Stats Calculator Kata

TDD Journey Summary

Python • pytest • Test-Driven Development

10 Tests • 100% Coverage • 11 Lines of Code

Overview

Kata: Stats Calculator

Language: Python 3.10+

Testing Framework: pytest

Final Results: 10 tests passing, 100% code coverage

Implementation: 11 lines total (4 executable statements)

Problem Statement

Create a function that calculates statistics (min, max, count, average) for a sequence of integers.

Function signature: calculate_stats(numbers) → dict

TDD Journey - Four Phases

Phase A: Empty/Single Element (Base Cases)

Test 1: Empty sequence - Hardcoded return value
Test 2: Single element [5] - Added conditional logic

Key Learning: Start with simplest cases. Fake it till you make it!

Phase B: Simple Sequences (2-3 elements)

Test 3: Two elements [1, 2] - THE BREAKTHROUGH MOMENT
This test forced the real implementation using Python built-ins:
• min(numbers) - smallest value
• max(numbers) - largest value
• len(numbers) - count
• sum(numbers) / len(numbers) - average

Tests 4-5: Three elements (ordered and unordered) - Triangulation
Both passed WITHOUT code changes!

Key Learning: Let tests force the general solution. Triangulation validates it.

Phase C: Edge Cases (Negatives and Duplicates)

Tests 6-9: All edge case tests passed WITHOUT code changes!
• Negative numbers: [-5, -1, -3]
• Mixed positive/negative: [-2, 0, 2]

• All duplicates: [2, 2, 2]
• Duplicates at extremes: [1, 2, 1]

Key Learning: Python's built-in functions handle edge cases automatically!

Phase D: Larger Sequences

Test 10: Larger sequence [1..10] - Passed WITHOUT code changes!
Validates that algorithm scales correctly.

Key Learning: Algorithm works for any size input.

Final Implementation

```
def calculate_stats(numbers):
    # Handle empty sequence
    if not numbers:
        return {'min': None, 'max': None, 'count': 0, 'average': None}

    # For non-empty sequences, calculate actual statistics
    return {
        'min': min(numbers),
        'max': max(numbers),
        'count': len(numbers),
        'average': sum(numbers) / len(numbers)
    }
```

Test Results

■ 10 tests passing

■ 100% code coverage

■ 4 executable statements

■ O(n) time complexity, O(1) space complexity

Key TDD Principles Demonstrated

• **Red-Green-Refactor:** Followed for each test

• **Fake It Till You Make It:** Started with hardcoded values

• **Triangulation:** Multiple tests validated general solution

• **Let Algorithm Emerge:** Didn't design upfront

• **Small Steps:** One test at a time

• **Edge Cases Validate Design:** All passed without changes

Key Learnings

• Start simple and let algorithm emerge from tests

• Python built-ins handle edge cases automatically

• Simple solution is often the best solution

• TDD leads to minimal, correct code

• 100% coverage achievable with thoughtful tests

Generated with Claude Code • https://claude.com/claude-code