

# Assignment-10.3

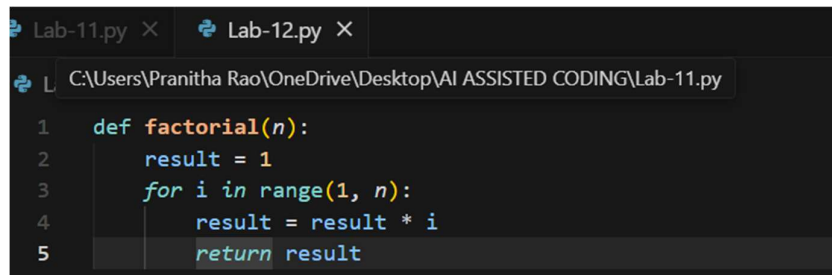
Gujja Pranitha

2303A52171

Batch 41

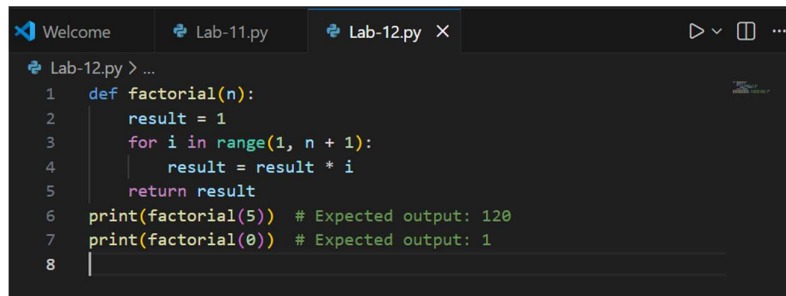
Task-1: AI-Assisted Bug Detection

Provided Code:

A screenshot of a code editor with two tabs: 'Lab-11.py' and 'Lab-12.py'. The active tab is 'Lab-11.py', showing a Python function 'factorial(n)'. The function initializes 'result = 1' and enters a 'for' loop with 'range(1, n)'. Inside the loop, it multiplies 'result' by 'i'. The loop ends with a 'return result' statement. The code is as follows:

```
1 def factorial(n):
2     result = 1
3     for i in range(1, n):
4         result = result * i
5     return result
```

Fix Code:

A screenshot of a code editor with three tabs: 'Welcome', 'Lab-11.py', and 'Lab-12.py'. The active tab is 'Lab-12.py', showing the corrected Python function 'factorial(n)'. The function initializes 'result = 1' and enters a 'for' loop with 'range(1, n + 1)'. Inside the loop, it multiplies 'result' by 'i'. The loop ends with a 'return result' statement. Below the function, there are two test cases: 'print(factorial(5))' with a comment '# Expected output: 120' and 'print(factorial(0))' with a comment '# Expected output: 1'. The code is as follows:

```
1 def factorial(n):
2     result = 1
3     for i in range(1, n + 1):
4         result = result * i
5     return result
6 print(factorial(5)) # Expected output: 120
7 print(factorial(0)) # Expected output: 1
8
```

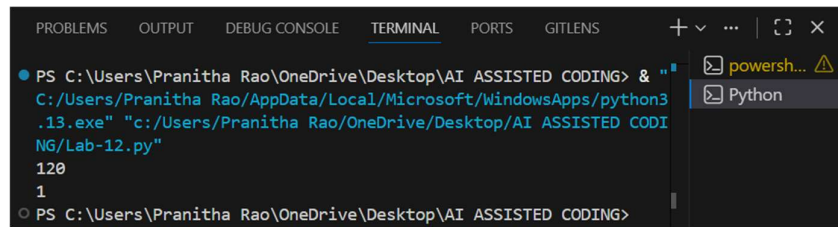
Logical bug in code:

1. return inside loop: Exits after first iteration, always returning 1 instead of computing the full factorial.
2. Wrong range: range(1, n) excludes n itself, so it doesn't multiply the last number needed for the factorial.

Explanation:

The factorial function calculates the product of all positive integers up to a given number n. It initializes a variable result to 1, then iterates from 1 to n (inclusive), multiplying result by each integer i. Finally, it returns the computed factorial value. The test cases demonstrate that factorial(5) correctly returns 120, and factorial(0) returns 1, which is the defined value for 0!.

