grading logic into a reusable function calculate_grade(marks) with clean conditional structure and docstring.

CODE:

```
def calculate_grade(marks: int) -> str:
    if marks >= 90:
        return "Grade A"
    elif marks >= 75:
        return "Grade B"
    else:
        return "Grade C"
print(calculate_grade(85))
print(calculate_grade(72))
```

Task-7

Refactor this procedural code into modular functions: get_input(), calculate_square(), and display_result(). Maintain behavior.

CODE:

```
def get_input() -> int:
```

```
    return int(input("Enter number: "))
def calculate_square(number: int) -> int:
    return number * number
def display_result(result: int) -> None:
    print("Square:", result)
num = get_input()
square = calculate_square(num)
display_result(square)
```