Assignment – 10.2

2303A51927

Task 1 – Error Detection and Correction

PROMPT :

#def calculate_total(nums)

sum = 0

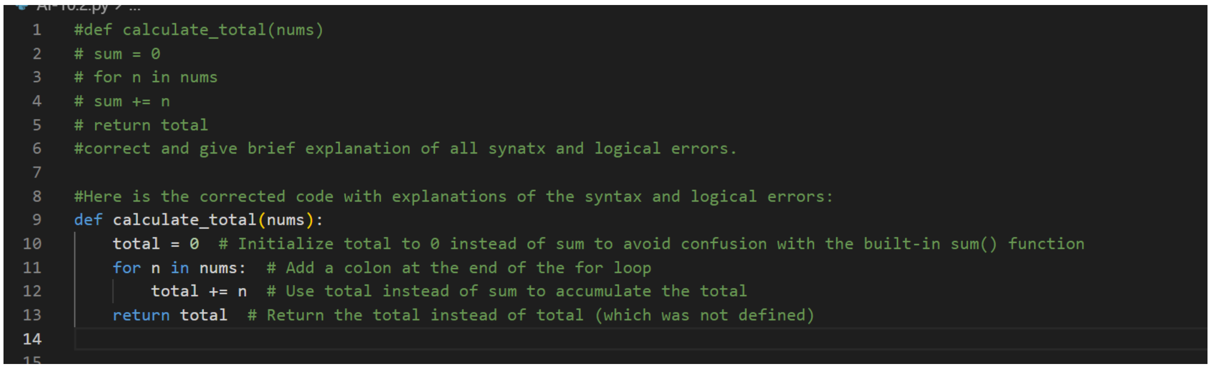
for n in nums

sum += n

return total

#correct and give brief explanation of all synatx and logical errors.

CODE :



```
#def calculate_total(nums)
# sum = 0
# for n in nums
# sum += n
# return total
#correct and give brief explanation of all synatx and logical errors.

#Here is the corrected code with explanations of the syntax and logical errors:
def calculate_total(nums):
    total = 0  # Initialize total to 0 instead of sum to avoid confusion with the built-in sum() function
    for n in nums:  # Add a colon at the end of the for loop
        total += n  # Use total instead of sum to accumulate the total
    return total  # Return the total instead of total (which was not defined)
```

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 2 – Code Style Standardization

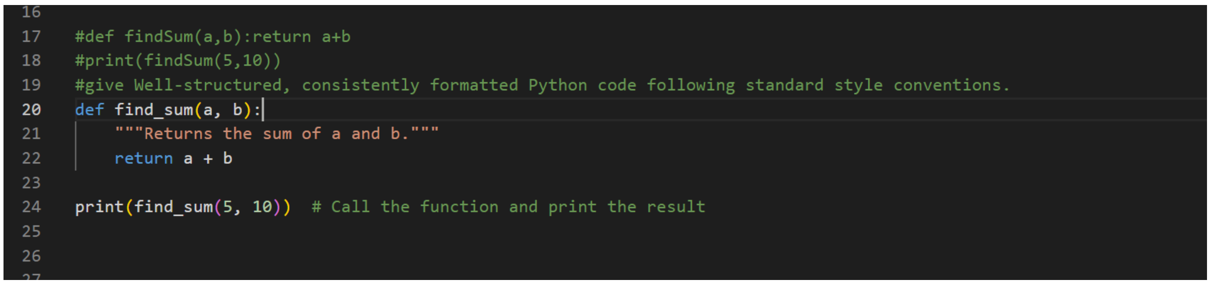
PROMPT :

#def findSum(a,b):return a+b

#print(findSum(5,10))

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

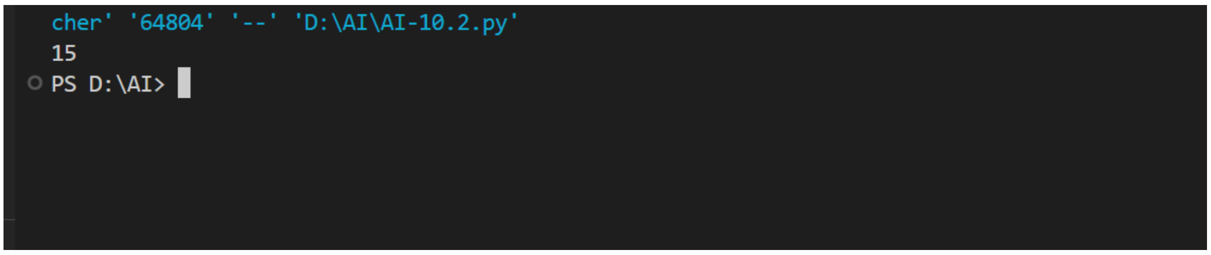
CODE :



```python
#def findSum(a,b):return a+b
#print(findSum(5,10))
#give Well-structured, consistently formatted Python code following standard style conventions.
def find_sum(a, b):
    """Returns the sum of a and b."""
    return a + b

print(find_sum(5, 10))  # Call the function and print the result
```