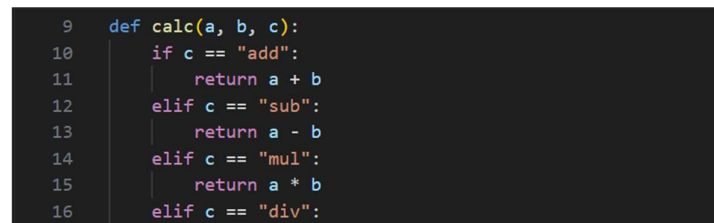
## OUTPUT:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-12.py"
120
1
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

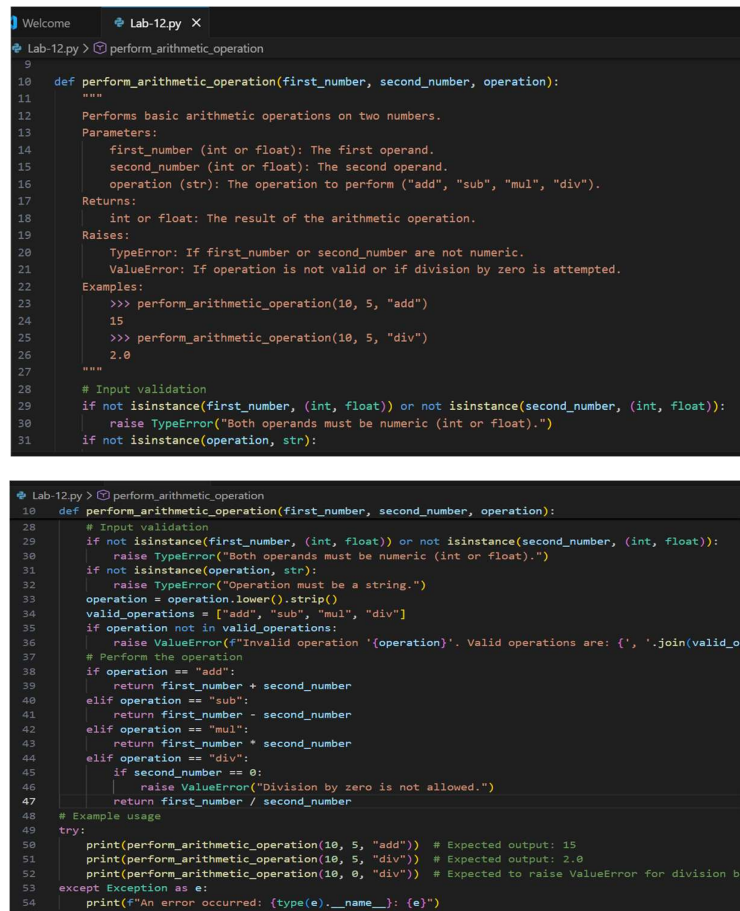
## Task-2: Improving Readability & Documentation

### Provided Code:



```
9 def calc(a, b, c):
10     if c == "add":
11         return a + b
12     elif c == "sub":
13         return a - b
14     elif c == "mul":
15         return a * b
16     elif c == "div":
```

### Fix Code:



```
10 def perform_arithmetic_operation(first_number, second_number, operation):
11     """
12     Performs basic arithmetic operations on two numbers.
13     Parameters:
14         first_number (int or float): The first operand.
15         second_number (int or float): The second operand.
16         operation (str): The operation to perform ("add", "sub", "mul", "div").
17     Returns:
18         int or float: The result of the arithmetic operation.
19     Raises:
20         TypeError: If first_number or second_number are not numeric.
21         ValueError: If operation is not valid or if division by zero is attempted.
22     Examples:
23         >>> perform_arithmetic_operation(10, 5, "add")
24         15
25         >>> perform_arithmetic_operation(10, 5, "div")
26         2.0
27     """
28     # Input validation
29     if not isinstance(first_number, (int, float)) or not isinstance(second_number, (int, float)):
30         raise TypeError("Both operands must be numeric (int or float).")
31     if not isinstance(operation, str):
32         raise TypeError("Operation must be a string.")
33     operation = operation.lower().strip()
34     valid_operations = ["add", "sub", "mul", "div"]
35     if operation not in valid_operations:
36         raise ValueError(f"Invalid operation '{operation}'. Valid operations are: {', '.join(valid_operations)}")
37     # Perform the operation
38     if operation == "add":
39         return first_number + second_number
40     elif operation == "sub":
41         return first_number - second_number
42     elif operation == "mul":
43         return first_number * second_number
44     elif operation == "div":
45         if second_number == 0:
46             raise ValueError("Division by zero is not allowed.")
47         return first_number / second_number
48     # Example usage
49     try:
50         print(perform_arithmetic_operation(10, 5, "add")) # Expected output: 15
51         print(perform_arithmetic_operation(10, 5, "div")) # Expected output: 2.0
52         print(perform_arithmetic_operation(10, 0, "div")) # Expected to raise ValueError for division by
53     except Exception as e:
54         print(f"An error occurred: (type(e).__name__): {e}")
```

## Explanation:

The function `perform\_arithmetic\_operation` takes two numbers and an operation as input. It first validates the inputs to ensure they are of the correct type and that the operation is valid. If the inputs are valid, it performs the specified arithmetic operation and returns the result. If any validation fails, it raises appropriate exceptions with descriptive error messages.

