Lab Assignment 10.2

Program Name: B. Tech
Assignment Type: Lab
Academic Year: 2025-2026
Course Code: 23CS002PC304
Course Title: AI Assisted Coding
Year/Sem: III/II
Regulation: R23

Student Name: Ravula seetharam reddy
Roll No: 2303A52440
Assignment Number: 10.1 / 24
Day: Tuesday

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 1 – Error Detection and Correction

Prompt Used

Analyze the Python code and fix all syntax and logical errors. Provide corrected executable code and explanation.

Given Code

```
def calculate_total(nums)
sum = 0
```

```
for n in nums
sum += n
return total
```

AI Corrected Code

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

numbers = [1,2,3,4,5]
print(calculate_total(numbers))
```

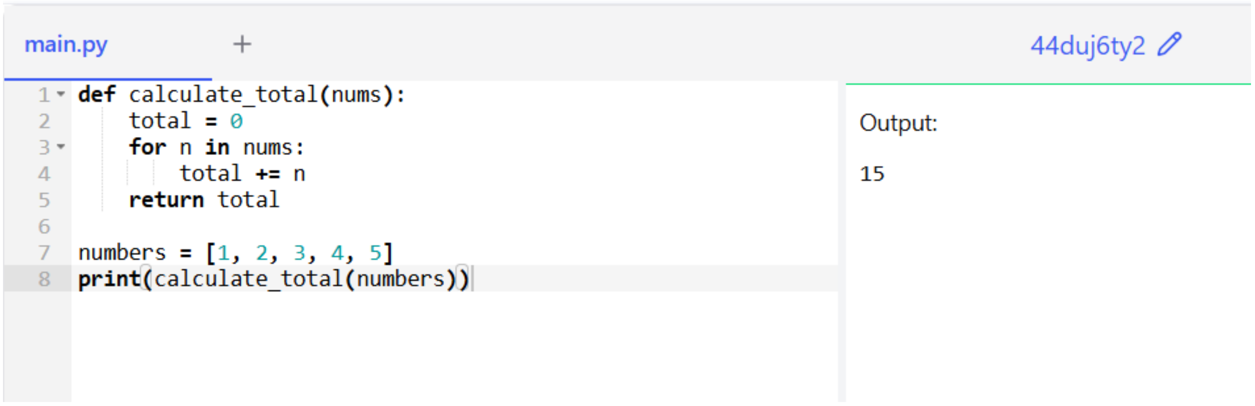
Explanation

- Missing colon after function and for loop fixed
- Indentation corrected
- Variable name changed from sum (reserved name) to total
- Returned variable corrected

Output

15

SCREENSHOTS:



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

numbers = [1, 2, 3, 4, 5]
print(calculate_total(numbers))
```

```
Output:

15
```

Task 2 – Code Style Standardization

Prompt Used

Refactor the code to follow PEP 8 coding standards with proper formatting.

Given Code

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

Refactored Code (PEP 8)

```
def find_sum(a, b):
    return a + b

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

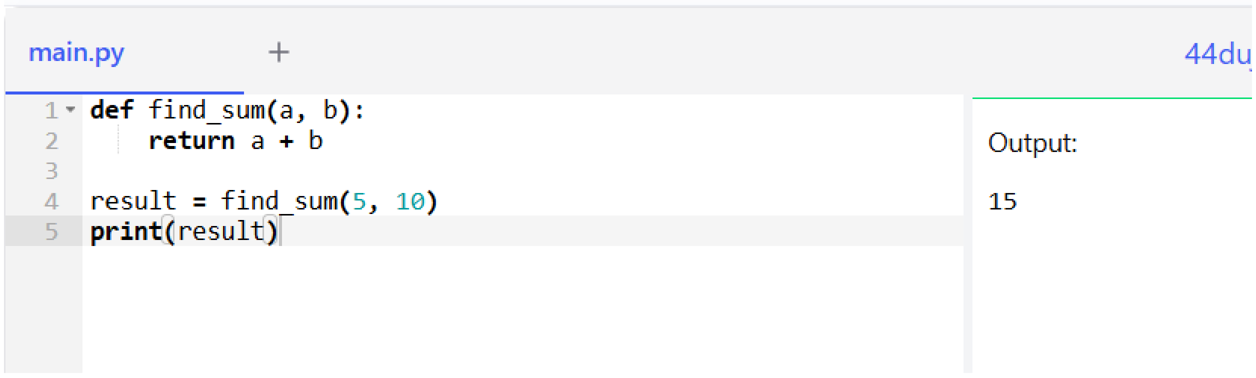
Explanation

- Function renamed using snake_case
- Proper spacing added
- Stored output in variable for readability

Output

15

SCREENSHOTS:



```
def find_sum(a, b):
    return a + b

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

Output:

```
15
```

Task 3 – Code Clarity Improvement

Prompt Used

Improve readability using meaningful variable and function names without changing functionality.

