SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

ProgramName:B. Tech	**AssignmentType: Lab**	**AcademicYear:**2025-2026

CourseCoordinatorName	Dr. Rishabh Mittal		
Instructor(s)Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpulasumalatha Mr. Bingi Raju		
CourseCode	23CS002PC304	**CourseTitle**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
DateandDay of Assignment	Week5 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	23CSBTB01 To 23CSBTB52
AssignmentNumber:10.1(Presentassignmentnumber)/**24**(Totalnumberofassignments)			

Q.No.	**Question**	***Expected Time to complete***

1

Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability

Lab Objectives

- Use AI for automated code review and quality enhancement.
- Identify and fix syntax, logical, performance, and security issues in Python code.
- Improve readability and maintainability through structured refactoring and comments.
- Apply prompt engineering for targeted improvements.
- Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices

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 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 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
```

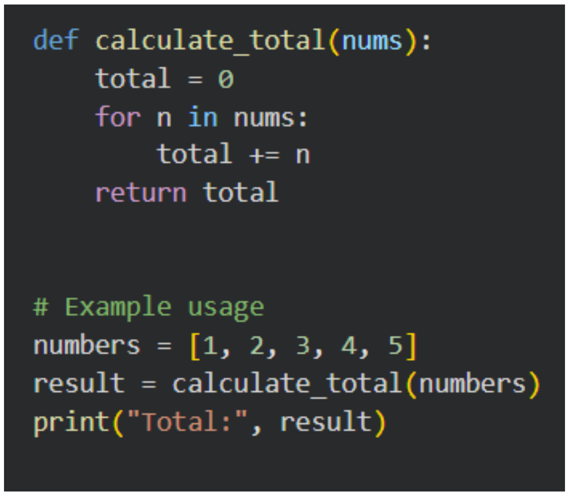
Week5 - Tuesday

Task 1: (Error Detection and Correction)

Prompt:

Analyze the following Python code and identify all syntax and logical errors. Provide corrected executable code along with brief explanations of each issue found.

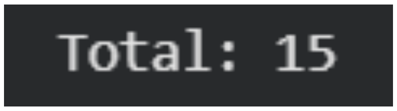
Code:



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


# Example usage
numbers = [1, 2, 3, 4, 5]
result = calculate_total(numbers)
print("Total:", result)
```

Sample Input/Output:



```
Total: 15
```

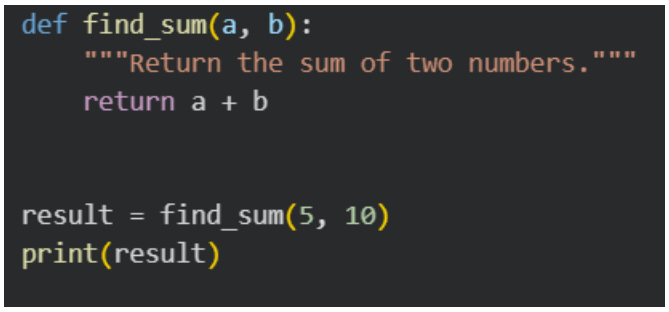
Explanation:

- The function definition was missing a colon (:), which causes a syntax error.
- The for loop statement also lacked a colon, making the loop invalid.
- The variable sum was used initially but total was returned, causing a logical NameError.
- The built-in function name sum was overridden, which is not recommended in Python.
- Indentation structure was unclear, which could lead to execution errors.
- The corrected version ensures consistent variable naming and proper syntax for execution.

Task 2 : (Code Style Standardization)

Prompt:
Refactor the following Python code to follow standard coding style guidelines (PEP 8). Improve formatting, indentation, naming conventions, and structure without changing functionality.

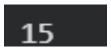
Code:



```python
def find_sum(a, b):
    """Return the sum of two numbers."""
    return a + b


result = find_sum(5, 10)
print(result)
```

Sample Input/Output:



```
15
```

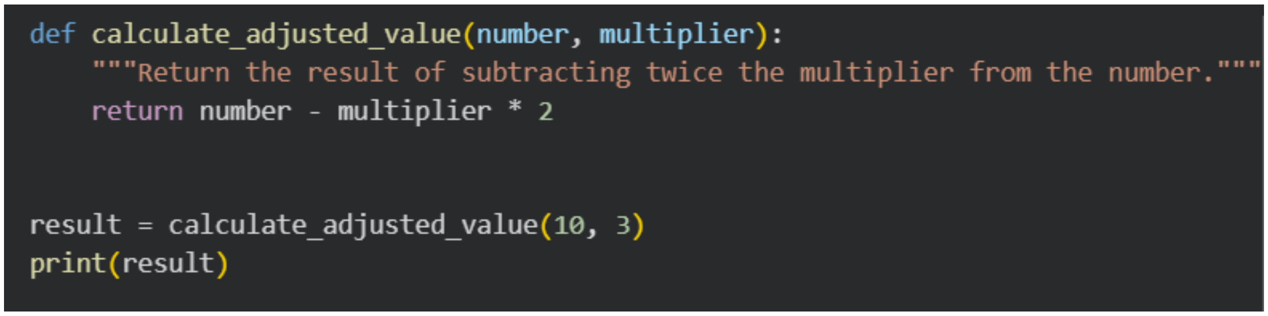
Explanation:

- The function name was changed from findSum to find_sum to follow snake_case naming convention as per PEP 8.
- Proper spacing was added between parameters (a, b) to improve readability.
- The one-line function was expanded into a properly indented multi-line structure.
- A docstring was added to improve code documentation and maintainability.
- The function call was stored in a variable (result) before printing for better structure.
- Overall formatting now follows Python's standard coding style guidelines.

Task 3: (Code Clarity Improvement)

Prompt:
Improve the readability of the following Python code without changing its functionality use meaningful function and variable names, proper indentation, and clear structure.

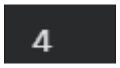
Code:



```python
def calculate_adjusted_value(number, multiplier):
    """Return the result of subtracting twice the multiplier from the number."""
    return number - multiplier * 2


result = calculate_adjusted_value(10, 3)
print(result)
```

Sample Input/Output:



```
4
```

Explanation:

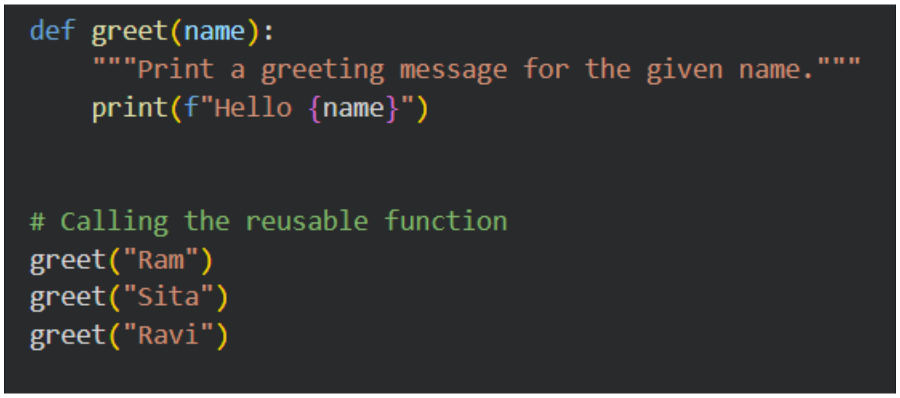
- The function name f was renamed to calculate_adjusted_value to clearly describe its purpose.
- Parameter names x and y were changed to number and multiplier for better understanding.
- Proper indentation was maintained to follow Python formatting standards.
- A docstring was added to explain what the function does.
- The function call result was stored in a variable (result) to improve clarity.
- The logic remains unchanged; only readability and structure were improved.

Task 4: (Structural Refactoring)

Prompt:

Refactor the following repetitive Python code into reusable functions.
Eliminate redundancy while maintaining the same output.

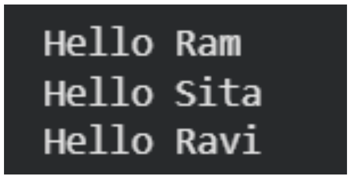
Code:



```python
def greet(name):
    """Print a greeting message for the given name."""
    print(f"Hello {name}")


# Calling the reusable function
greet("Ram")
greet("Sita")
greet("Ravi")
```

Sample Input/Output:



```
Hello Ram
Hello Sita
Hello Ravi
```

Explanation:

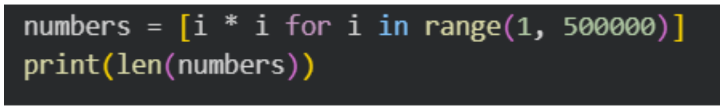
- The repeated print statements were refactored into a single reusable function named greet.
- A parameter name was introduced to allow dynamic greeting messages.
- This eliminates code repetition and improves maintainability.
- The function follows modular programming principles.
- A docstring was added for better documentation and readability.
- The output remains exactly the same as the original code.

Task 5: (Efficiency Enhancement)

Prompt:

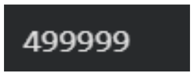
Optimize the following Python code to improve performance without changing its output. Apply efficient Python techniques and maintain readability

Code:



```
numbers = [i * i for i in range(1, 500000)]
print(len(numbers))
```

Sample Input/Output:



```
499999
```

Explanation:

- The original code used a loop with append(), which is slower due to repeated list resizing operations.
- The optimized version uses a list comprehension, which is faster and more memory-efficient in Python.
- List comprehensions are internally optimized and execute quicker than manual loops.
- The functionality remains exactly the same generating squares from 1 to 499999.
- The length of the list is unchanged, ensuring identical output.
- The new version improves performance while keeping the code concise and readable.