## OUTPUT:

```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-12.py"
15
2.0
An error occurred: ValueError: Division by zero is not allowed.
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

## Task-3: Enforcing Coding Standards

### Provided Code:

```
47
48 def Checkprime(n):
49     for i in range(2, n):
50         if n % i == 0:
51             return False
52     return True
```

### Fix Code:

```
Welcome  Lab-12.py X
Lab-12.py > check_prime
48 def check_prime(n):
49     """
50     Checks if a number is prime.
51     Parameters:
52     n (int): The number to check.
53     Returns:
54     bool: True if the number is prime, False otherwise.
55     """
56     if n < 2:
57         return False
58
59     for i in range(2, n):
60         if n % i == 0:
61             return False
62     return True
63 # Example usage
64 try:
65     print(check_prime(7)) # Expected output: True
66     print(check_prime(10)) # Expected output: False
67     print(check_prime(-5)) # Expected output: False
68 except Exception as e:
69     print(f"An error occurred: {type(e).__name__}: {e}")
70
```

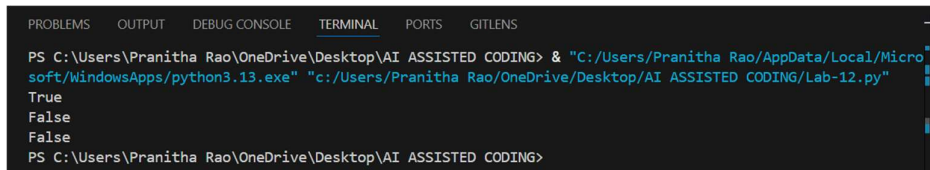
## AI Generated list of PEP8 Violations:

- Missing two blank lines between the import math and the top-level function definition (should be two blank lines before def is\_prime).
- Top-level executable code (the print(...) examples) is not protected by if \_\_name\_\_ == "\_\_main\_\_": (recommended for modules).
- Inline comments spacing inconsistent: some prints use one space before # (e.g. print(is\_prime(17)) # True, print(is\_prime(18)) # False) — PEP8 recommends two spaces before inline comments.
- Excessive blank lines at the end of the file (many consecutive empty lines); reduce to a single final newline.
- Minor: the exception message in TypeError("number must be an integer") is sentence-style; consider starting with a capital letter for consistency (not strictly enforced by PEP8).

## Explanation:

The check\_prime function checks if a number n is prime by first checking if it is less than 2 (since prime numbers are defined as greater than 1). If n is less than 2, it returns False. Then, it iterates from 2 to n-1 and checks if n is divisible by any of those numbers. If it finds any divisor, it returns False, indicating that n is not prime. If it completes the loop without finding any divisors, it returns True, indicating that n is prime.

## OUTPUT:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-12.py"
True
False
False
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

```
# A PEP8-compliant version of the function,
import math
def is_prime(number):
    if not isinstance(number, int):
        raise TypeError("number must be an integer")
    if number < 2:
        return False
    if number % 2 == 0:
        return number == 2
    limit = math.isqrt(number) + 1
    for i in range(3, limit, 2):
        if number % i == 0:
            return False
    return True
```

## Task-4: AI as a Code Reviewer in Real Projects

### Provided Code:

```
Welcome  Lab-12.py X
Lab-12.py > processData
67 def processData(d):
68     return [x * 2 for x in d if x % 2 == 0]
```

### Fix Code:

```
Welcome  Lab-12.py X
Lab-12.py > filter_and_transform_numbers > predicate
68 from typing import Callable
69 def filter_and_transform_numbers(
70     numbers: list[int | float],
71     predicate: Callable[[int | float], bool] | None = None,
72     multiplier: int | float = 2
73 ) -> list[int | float]:
74     """
75     Filters numbers based on a predicate and applies a multiplier transformation.
76
77     Args:
78         numbers: A list of numeric values to process.
79         predicate: A filtering function that returns True for values to include.
80                   Defaults to filtering even numbers if None.
81         multiplier: The factor to multiply filtered numbers by (default: 2).
82
83     Returns:
84         A list of transformed numbers that satisfy the predicate condition.
85
86     Raises:
87         TypeError: If numbers is not a list or contains non-numeric values.
88         ValueError: If numbers is empty.
89
90     Examples:
91         >>> filter_and_transform_numbers([1, 2, 3, 4])
92         [4, 8]
93         >>> filter_and_transform_numbers([1, 2, 3, 4], lambda x: x > 2, 3)
94         [9, 12]
95     """
```

