Summary

Unit testing is a powerful concept which, if properly implemented, can both reduce maintenance costs and increase flexibility in any long-term project. It is also important to understand that unit testing is not a panacea, a Magic Problem Solver, or a silver bullet. Writing good test cases is hard, and keeping them up to date takes discipline (especially when customers are screaming for critical bug fixes). Unit testing is not a replacement for other forms of testing, including functional testing, integration testing, and user acceptance testing. But it is feasible, and it does work, and once you've seen it work, you'll wonder how you ever got along without it.

These few chapters have covered a lot of ground, and much of it wasn't even Python-specific. There are unit testing frameworks for many languages, all of which require you to understand the same basic concepts:

- Designing test cases that are specific, automated, and independent
- Writing test cases *before* the code they are testing
- Writing tests that test good input and check for proper results
- Writing tests that test bad input and check for proper failure responses
- Writing and updating test cases to reflect new requirements
- Refactoring mercilessly to improve performance, scalability, readability, maintainability, or whatever other -ility you're lacking