OUTPUT :



```
cher' '64804' '--' 'D:\AI\AI-10.2.py'
15
PS D:\AI>
```

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 3 – Code Clarity Improvement

PROMPT :

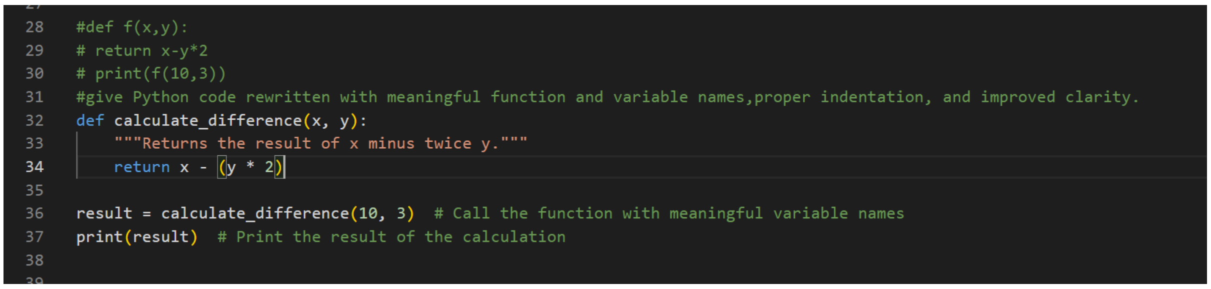
#def f(x,y):

return x-y*2

print(f(10,3))

#give Python code rewritten with meaningful function and variable names,proper indentation, and improved clarity.

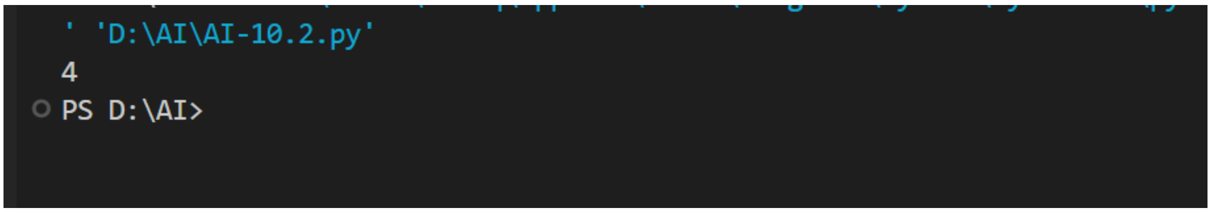
CODE :



```python
#def f(x,y):
# return x-y*2
# print(f(10,3))
#give Python code rewritten with meaningful function and variable names,proper indentation, and improved clarity.
def calculate_difference(x, y):
    """Returns the result of x minus twice y."""
    return x - (y * 2)

result = calculate_difference(10, 3)  # Call the function with meaningful variable names
print(result)  # Print the result of the calculation
```

OUTPUT :



```
' 'D:\AI\AI-10.2.py'
4
PS D:\AI>
```

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 4 – Structural Refactoring

PROMPT :

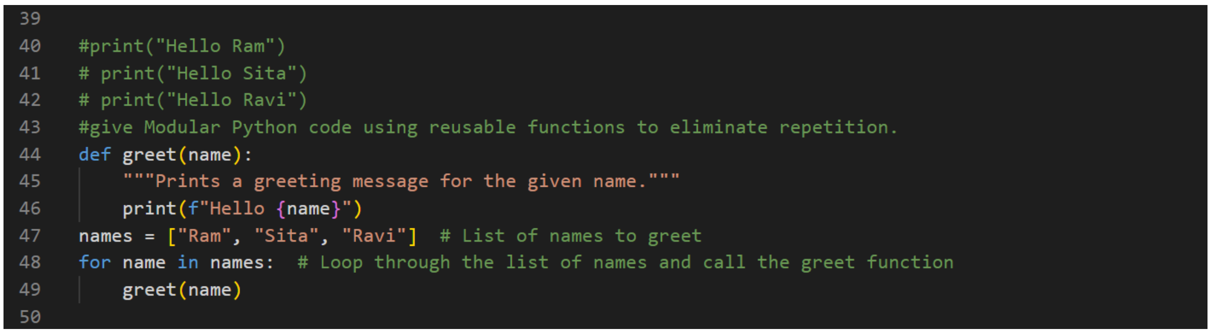
print("Hello Ram")

print("Hello Sita")

print("Hello Ravi")

#give Modular Python code using reusable functions to eliminate repetition.