```
Welcome  Lab-12.py X
Lab-12.py > filter_and_transform_numbers > predicate
69 def filter_and_transform_numbers(
96     if not isinstance(numbers, (list, tuple)):
97         raise TypeError(f"Expected list or tuple, got {type(numbers).__name__}")
98
99     if not numbers:
100         raise ValueError("List cannot be empty")
101
102     if not all(isinstance(x, (int, float)) for x in numbers):
103         raise TypeError("All elements must be numeric (int or float)")
104
105     if predicate is None:
106         predicate = lambda x: x % 2 == 0
107
108     return [x * multiplier for x in numbers if predicate(x)]
109 # Example usage
110 try:
111     print(filter_and_transform_numbers([1, 2, 3, 4])) # Expected output: [4, 8]
112     print(filter_and_transform_numbers([1, 2, 3, 4], lambda x: x > 2, 3)) # Expected output: [9,
113     print(filter_and_transform_numbers([], lambda x: x > 2, 3)) # Expected to raise ValueError fo
114 except Exception as e:
115     print(f"An error occurred: {type(e).__name__}: {e}")
116
```

### Explanation:

The function `filter_and_transform_numbers` takes a list of numbers and applies a filtering predicate and a multiplier transformation. It includes input validation to ensure that the input is a list of numeric values and that it is not empty. If the predicate is not provided, it defaults to filtering even numbers. The function returns a new list of transformed numbers that satisfy the predicate condition. The example usage demonstrates how to use the function and handles potential exceptions gracefully.

## OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> ^C
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/AppData/Local/Micro
soft/WindowsApps/python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-12.py"
[4, 8]
[9, 12]
An error occurred: ValueError: List cannot be empty
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
Ln 115, Col 57 Spaces: 4 UTF-8 CRLF {} Python 3.13.12 (Microsoft Store)
```

## Task-5: AI-Assisted Performance Optimization

### Provided Code:

```
Welcome Lab-12.py X
Lab-12.py > sum_of_squares
103 def sum_of_squares(numbers):
104     total = 0
105     for num in numbers:
106         total += num ** 2
107     return total
```

