ASSIGNMENT:10.2

HALL TICKET:2303A51598

BATCH:29

Task Description -1(Error Detection and Correction)

PROMPT:

Use AI to analyze a Python script and correct all syntax and logical

errors.

def calculate_total(nums)

sum = 0

for n in nums

sum += n

return total

```
#Use AI to analyze a Python script and correct all syntax and logical errors.
# def calculate_total(nums)
# sum = 0
# for n in nums
# sum += n
# return total
def calculate_total(nums):
    """
    Calculate the total sum of a list of numbers.

    Args:
        nums (list): A list of numeric values

    Returns:
        int or float: The total sum of the numbers in the list
    """
    total = 0  # Initialize total to 0
    for n in nums:  # Iterate through each number in the list        total += n  # Add the current number to the total
    return total  # Return the final total
```

JUSTIFICATION:

This task helps students learn how to identify and fix syntax errors (missing colons, indentation, etc.) and logical errors (wrong variables, incorrect output). It

improves debugging skills and teaches how AI can assist in making code executable and correct. It also builds confidence in analyzing faulty programs.

Task Description -2(Code Style Standardization)

PROMPT:

Use AI to refactor Python code to comply with standard coding style

Guidelines

```
#Task Description -2(Code Style Standardization)

#Use AI to refactor Python code to comply with standard coding style guidelines
#def findSum(a,b):
# return a+b
def find_sum(a, b):
    """
    Calculate the sum of two numbers.

    Args:
        a (int or float): The first number
        b (int or float): The second number

    Returns:
        int or float: The sum of the two numbers
    """
    return a + b
```

JUSTIFICATION:

This task focuses on writing clean, readable, and professional Python code following standard conventions (PEP-8). It helps students understand the importance of proper formatting, spacing, naming, and structure. Using AI here demonstrates how automated tools can improve code quality and maintainability.

Task Description -3(Code Clarity Improvement)

PROMPT:

Use AI to improve code readability without changing its functionality.

def f(x,y):

return x-y*2

print(f(10,3))

```
#Use AI to improve code readability without changing its functionality.
#def f(x,y):
# return x-y*2
# print(f(10,3))
def calculate_difference(x, y):
    """
    Calculate the difference between two numbers, where the second number is multiplied by 2 before subtraction.

    Args:
        x (int or float): The first number
        y (int or float): The second number

    Returns:
        int or float: The result of x minus (y multiplied by 2)
    """
    return x - (y * 2)
print(calculate_difference(10, 3))  # Output: 4 - This will calculate the difference as 10 - (3 * 2) = 10 - 6 = 4
```

JUSTIFICATION:

This task improves program readability by replacing unclear function and variable names with meaningful ones. It teaches that code should be easy to understand, not just correct. Clear code helps future developers (and yourself) quickly understand logic, reducing errors during maintenance.

Task Description -4(Structural Refactoring)

PROMPT:

Use AI to refactor repetitive code into reusable functions.

print("Hello Ram")

print("Hello Sita")

print("Hello Ravi")

```
#Use AI to refactor repetitive code into reusable functions.
#print("Hello Ram")
# print("Hello Sita")
# print("Hello Ravi")
# def greet(name):
#     """
#     Print a greeting message for the given name.

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

#     Returns:
#         None
#     """
#     print(f"Hello {name}")
# greet("Ram")
# greet("Sita")
# greet("Ravi")
```

JUSTIFICATION:

This task introduces modular programming by removing repetition and creating reusable functions. It shows how AI can help refactor duplicated code into cleaner structures, making programs more scalable and organized. It also promotes the DRY principle (Don't Repeat Yourself).

Task Description -5(Efficiency Enhancement)

PROMPT:

Use AI to optimize Python code for better performance.

numbers = [ ]

for i in range(1, 500000):

numbers.append(i * i)

print(len(numbers))

```python
#Use AI to optimize Python code for better performance.
# numbers = [ ]
# for i in range(1, 500000):
# numbers.append(i * i)
# print(len(numbers))
# squares = [i * i for i in range(1, 500000)]  # Using list comprehension for better performance
# print(len(squares) )
```

JUSTIFICATION:

This task teaches performance optimization by converting slow loops into faster approaches like list comprehensions. It highlights how AI can improve execution speed and memory usage. Students learn to write efficient Python programs, which is important for real-world applications with large data.