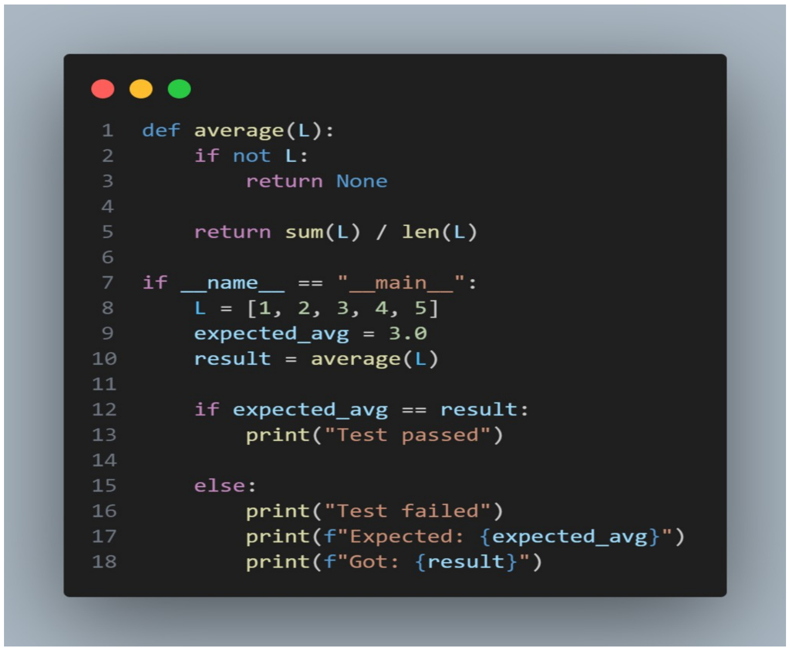
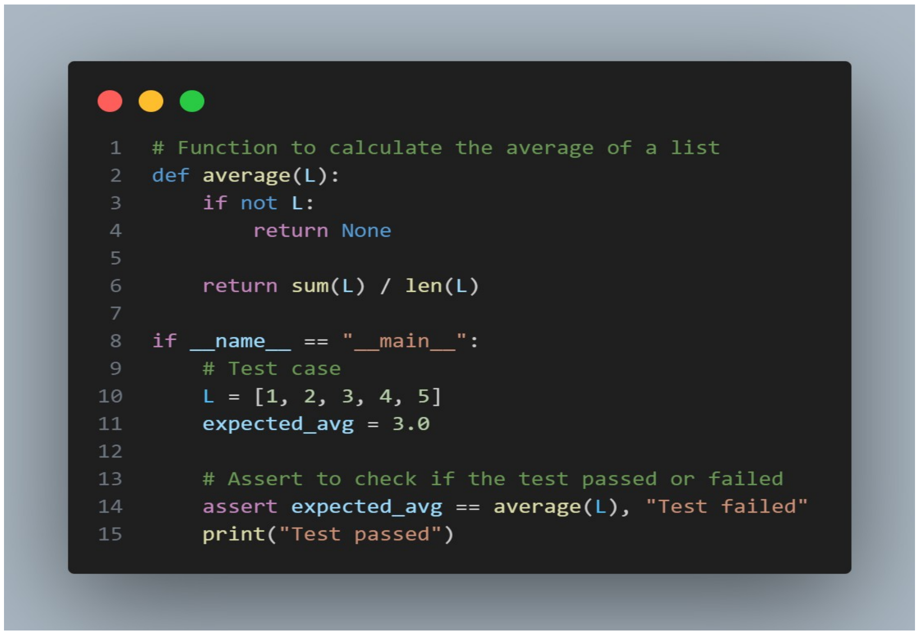
Code Testing: Functions, Assertions, and Unit Testing

1. Functions
Functions play a central role in organizing and testing code. By breaking complex problems into smaller, reusable blocks, they simplify debugging and testing.



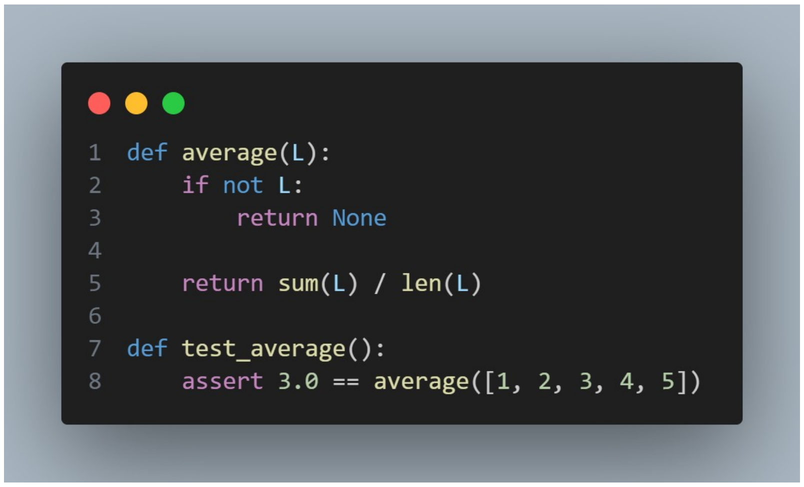
```python
def average(L):
    if not L:
        return None

    return sum(L) / len(L)

if __name__ == "__main__":
    L = [1, 2, 3, 4, 5]
    expected_avg = 3.0
    result = average(L)

    if expected_avg == result:
        print("Test passed")

    else:
        print("Test failed")
        print(f"Expected: {expected_avg}")
        print(f"Got: {result}")
```

2. Assertions
Assertions are used to verify that a piece of code behaves as expected. An assertion is a statement that checks a condition, and if the condition evaluates to `False`, the program halts with an error. This helps catch bugs early.



```python
# Function to calculate the average of a list
def average(L):
    if not L:
        return None

    return sum(L) / len(L)

if __name__ == "__main__":
    # Test case
    L = [1, 2, 3, 4, 5]
    expected_avg = 3.0

    # Assert to check if the test passed or failed
    assert expected_avg == average(L), "Test failed"
    print("Test passed")
```

3. Unit Testing with Pytest
Pytest is a popular Python testing framework that makes it easy to write simple and scalable test cases. Its minimal syntax and powerful features make it a preferred choice for many developers.



```python
def average(L):
    if not L:
        return None

    return sum(L) / len(L)

def test_average():
    assert 3.0 == average([1, 2, 3, 4, 5])
```

4. Table-Driven Testing
Table-driven testing is a method that uses tables (often as lists or arrays in code) to store test cases and their expected outcomes. This approach reduces repetition and ensures consistency, making it easier to add or modify test cases.