### Fix code:

```
Welcome Lab-12.py X
Lab-12.py > ...
105 def sum_of_squares(numbers: list[int | float]) -> int | float:
106     """
107     Calculates the sum of squares for a list of numbers.
108     Args:
109         numbers: A list of numeric values to square and sum.
110     Returns:
111         The sum of all squared numbers.
112     Raises:
113         TypeError: If numbers is not a list or contains non-numeric values.
114     Examples:
115         >>> sum_of_squares([1, 2, 3])
116         14
117         >>> sum_of_squares([2, 4])
118         20
119     """
120     if not isinstance(numbers, (list, tuple)):
121         raise TypeError(f"Expected list or tuple, got {type(numbers).__name__}")
122
123     if not all(isinstance(x, (int, float)) for x in numbers):
124         raise TypeError("All elements must be numeric (int or float)")
125     total = 0
126     for num in numbers:
127         total += num ** 2
128     return total
129
130 # Example usage
131 try:
132     print(sum_of_squares([1, 2, 3])) # Expected output: 14
133     print(sum_of_squares([2, 4])) # Expected output: 20
134     print(sum_of_squares("not a list")) # Expected to raise TypeError for invalid input type
135 except Exception as e:
136     print(f"An error occurred: {type(e).__name__}: {e}")
137
138 # PERFORMANCE OPTIMIZATION ANALYSIS
139 """
140 """
```



```
Welcome  Lab-12.py X
Lab-12.py > ...
139 import time
140 def sum_of_squares_optimized_v1(numbers: list[int | float]) -> int | float:
141     """
142     Optimized version using built-in sum() with generator expression.
143
144     Time Complexity: O(n)
145     Space Complexity: O(1)
146     """
147     return sum(x * x for x in numbers)
148 def test_sum_of_squares():
149     """
150     Comprehensive test suite for sum_of_squares() function.
151     Tests functionality, edge cases, and performance.
152     """
153     print("\n" + "=" * 67)
154     print("TESTING sum_of_squares() FUNCTION")
155     print("=" * 67)
156
157     # Test 1: Small list
158     print("\n[Test 1] Small list [1, 2, 3, 4]")
159     test_data_1 = [1, 2, 3, 4]
160     result_1 = sum_of_squares(test_data_1)
161     expected_1 = 30 # 1^2 + 2^2 + 3^2 + 4^2 = 1 + 4 + 9 + 16 = 30
162     status_1 = "✓ PASSED" if result_1 == expected_1 else "X FAILED"
163     print(f"Result: {result_1}, Expected: {expected_1}")
164     print(status_1)
165
166     # Test 2: Range object
167     print("\n[Test 2] Range object range(1, 101)")
168     start_2 = time.perf_counter()
169     test_data_2 = range(1, 101)
170     result_2 = sum_of_squares(list(test_data_2))
171     time_2 = time.perf_counter() - start_2
172     print(f"Sum of squares from 1 to 100: {result_2}")
173     print("✓ PASSED")
```

```
Welcome  Lab-12.py X
Lab-12.py > ...
148 def test_sum_of_squares():
181     # Test 4: Large range (performance test)
182     print("\n[Test 4] Large range - range(1, 1000001)")
183     start_4 = time.perf_counter()
184     test_data_4 = list(range(1, 1000001))
185     result_4 = sum_of_squares(test_data_4)
186     time_4 = time.perf_counter() - start_4
187     print(f"Sum of squares from 1 to 1,000,000: {result_4}")
188     print(f"Time taken: {time_4:.4f} seconds")
189     print("✓ PASSED")
190
191     # Test 5: Performance comparison
192     print("\n" + "=" * 67)
193     print("PERFORMANCE COMPARISON")
194     print("=" * 67)
195     test_data = list(range(1, 1000001))
196
197     # Original implementation
198     start = time.perf_counter()
199     result_original = sum_of_squares(test_data)
200     time_original = time.perf_counter() - start
201
202     # Optimized version
203     start = time.perf_counter()
204     result_optimized = sum_of_squares_optimized_v1(test_data)
205     time_optimized = time.perf_counter() - start
206     speedup = time_original / time_optimized
207
208     print(f"\nOriginal (for loop):      {time_original:.6f}s")
209     print(f"Optimized (generator):      {time_optimized:.6f}s")
210     print(f"Speedup:                        {speedup:.2f}x faster")
211     print(f"Result Match:                  {result_original == result_optimized}")
212
213     print("\n" + "=" * 67)
214     print("ALL TESTS COMPLETED SUCCESSFULLY!")
215     print("=" * 67 + "\n")
216
217 if __name__ == "__main__":
218     test_sum_of_squares()
```

## Explanation:

The `sum_of_squares()` function calculates the sum of squares for a list of numbers. It includes input validation to ensure that the input is a list or tuple and that all elements are numeric. The function iterates through each number, squares it, and adds it to a total, which is returned at the end. The `sum_of_squares_optimized_v1()` function is an optimized version that uses a generator expression with the built-in `sum()` function, which can be more efficient for large lists. The `test_sum_of_squares()` function runs a series of tests to validate the correctness and performance of the `sum_of_squares()` function. It tests small lists, range objects, float numbers, and a large range to ensure

that the function works correctly and efficiently. The performance comparison shows the time taken by both the original and optimized versions, demonstrating the speedup achieved by the optimization.

## OUTPUT:

```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-12.py"
14
20
An error occurred: TypeError: Expected list or tuple, got str

=====
TESTING sum_of_squares() FUNCTION
=====

[Test 1] Small list [1, 2, 3, 4]
Result: 30, Expected: 30
✓ PASSED

[Test 2] Range object range(1, 101)
Sum of squares from 1 to 100: 338350
✓ PASSED

[Test 3] Float numbers [1.5, 2.5, 3.5]
Result: 20.75, Expected: 20.75
✓ PASSED

[Test 4] Large range - range(1, 1000001)
Sum of squares from 1 to 1,000,000: 333333833333350000
Time taken: 0.1522 seconds
✓ PASSED

=====
PERFORMANCE COMPARISON
=====
```

```
✓ PASSED

=====
PERFORMANCE COMPARISON
=====

Original (for loop):      0.093711s
Optimized (generator):   0.052264s
Speedup:                 1.79x faster
Result Match:            True

=====
ALL TESTS COMPLETED SUCCESSFULLY!
=====

PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```