

## 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:** Sanjay Karupothula

**Roll No:** 2303A52337

**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:



The screenshot shows a code editor interface. On the left, a file named 'main.py' is open. The code is as follows:

```
1 def calculate_total(nums):
2     total = 0
3     for n in nums:
4         total += n
5     return total
6
7 numbers = [1, 2, 3, 4, 5]
8 print(calculate_total(numbers))
```

On the right side of the editor, the output is displayed:

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)
```

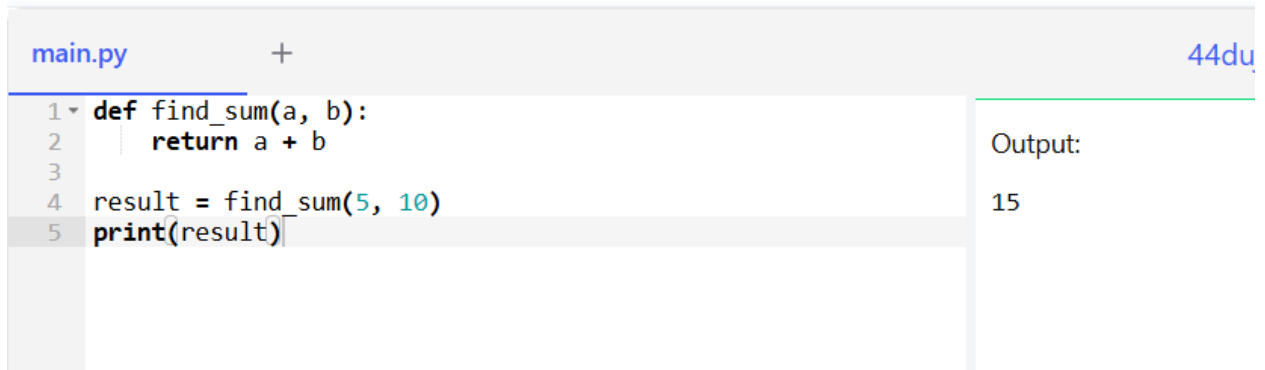
## Explanation

- Function renamed using snake\_case
- Proper spacing added
- Stored output in variable for readability

## Output

15

## SCREENSHOTS:



The screenshot shows a code editor window titled 'main.py' with a '+' icon and a '44du' logo in the top right corner. The code is as follows:

```
1 def find_sum(a, b):  
2     return a + b  
3  
4 result = find_sum(5, 10)  
5 print(result)
```

To the right of the code editor, the output is displayed: '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)
```

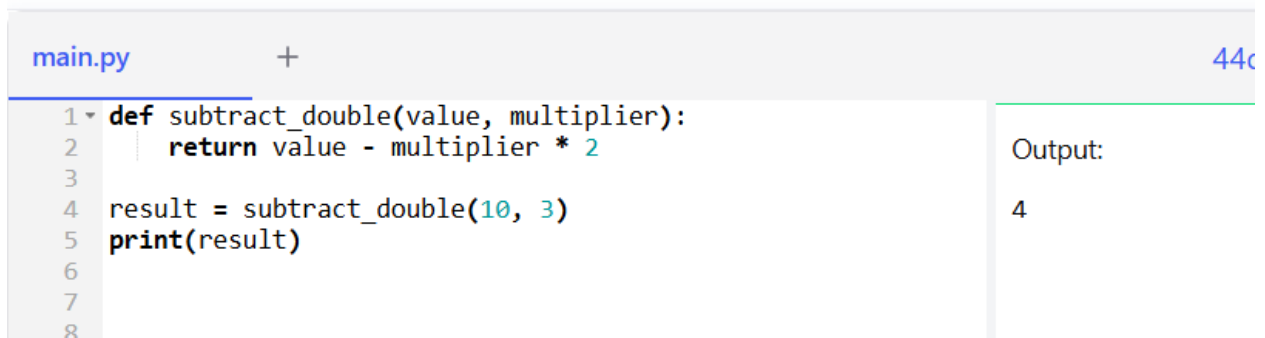
## Explanation

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

## Output

4

### SCREENSHOTS:



The screenshot shows a code editor with a file named 'main.py'. The code is as follows:

```
1 def subtract_double(value, multiplier):  
2     return value - multiplier * 2  
3  
4 result = subtract_double(10, 3)  
5 print(result)  
6  
7  
8
```

To the right of the code editor, the output is displayed:

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 <a href="#">🔗</a>
<pre>1 def greet(name): 2     print(f"Hello {name}") 3 4 greet("Ram") 5 greet("Sita") 6 greet("Ravi") 7 8 9 10 11</pre>		<p>Output:</p> <pre>Hello Ram Hello Sita Hello Ravi</pre>

---

## 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))
```

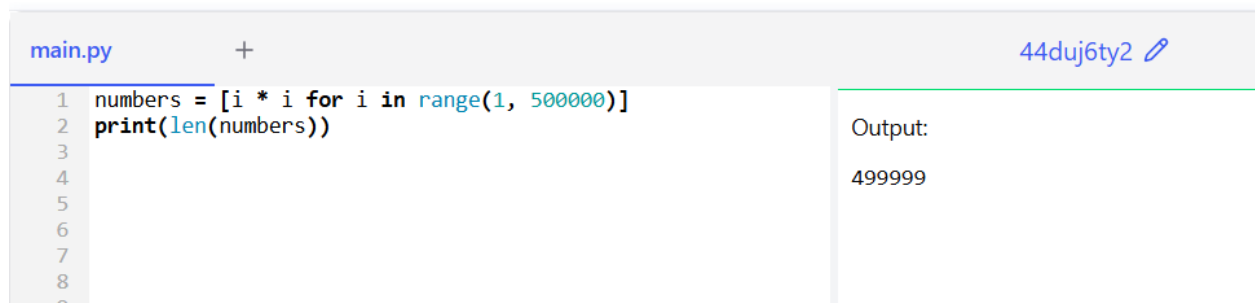
## Explanation

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

## Output

499999

## SCREENSHOTS:



The screenshot shows a code editor window with a file named 'main.py'. The code contains two lines: a list comprehension to generate a list of squares from 1 to 500,000, and a print statement to output the length of this list. The output shown on the right is 499,999. The editor interface includes a tab for 'main.py', a '+' icon for additional files, and a user identifier '44duj6ty2' with an edit icon.

```
1 numbers = [i * i for i in range(1, 500000)]
2 print(len(numbers))
3
4
5
6
7
8
9
```

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.