Unit testing

- determine the correctness of a single function isolated from a larger codebase
- if all the units of an application work as intended in isolation, then integrating them as intended should be much easier
- frameworks for automating these tests:
  - unittest: built-in standard library
  - pytest: complete framework
  - nose: an extension to unittest
  - hypothesis: a unit test-generation tool

unittest

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when separator not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

unittest: test discovery

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method

python -m unittest tests/test_something.py
```

Automatic discovery: all of the test files must be modules or packages (including namespace packages) importable from the top-level directory of the project

```
cd project_directory
python -m unittest [discover]
```

discover options:

- `-s, --start-directory directory` (. default)
- `-p, --pattern pattern` (test*.py default)
- `-t, --top-level-directory directory` (defaults to start directory)

Prepare and tidy up

```
import unittest

class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')

    def tearDown(self):
        self.widget.dispose()
```