Given Code

```
def f(x,y):
    return x-y*2
print(f(10,3))
```

Improved Readable Code

```
def subtract_double(value, multiplier):
    return value - multiplier * 2

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

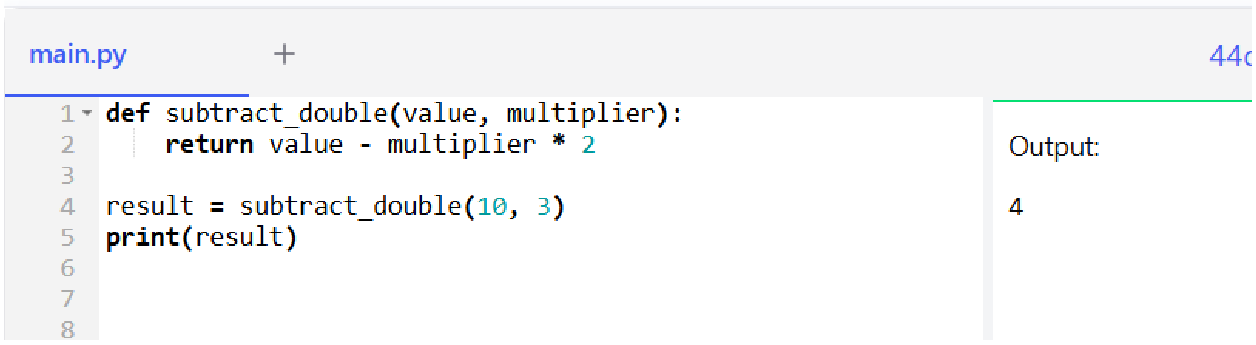
Explanation

- Function name made descriptive
- Variable names improved for clarity
- Output stored in variable

Output

4

SCREENSHOTS:



```
def subtract_double(value, multiplier):
    return value - multiplier * 2

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

```
Output:

4
```

Task 4 – Structural Refactoring

Prompt Used

Refactor repetitive code using reusable functions.

Given Code

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

Modular Code

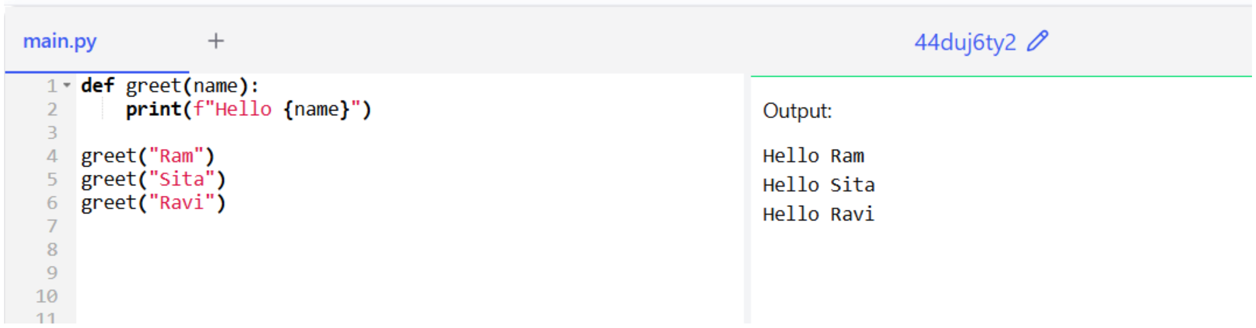
```
def greet(name):
    print(f"Hello {name}")

greet("Ram")
greet("Sita")
greet("Ravi")
```

Explanation

- Created reusable function greet()
- Eliminated repeated print statements

Output

Hello Ram Hello Sita Hello Ravi



main.py +

44duj6ty2

```
def greet(name):
    print(f"Hello {name}")

greet("Ram")
greet("Sita")
greet("Ravi")
```

Output:

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

Task 5 – Efficiency Enhancement

Prompt Used

Optimize the Python code for better performance.

Given Code

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

Optimized Code

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

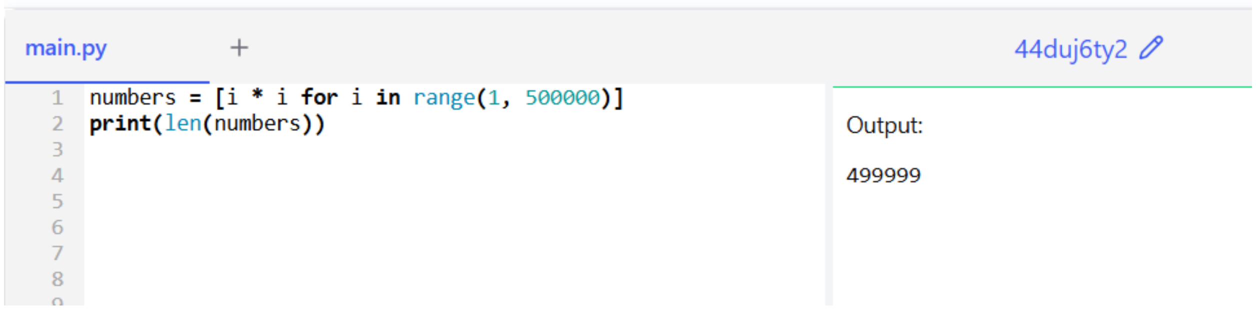
Explanation

- Replaced loop with list comprehension
- Faster execution and cleaner code

Output

499999

SCREENSHOTS:



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

```
Output:

499999
```

Conclusion

AI tools helped identify syntax errors, improve readability, enforce coding standards and optimize performance. The experiment demonstrated how prompt engineering can enhance code quality and maintainability effectively.