AI Assisted Coding Assignment-10.2

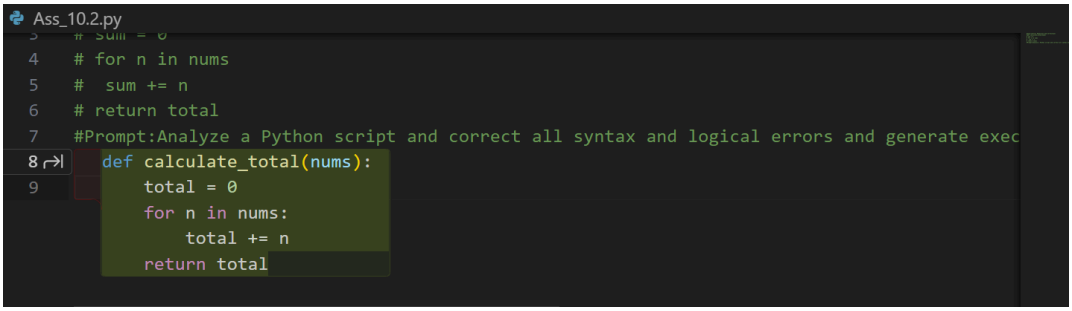
> 2303A51543 BT: 29 VELDI.HRUTHIKA 24.02.2026

💡 ***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.

```
#Task-1(Error Detection and Correction)
#def calculate_total(nums)
# sum = 0
# for n in nums
#  sum += n
# return total
#Prompt:Analyze a Python script and correct all syntax and logical errors and gener
ate executable Python code with brief explanations of the identified syntax and log
ic errors
# The original code has several syntax and logical errors:
# 1. The function definition is missing a colon (:) at the end of the line.
# 2. The code uses sum to accumulate the total, but the function returns an undefin
ed variable total. We should use total consistently.
# 3. The for loop is missing a colon (:) at the end of the line.

def calculate_total(nums):
    """
    Calculate the sum of all numbers in a given list.

    Parameters:
    nums (list): A list of numerical values (integers or floats) to be summed.

    Returns:
    int or float: The total sum of the numbers in the input list.

    Example:
```

```
    >>> calculate_total([1, 2, 3, 4])
    10
    """
    total = 0  # Initialize total to 0
    for n in nums:  # Iterate through each number in the input list 'nums'
        total += n  # Add each number to the total
    return total  # Return the final total

# Test cases
print(calculate_total([1, 2, 3, 4]))  # Expected: 10
```



💡 ***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

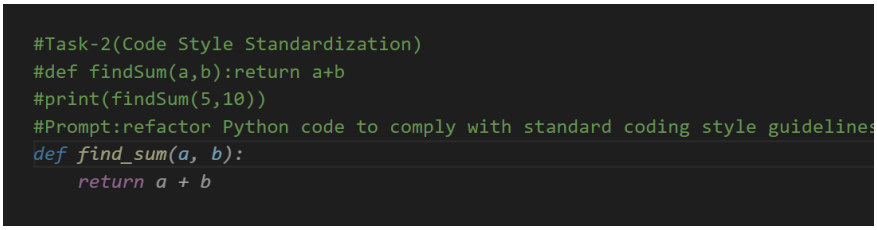
print(findSum(5,10))

Expected Output-2:

Well-structured, consistently formatted Python code following standard style conventions.

```
#Task-2(Code Style Standardization)
#def findSum(a,b):return a+b
#print(findSum(5,10))
#Prompt:refactor Python code to comply with standard coding style guidelines(No nee
d of explanation)
def find_sum(a, b):
    """
    Calculate the sum of two numbers.

    Parameters:
    a (int or float): The first number to be added.
    b (int or float): The second number to be added.

    Returns:
```

```
    int or float: The sum of the two input numbers.

    Example:
    >>> find_sum(5, 10)
    15
    """
    return a + b

# Example usage
print(find_sum(5, 10))  # Expected output: 15
```



💡 ***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))

