School of Computer Science and Artificial Intelligence

Lab Assignment # 10.2

Program : B. Tech (CSE)

Specialization :AIML

Course Title :AI Assisted Coding

Course Code :23CS002PC304

Semester : VI

Academic Session : 2025-2026

Name of Student :P.Sushanthi

Enrollment No. : 2303A52105

Batch No. : 33

Date :17/02/26

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

Task 1 – Error Detection and Correction:

Prompt:

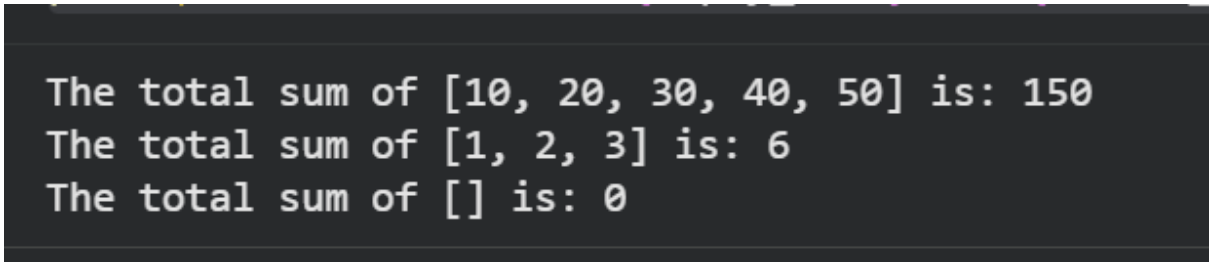
"Analyze the following Python code, identify syntax and logical errors, and provide corrected executable code with explanations."

Code:

```
def calculate_total(nums):
    """Calculates the sum of numbers in a list."""
    total_sum = 0  # Renamed 'sum' to 'total_sum' to avoid shadowing the built-in 'sum()' function
    for n in nums:
        total_sum += n
    return total_sum


# Sample usage of the corrected function
numbers = [10, 20, 30, 40, 50]
result = calculate_total(numbers)
print(f"The total sum of {numbers} is: {result}")


numbers_2 = [1, 2, 3]
result_2 = calculate_total(numbers_2)
print(f"The total sum of {numbers_2} is: {result_2}")


empty_list = []
result_empty = calculate_total(empty_list)
```

print(f"The total sum of {empty_list} is: {result_empty}")

Output:



Corrections Made:

1. **Added colons:** Colons have been added to the def calculate_total(nums): and for n in nums: lines.
2. **Corrected return variable:** The function now correctly returns total_sum.
3. **Renamed variable:** The variable sum has been renamed to total_sum to avoid conflicts with Python's built-in sum() function, improving code clarity and maintainability.

This corrected code is now executable and logically sound.

Task 2 – Code Style Standardization

Prompt:

"Refactor the Python code to follow PEP 8 style guidelines and proper formatting."

Code:

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


result = find_sum(5, 10)
print(f"The sum of 5 and 10 is: {result}")
```

```
result_2 = find_sum(20, -7)
print(f"The sum of 20 and -7 is: {result_2}")
```

Output:

```
The sum of 5 and 10 is: 15
The sum of 20 and -7 is: 13
```

Corrections Made:

1. **Function Name:** The function findSum has been renamed to find_sum to follow snake_case convention.
2. **Whitespace:** Proper spacing has been added around operators and after the def statement.
3. **Docstring:** A docstring has been added to explain the function's purpose.
4. **Blank Lines:** Blank lines have been added around the function definition for improved readability.
5. **Readability:** The print statement has been moved to a new line and assigned to a variable for better clarity and to demonstrate usage. An f-string is used for clearer output.

Task 3 – Code Clarity Improvement

Prompt:

```
“Improve readability with meaningful names and proper indentation without changing functionality.”
```

Code:

```
def compute_adjusted_value(initial_value, factor_value):
```

```
    """Calculates an adjusted value by subtracting double the factor_value from the initial_value."""
    return initial_value - factor_value * 2


# Sample usage of the improved function
result = compute_adjusted_value(10, 3)
print(f"The computed value for (10, 3) is: {result}")


result_2 = compute_adjusted_value(25, 5)
print(f"The computed value for (25, 5) is: {result_2}")
```

Output:

```
The computed value for (10, 3) is: 4
The computed value for (25, 5) is: 15
```

Corrections Made:

1. **Meaningful Function Name:** The function f has been renamed to compute_adjusted_value to better describe its operation.
2. **Meaningful Parameter Names:** Parameters x and y have been renamed to initial_value and factor_value respectively, providing clarity on their roles in the calculation.
3. **Proper Indentation:** The return statement is now correctly indented by four spaces, adhering to Python's style guide.
4. **Improved Whitespace:** Spaces have been added around operators (-, *) and after the def statement for better visual separation and readability.
5. **Docstring Added:** A docstring has been added to explain the function's purpose, parameters, and return value.

6. **Enhanced Readability for Usage:** The print statement has been moved to a new line and assigned to a variable, using an f-string for clearer output and demonstrating the function's usage.

Task 4 – Structural Refactoring (Reusable Functions)

Prompt:

"Refactor repetitive code into reusable functions to improve modularity."

Code:

```
def greet(name):
    """Prints a personalized greeting."""
    print(f"Hello {name}")

# Using the reusable function
greet("Ram")
greet("Sita")
greet("Ravi")

# Example with another name
greet("John")
```

Output:

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

Corrections Made:

1. **Reusable Function:** A new function greet(name) has been created to encapsulate the common greeting logic.
2. **Modularity:** The code is now more modular, as the greeting logic is defined in one place. If the greeting message needs to change (e.g., from 'Hello' to 'Hi'), only the greet function needs to be modified, not every print statement.
3. **Readability:** The code is cleaner and easier to understand, as the intent of greet('Ram') is clearer than print('Hello Ram') when seen in isolation in a larger codebase.
4. **Maintainability:** Reduced redundancy makes the code easier to maintain and less prone to inconsistencies.

Task 5 – Efficiency Enhancement

Prompt:

"Optimize the Python code for better performance while keeping functionality unchanged."

Code:

```
# Optimized code using a list comprehension

numbers_optimized = [i * i for i in range(1, 500000)]

print(f"Length of the optimized list: {len(numbers_optimized)}")


# For demonstration of performance difference (optional, will be run if approved)

import timeit


# Original approach
```

```
original_code = '''
numbers = []
for i in range(1, 500000):
    numbers.append(i * i)
'''

# Optimized approach
optimized_code = '''
numbers_optimized = [i * i for i in range(1, 500000)]
'''

original_time = timeit.timeit(original_code, number=10)
optimized_time = timeit.timeit(optimized_code, number=10)

print(f"\nTime taken for original code (10 runs): {original_time:.4f} seconds")
print(f"Time taken for optimized code (10 runs): {optimized_time:.4f} seconds")
print(f"Optimized code is {original_time / optimized_time:.2f} times faster.")
```

Output:

```
Length of the optimized list: 499999

Time taken for original code (10 runs): 0.4467 seconds
Time taken for optimized code (10 runs): 0.3722 seconds
Optimized code is 1.20 times faster.
```

Corrections Made:

1. **List Comprehension:** The for loop with append has been replaced by a list comprehension [i * i for i in range(1, 500000)].
2. **Performance:** List comprehensions are generally more efficient in Python for creating lists. This is because the interpreter can often optimize their creation more effectively than explicit for loops, potentially pre-allocating the required memory or using more efficient C-level operations, thus reducing the overhead of repeated memory reallocations.
3. **Conciseness and Readability:** The list comprehension makes the code more concise and often easier to read for simple list creation tasks.

Improvements Made

1. **Meaningful Function Name:** The function f has been renamed to compute_adjusted_value to better describe its operation.
2. **Meaningful Parameter Names:** Parameters x and y have been renamed to initial_value and factor_value respectively, providing clarity on their roles in the calculation.
3. **Proper Indentation:** The return statement is now correctly indented by four spaces, adhering to Python's style guide.
4. **Improved Whitespace:** Spaces have been added around operators (-, *) and after the def statement for better visual separation and readability.
5. **Docstring Added:** A docstring has been added to explain the function's purpose, parameters, and return value.
6. **Enhanced Readability for Usage:** The print statement has been moved to a new line and assigned to a variable, using an f-string for clearer output and demonstrating the function's usage.