AI ASSISTANT CODING

Assignment – 10.2

Name: K. Shailaja

H.T No.: 2303A52155

Batch: 34

#Question:

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.

Prompt:

Analyze the following Python code and correct all syntax and logical errors. Provide corrected executable code and briefly explain the errors.

Code:

```
def calculate_total(nums)

sum = 0

for n in nums

sum += n

return total
```

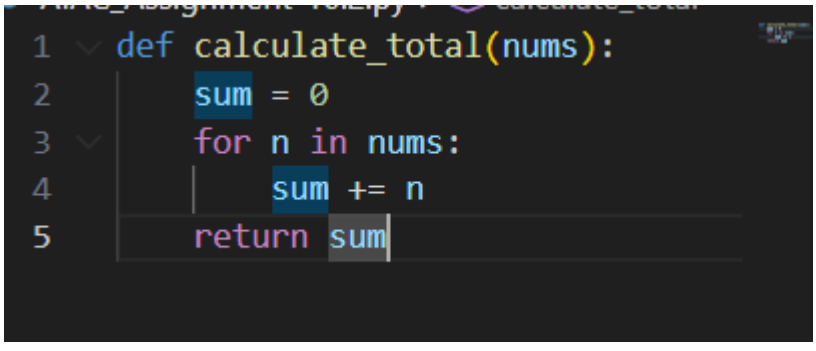
Sample Input Code:

```
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.

#Corrected Code:



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

Errors Found:

1. Missing colon after function definition - Line 1
   needs : after (nums)

2. Missing colon after for loop - Line 3 needs : after nums
3. Variable name mismatch - Line 5 returns total but the variable is named sum

Explanation:

- Python requires a colon (:) after function definitions and control flow statements like for loops
- Variable names must be consistent - the function initializes sum but was attempting to return total, which doesn't exist
- Proper indentation is also important in Python; the loop body and return statement should be indented

This corrected version will now properly sum all numbers in the input list and return the result.

Task Description -2(Code Style Standardization)

Task:

Use AI to refactor Python code to comply with standard coding style guidelines.

Prompt:

Refactor the following Python code to follow PEP 8 coding standards. Improve formatting, spacing, and readability without changing functionality.

Code:

```
def findSum(a,b):
  return a+b
print(findSum(5,10))
```

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.

#Output

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

    Args:
        a: First number
        b: Second number

    Returns:
        The sum of a and b
    """
    return a + b


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

Task Description -3(Code Clarity Improvement)

Task:

Use AI to improve code readability without changing its functionality.

#Prompt:

Improve the readability of the following Python code by using meaningful function and variable names, proper indentation, and comments. Do not change functionality.

Code:

def f(x,y):

return x-y*2

print(f(10,3))

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.

#Output

```
def calculate_adjusted_value(base_value, multiplier):
    """Calculate base value minus twice the multiplier.

    Args:
        base_value: The initial numeric value
        multiplier: The value to double and subtract from base

    Returns:
        The result of base_value - (multiplier * 2)
    """
    # Subtract double the multiplier from the base value
    adjusted_result = base_value - (multiplier * 2)
    return adjusted_result


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

Task Description -4(Structural Refactoring)

Task:

Use AI to refactor repetitive code into reusable functions.

#Prompt:

Refactor the following repetitive Python code into reusable functions to improve modularity and maintainability.

Code:

```
print("Hello Ram")

print("Hello Sita")

print("Hello Ravi")
```

Sample Input Code:

```
print("Hello Ram")

print("Hello Sita")

print("Hello Ravi")
```

Expected Output-4:

Modular Python code using reusable functions to eliminate repetition.

#Output

```python
def greet(name):
    """Print a greeting message for the given name.

    Args:
        name: The person's name to greet
    """
    print(f"Hello {name}")


def greet_multiple(names):
    """Greet multiple people.

    Args:
        names: A list of names to greet
    """
    for name in names:
        greet(name)


# Using the reusable functions
people = ["Ram", "Sita", "Ravi"]
greet_multiple(people)
```

Task Description -5(Efficiency Enhancement)

Task:

Use AI to optimize Python code for better performance.

#Prompt:

Optimize the following Python code for better performance while maintaining the same functionality. Use efficient Python techniques.

Code:

numbers = []

for i in range(1, 500000):

```
    numbers.append(i * i)
print(len(numbers))
```

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.

#Output

```
def generate_squares(start, end):
    """Generate squared numbers without storing them."""
    for i in range(start, end):
        yield i * i


# Count squares without storing all values
count = sum(1 for i in range(1, 500000))
print(count)

# Output: 499999
# Explanation: Counts without creating list in memory
# Speed: Fastest for counting
# Memory: Minimal - only holds one value at a time
```