CODE :

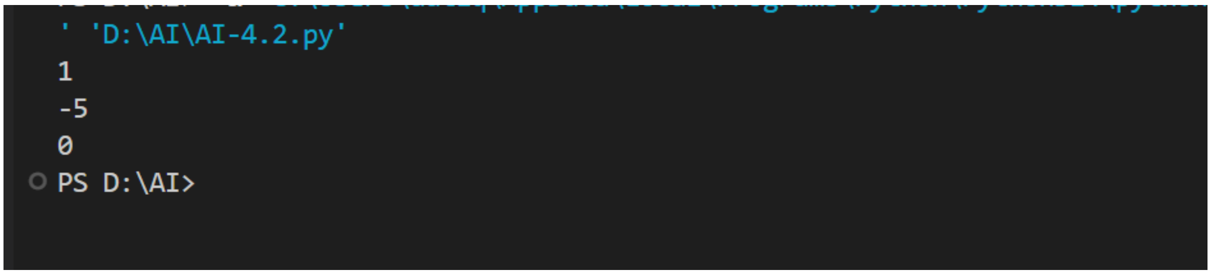


```python

#print("Hello Ram")
# print("Hello Sita")
# print("Hello Ravi")
#give Modular Python code using reusable functions to eliminate repetition.
def greet(name):
    """Prints a greeting message for the given name."""
    print(f"Hello {name}")
names = ["Ram", "Sita", "Ravi"]  # List of names to greet
for name in names:  # Loop through the list of names and call the greet function
    greet(name)

```

OUTPUT :



```
' 'D:\AI\AI-4.2.py'
1
-5
0
PS D:\AI>
```

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 5 – Efficiency Enhancement

PROMPT :

numbers = [ ]

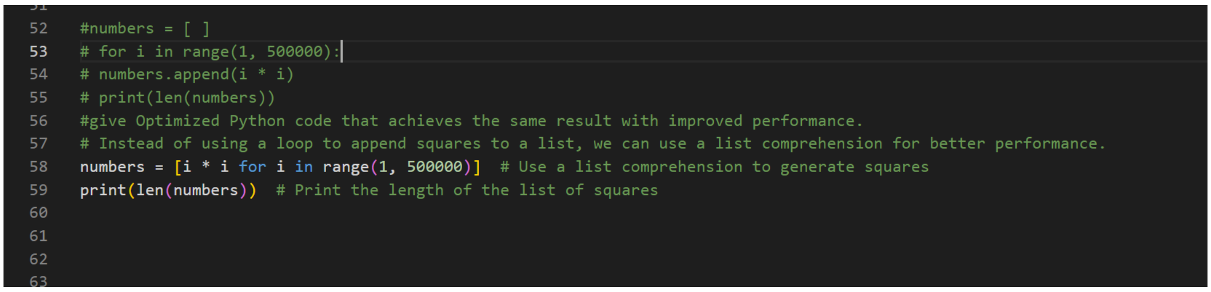
for i in range(1, 500000):

numbers.append(i * i)

print(len(numbers))

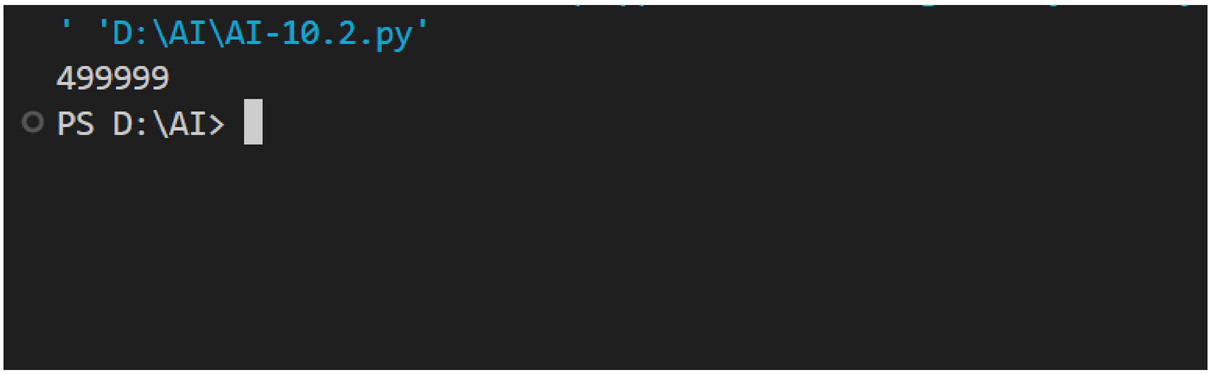
#give Optimized Python code that achieves the same result with improved performance.

CODE :



```python
#numbers = [ ]
# for i in range(1, 500000):
# numbers.append(i * i)
# print(len(numbers))
#give Optimized Python code that achieves the same result with improved performance.
# Instead of using a loop to append squares to a list, we can use a list comprehension for better performance.
numbers = [i * i for i in range(1, 500000)]  # Use a list comprehension to generate squares
print(len(numbers))  # Print the length of the list of squares
```

OUTPUT :



```
' 'D:\AI\AI-10.2.py'
499999
PS D:\AI>
```

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.