26. Development Tools

The modules described in this chapter help you write software. For example, the `pydoc` module takes a module and generates documentation based on the module's contents. The `doctest` and `unittest` modules contains frameworks for writing unit tests that automatically exercise code and verify that the expected output is produced. **2to3** can translate Python 2.x source code into valid Python 3.x code.

The list of modules described in this chapter is:

- 26.1. `typing` — Support for type hints
  - 26.1.1. Type aliases
  - 26.1.2. NewType
  - 26.1.3. Callable
  - 26.1.4. Generics
  - 26.1.5. User-defined generic types
  - 26.1.6. The `Any` type
  - 26.1.7. Classes, functions, and decorators
- 26.2. `pydoc` — Documentation generator and online help system
- 26.3. `doctest` — Test interactive Python examples
  - 26.3.1. Simple Usage: Checking Examples in Docstrings
  - 26.3.2. Simple Usage: Checking Examples in a Text File
  - 26.3.3. How It Works
    - 26.3.3.1. Which Docstrings Are Examined?
    - 26.3.3.2. How are Docstring Examples Recognized?
    - 26.3.3.3. What's the Execution Context?
    - 26.3.3.4. What About Exceptions?
    - 26.3.3.5. Option Flags
    - 26.3.3.6. Directives
    - 26.3.3.7. Warnings
  - 26.3.4. Basic API
  - 26.3.5. Unittest API
  - 26.3.6. Advanced API
    - 26.3.6.1. DocTest Objects
    - 26.3.6.2. Example Objects
    - 26.3.6.3. DocTestFinder objects
    - 26.3.6.4. DocTestParser objects
    - 26.3.6.5. DocTestRunner objects
    - 26.3.6.6. OutputChecker objects
  - 26.3.7. Debugging
  - 26.3.8. Soapbox
- 26.4. `unittest` — Unit testing framework
  - 26.4.1. Basic example
  - 26.4.2. Command-Line Interface

        - 26.4.2.1. Command-line options
    - 26.4.3. Test Discovery
    - 26.4.4. Organizing test code
    - 26.4.5. Re-using old test code
    - 26.4.6. Skipping tests and expected failures
    - 26.4.7. Distinguishing test iterations using subtests
    - 26.4.8. Classes and functions
        - 26.4.8.1. Test cases
            - 26.4.8.1.1. Deprecated aliases
        - 26.4.8.2. Grouping tests
        - 26.4.8.3. Loading and running tests
            - 26.4.8.3.1. load_tests Protocol
    - 26.4.9. Class and Module Fixtures
        - 26.4.9.1. setUpClass and tearDownClass
        - 26.4.9.2. setUpModule and tearDownModule
    - 26.4.10. Signal Handling
- 26.5. `unittest.mock` — mock object library
    - 26.5.1. Quick Guide
    - 26.5.2. The Mock Class
        - 26.5.2.1. Calling
        - 26.5.2.2. Deleting Attributes
        - 26.5.2.3. Mock names and the name attribute
        - 26.5.2.4. Attaching Mocks as Attributes
    - 26.5.3. The patchers
        - 26.5.3.1. patch
        - 26.5.3.2. patch.object
        - 26.5.3.3. patch.dict
        - 26.5.3.4. patch.multiple
        - 26.5.3.5. patch methods: start and stop
        - 26.5.3.6. patch builtins
        - 26.5.3.7. TEST_PREFIX
        - 26.5.3.8. Nesting Patch Decorators
        - 26.5.3.9. Where to patch
        - 26.5.3.10. Patching Descriptors and Proxy Objects
    - 26.5.4. MagicMock and magic method support
        - 26.5.4.1. Mocking Magic Methods
        - 26.5.4.2. Magic Mock
    - 26.5.5. Helpers
        - 26.5.5.1. sentinel
        - 26.5.5.2. DEFAULT
        - 26.5.5.3. call
        - 26.5.5.4. create_autospec
        - 26.5.5.5. ANY
        - 26.5.5.6. FILTER_DIR
        - 26.5.5.7. mock_open

        - 26.5.5.8. Autospeccing
- 26.6. `unittest.mock` — getting started
    - 26.6.1. Using Mock
        - 26.6.1.1. Mock Patching Methods
        - 26.6.1.2. Mock for Method Calls on an Object
        - 26.6.1.3. Mocking Classes
        - 26.6.1.4. Naming your mocks
        - 26.6.1.5. Tracking all Calls
        - 26.6.1.6. Setting Return Values and Attributes
        - 26.6.1.7. Raising exceptions with mocks
        - 26.6.1.8. Side effect functions and iterables
        - 26.6.1.9. Creating a Mock from an Existing Object
    - 26.6.2. Patch Decorators
    - 26.6.3. Further Examples
        - 26.6.3.1. Mocking chained calls
        - 26.6.3.2. Partial mocking
        - 26.6.3.3. Mocking a Generator Method
        - 26.6.3.4. Applying the same patch to every test method
        - 26.6.3.5. Mocking Unbound Methods
        - 26.6.3.6. Checking multiple calls with mock
        - 26.6.3.7. Coping with mutable arguments
        - 26.6.3.8. Nesting Patches
        - 26.6.3.9. Mocking a dictionary with MagicMock
        - 26.6.3.10. Mock subclasses and their attributes
        - 26.6.3.11. Mocking imports with patch.dict
        - 26.6.3.12. Tracking order of calls and less verbose call assertions
        - 26.6.3.13. More complex argument matching
- 26.7. 2to3 - Automated Python 2 to 3 code translation
    - 26.7.1. Using 2to3
    - 26.7.2. Fixers
    - 26.7.3. `lib2to3` - 2to3's library
- 26.8. `test` — Regression tests package for Python
    - 26.8.1. Writing Unit Tests for the `test` package
    - 26.8.2. Running tests using the command-line interface
- 26.9. `test.support` — Utilities for the Python test suite