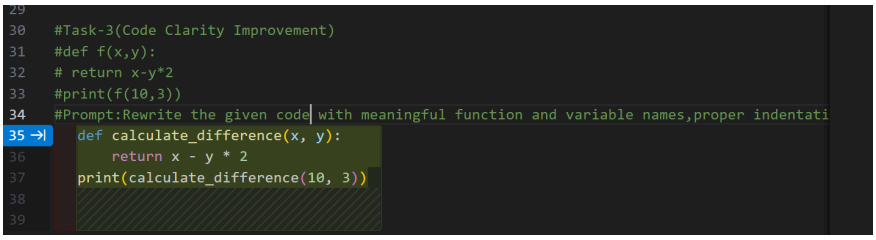
Expected Output-3:

Python code rewritten with meaningful function and variable names,

proper indentation, and improved clarity.

```
#Task-3(Code Clarity Improvement)
#def f(x,y):
# return x-y*2
#print(f(10,3))
#Prompt:Rewrite the given code with meaningful function and variable names,proper i
ndentation, and improved clarity

def calculate_difference(initial_value, subtract_value):
    """
    Calculate the difference by subtracting twice the 'subtract_value' from the 'in
itial_value'.

    Parameters:
    initial_value (int or float): The value from which the product of 'subtract_val
ue' and 2 will be subtracted.
    subtract_value (int or float): The value that will be doubled and subtracted fr
```

```
om 'initial_value'.

    Returns:
    int or float: The result of the calculation (initial_value - 2 * subtract_valu
e).

    Example:
    >>> calculate_difference(10, 3)
    4
    """
    return initial_value - subtract_value * 2

# Example usage
print(calculate_difference(10, 3))  # Expected output: 4
```



💡 ***Task Description -4(Structural Refactoring)***

Task:

Use AI to refactor repetitive code into reusable functions.

Sample Input Code:

print("Hello Ram")

print("Hello Sita")

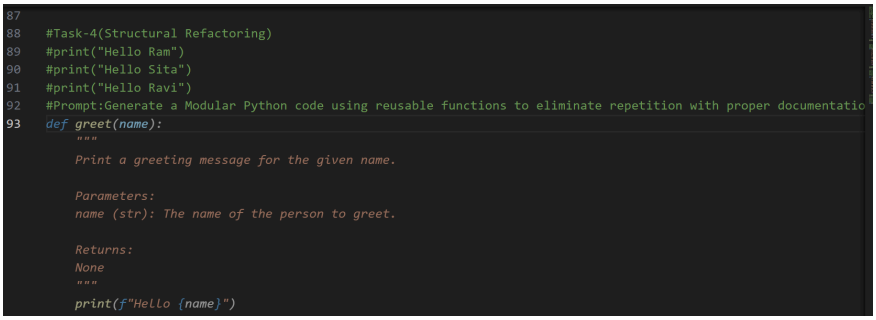
print("Hello Ravi")

Expected Output-4:

Modular Python code using reusable functions to eliminate repetition.

```
#Task-4(Structural Refactoring)
#print("Hello Ram")
#print("Hello Sita")
#print("Hello Ravi")
#Prompt:Generate a Modular Python code using reusable functions to eliminate repeti
tion with proper documentation in the code
def greet(name):
    """
    Print a greeting message for the given name.

    Parameters:
    name (str): The name of the person to greet.
```

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



💡 ***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.

```
#Task-5(Efficiency Enhancement)
#numbers = [ ]
#for i in range(1, 500000):
# numbers.append(i * i)
#print(len(numbers))
#Prompt:Optimize the  Python code that achieves the same result with improved perfo
rmance with proper documentaation in the code.
def generate_squares(n):
    """
    Generate a list of squares of numbers from 1 to n-1.

    Parameters:
    n (int): The upper limit (exclusive) for generating squares.
```

```
    Returns:
    list: A list containing the squares of numbers from 1 to n-1.

    Example:
    >>> generate_squares(10)
    [1, 4, 9, 16, 25, 36, 49, 64, 81]
    """
    return [i * i for i in range(1, n)]  # Using list comprehension for efficient g
eneration of squares
# Example usage
squares = generate_squares(500000)
print(len(squares))  # Expected output: 499999
```