ASSIGNMENT - 10.2

2303A510G1
Batch : 30

Task Description -1(Error Detection and Correction)
Task:
Use AI to analyze a Python script and correct all syntax and logical errors.
Sample Input Code:

```
def calculate_total(nums)
sum = 0
for n in nums
sum += n
return total
```

Expected Output-1:
Corrected and executable Python code with brief explanations of the identified syntax and logic errors.

task1.py > ...

```python
def calculate_total(nums):
    # Initialize total
    total = 0

    # Add each number to total
    for n in nums:
        total += n

    return total


#  Taking user input
# User enters numbers separated by space (example: 10 20 30)
user_input = input("Enter numbers separated by space: ")

# Convert input string into list of integers
numbers = list(map(int, user_input.split()))

# Call function and print result
result = calculate_total(numbers)

print("Total sum is:", result)
```

Output:

```
arepallyramcharan@Arepallys-MacBook-Air assignment-10.2.py % /usr/bin/python3 "/Users/arepallyramcharan/ai assis
tent.py/assignment-10.2.py/task1.py"
Enter numbers separated by space: 23 344 44 33 33
Total sum is: 477
```

Observation:

1. The original code had **syntax errors** (missing colons after function and loop).
2. Indentation was missing, which is mandatory in Python.
3. There was a **logical error** due to variable mismatch (`sum` vs `total`).

4. The variable name `sum` overrides Python's built-in `sum()` function.

5. After correction, the code becomes executable, logically consistent, and follows proper Python structure.

Task Description -2(Code Style Standardization)
Task:
Use AI to refactor Python code to comply with standard coding style guidelines.
Sample Input Code:
def findSum(a,b):return a+b
print(findSum(5,10))
Expected Output-2:
Well-structured, consistently formatted Python code following standard style conventions.

task2.py > find_sum

```python
# Function to calculate the sum of two numbers
def find_sum(a: int, b: int) -> int:
    """
    Return the sum of two numbers.

    Args:
        a (int): First number
        b (int): Second number

    Returns:
        int: Sum of a and b
    """

    # Add the two numbers and return the result
    return a + b


# Main function to execute the program
def main() -> None:
    # Call the function with sample values
    result = find_sum(5, 10)

    # Print the result to the console
    print(result)


# Ensure the script runs only when executed directly
if __name__ == "__main__":
    main()
```

Output:

```
arepallyramcharan@Arepallys-MacBook-Air assignment-10.2.py % /usr/bin/python3 "/Users/arepallyramcharan/ai assis
tent.py/assignment-10.2.py/task2.py"
15
```

Observation:

1. The original function name `findSum` did not follow **PEP 8 snake_case convention**.

2. The one-line function reduced readability and maintainability.

3. Lack of spacing and structure made the code harder to read.

4. Refactoring improved formatting, indentation, and structure.

5. The updated version follows professional coding standards and improves maintainability.

Task Description -3(Code Clarity Improvement)
Task:
Use AI to improve code readability without changing its functionality.
Sample Input Code:

```
def f(x,y):
return x-y*2
print(f(10,3))
```

Expected Output-3:
Python code rewritten with meaningful function and variable names,
proper indentation, and improved clarity.

task3.py > calculate_adjusted_value

```python
def calculate_adjusted_value(number: int, multiplier: int) -> int:
    """
    Subtract twice the multiplier from the given number.

    Args:
        number (int): The original value.
        multiplier (int): The value to be doubled and subtracted.

    Returns:
        int: Result after subtracting (multiplier * 2) from number.
    """
    result = number - (multiplier * 2)
    return result


# Call the function with sample values
output = calculate_adjusted_value(10, 3)

# Display the result
print(output)
```

Output:

```
arepallyramcharan@Arepallys-MacBook-Air assignment-10.2.py % /usr/bin/python3 "/Users/arepallyramcharan/ai assis
tent.py/assignment-10.2.py/task3.py"
4
```

Observation:

1. The function name f was not descriptive.
2. Variable names x and y lacked clarity about their purpose.
3. The mathematical expression was not clearly structured.
4. Adding meaningful names improved understanding without changing functionality.
5. Proper indentation and documentation enhanced readability and code clarity.

Task Description -4(Structural Refactoring)
Task:
Use AI to refactor repetitive code into reusable functions.
Sample Input Code:
print("Hello Ram")
print("Hello Sita")
print("Hello Ravi")
Expected Output-4:
Modular Python code using reusable functions to eliminate repetition.

task4.py > ...

```python
# Function to print a greeting message
def greet_person(name: str) -> None:
    """
    Prints a greeting for the given person.

    Args:
        name (str): The name of the person to greet.
    """

    # Print greeting message using formatted string
    print(f"Hello {name}")


# Main function to organize program execution
def main() -> None:

    # List containing names of people to greet
    names = ["Ram", "Sita", "Ravi"]

    # Loop through each name in the list
    for person in names:

        # Call the reusable greeting function
        greet_person(person)


# Ensures the program runs only when executed directly
if __name__ == "__main__":
    main()
```

Output:

```
arepallyramcharan@Arepallys-MacBook-Air assignment-10.2.py % /usr/bin/python3 "/Users/arepallyramcharan/ai assis
tent.py/assignment-10.2.py/task4.py"
Hello Ram
Hello Sita
Hello Ravi
```

Observation:

1. The original code contained repetitive `print()` statements.
2. Repetition reduces scalability and maintainability.
3. Introducing a reusable function eliminated duplication.
4. Using a loop and list makes the program extensible.
5. Modular structure improves code organization and future modification.

Task Description -5(Efficiency Enhancement)
Task:
Use AI to optimize Python code for better performance.
Sample Input Code:
numbers = [ ]
for i in range(1, 500000):
numbers.append(i * i)
print(len(numbers))
Expected Output-5:
Optimized Python code that achieves the same result with improved performance.

task5.py > ...

```python
def generate_squares():
    """
    Generate squares of numbers from 1 to 499999
    and return the total count.
    """
    # List comprehension (faster than append loop)
    numbers = [i * i for i in range(1, 500000)]

    # Return the length of the list
    return len(numbers)


def main():
    count = generate_squares()
    print("Total numbers generated:", count)


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

Output:

```
Hello Ravi
arepallyramcharan@Arepallys-MacBook-Air assignment-10.2.py % /usr/bin/python3 "/Users/arepallyramcharan/ai assistent.py/assignment-10.2.py/task5.py"
Total numbers generated: 499999
arepallyramcharan@Arepallys-MacBook-Air assignment-10.2.py %
```

Observation:

The original loop used repeated `.append()` operations, which are slower.

1. List comprehension improves performance and readability.
2. Avoiding unnecessary list creation reduces memory usage.
3. Using `len(range())` is the most memory-efficient approach when only count is needed.
4. Optimization depends on requirement (data storage vs result only).