

## ASSIGNMENT-10.2

2303A510i9

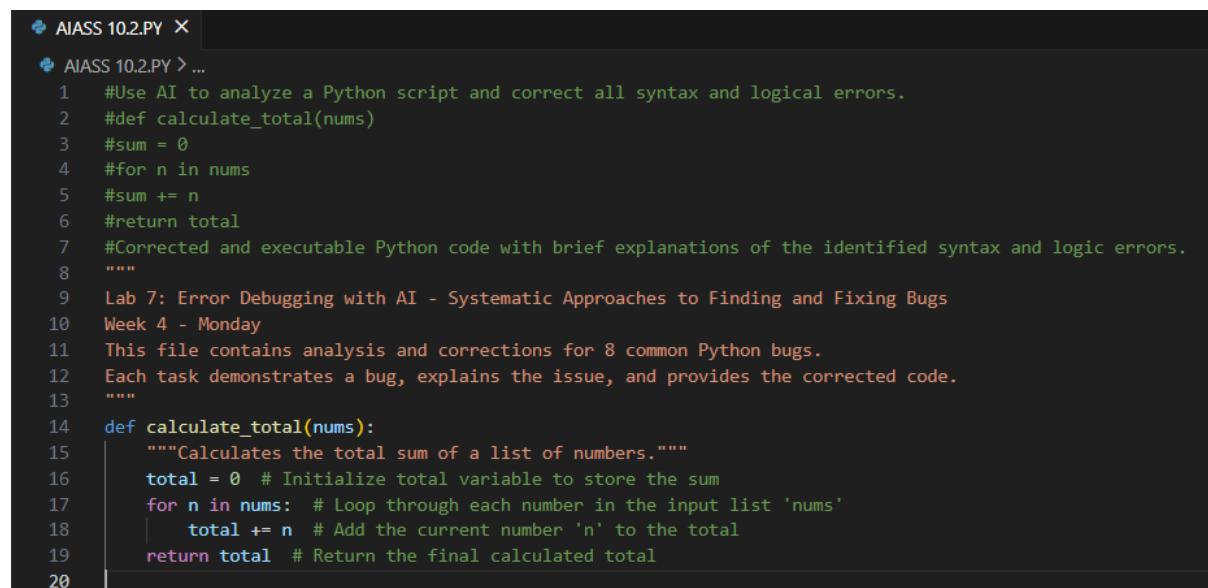
BATCH:30

**TASK1:** 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
```

**CODE:**



The screenshot shows a code editor window titled "AIASS 10.2.PY X". The code is a Python function named "calculate\_total" that sums up a list of numbers. The AI has provided corrections and explanations for the original code. The corrected code is as follows:

```
1  #Use AI to analyze a Python script and correct all syntax and logical errors.
2  #def calculate_total(nums)
3  #sum = 0
4  #for n in nums
5  #sum += n
6  #return total
7  #Corrected and executable Python code with brief explanations of the identified syntax and logic errors.
8  """
9  Lab 7: Error Debugging with AI - Systematic Approaches to Finding and Fixing Bugs
10 Week 4 - Monday
11 This file contains analysis and corrections for 8 common Python bugs.
12 Each task demonstrates a bug, explains the issue, and provides the corrected code.
13 """
14 def calculate_total(nums):
15     """Calculates the total sum of a list of numbers."""
16     total = 0 # Initialize total variable to store the sum
17     for n in nums: # Loop through each number in the input list 'nums'
18         total += n # Add the current number 'n' to the total
19     return total # Return the final calculated total
20
```

**JUSTIFICATION:**

The original code had syntax errors (missing colons) and a logical mistake (returning an undefined variable). Correcting these ensures the function runs properly and consistently returns the intended sum.

**TASK 2:** 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))
```

## CODE:

```
20
21 #Use AI to refactor Python code to comply with standard coding style guidelines.
22 #Sample Input Code:
23 #def findSum(a,b):return a+b
24 #print(findSum(5,10))
25 #Well-structured, consistently formatted Python code following standard style conventions.
26 def find_sum(a, b):
27     """Returns the sum of two numbers."""
28     return a + b
29 print(find_sum(5, 10))
30
```

## **JUSTIFICATION:**

Refactoring to PEP 8 standards (snake\_case, spacing, multi-line structure) makes the code easier to read, maintain, and share with others, aligning with professional Python practices.

**TASK3:** 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))
```

CODE

```
30
31 #Use AI to improve code readability without changing its functionality.
32 #Sample Input Code:
33 #def f(x,y):
34 #    return x-y**2
35 #print(f(10,3))
36 #Python code rewritten with meaningful function and variable names, proper indentation, and improved clarity
37 def calculate_difference(x, y):
38     """Returns the difference of two numbers."""
39     return x - y * 2
40 print(calculate_difference(10, 3))
41
```

## **JUSTIFICATION:**

Using meaningful names for functions and variables clarifies the purpose of the code. This improves readability and makes it easier for others (or your future self) to understand without guessing.

**TASK 4:** Use AI to refactor repetitive code into reusable functions.

## Sample Input Code:

```
print("Hello Ram")  
print("Hello Sita")  
print("Hello Ravi")
```

## CODE:

## **JUSTIFICATION:**

By introducing a reusable function, repetitive print statements are eliminated. This modular approach reduces redundancy, improves maintainability, and allows easy scaling if more names need greetings.

**Task 5:** 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))
```

CODE:

```
54 #Task: Use AI to optimize Python code for better performance.
55 #Sample Input Code:
56 #numbers = [ ]
57 #for i in range(1, 50000):
58 #    numbers.append(i * i)
59 #print(len(numbers))
60 #Optimized Python code that achieves the same result with improved performance.
61 numbers = [i * i for i in range(1, 50000)] # Using list comprehension for better performance
62 print(len(numbers))
63
64
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

▼ TERMINAL

- PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> & "C:/Users/Ramya Sri/OneDrive/New folder/AI Assistant coding(.venv) PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> & 'c:\Users\Ramya Sri\OneDrive\New folder\thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64598' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\499999
- 499999
- .venv) PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> & "c:\Users\Ramya Sri\OneDrive\New folder\

### JUSTIFICATION:

Switching from a loop with `.append()` to a list comprehension improves performance because comprehensions are optimized internally. The result is the same, but execution is faster and more Pythonic.