Testing

Intermediate Application Development

Otago Polytechnic
Dunedin, New Zealand
Kaiako: Tom Clark

Introduction

Testing is important, but traditionally programmers aren't good at it. Modern programming practice puts more responsibility for testing, however. Thankfully (and probably as a result of this), tooling to automate tests is available.

Types of tests

- Unit tests
- Integration tests
- Acceptance tests

We will focus on unit tests, but much of this can also be applied to integration tests.

Automated testing

The only sensible way to handle unit tests is to automate them. This

- ensures that tests are performed;
- makes tests consistent;
- guides development;
- provides de facto documentation.

Components of Automated Tests

- Test cases
- Test fixtures
- Test suites
- Test runners

We will look at Python's `unittest` module and see how it provides these. There are other options, but `unittest` is provided in the standard library.

Example

```
class Multiplier:
    def __init__(self, factor):
        self.factor = factor

    def multiply(self, num):
        return self.factor * num
```

Let’s test this.

Example

```
import unittest

class TestMultiplier(unittest.TestCase):
    def setUp(self):
        self.m = Multiplier(2)

    def test_multiply(self):
        result = self.m.multiply(2)
        self.assertIsInstance(result, (int, float, complex))
        self.assertEqual(result, 4)
```

Testing Exceptions

```
import unittest

class TestMultiplier(unittest.TestCase):
    ...

    def test_multiply_raises(self):
        with self.assertRaises(TypeError):
            self.m.multiply(2)
```

Skipping Tests

Sometimes you want to skip a test, or you expect a test to fail. You can decorate a test with one of these.

```
@unittest.skip('message')

@unittest.skipIf(condition, 'message')

@unittest.expectedFailure
```

Skipping Tests

Sometimes you want to skip a test, or you expect a test to fail. You can decorate a test with one of these.

```
@unittest.skip('message')

@unittest.skipIf(condition, 'message')

@unittest.expectedFailure
```

Organising Tests

There is more than one approach to this, but here is a good overall strategy.

```
project_root/
     |
     |- multiplier.py
     | ...
     |- tests/
          |
          |- test_multiplier.py
          | ...
```

Organising Tests

Inside `test_multiplier.py`, we have

```
import unittest
from multiplier import Multiplier

class TestMultiplier(unittest.TestCase):
    ...
```

Running Tests

From our project root directory, we can use commands like

```
python -m unittest tests/test_multiplier -v
python -m unittest discover
```

Programming Activity

1. Pull the course materials repo.
2. Create a new branch, `18-practical` in your practicals repo.
3. Copy the subdirectory, `18-practical` from the class materials into your repo.
4. See the README for directions.
5. We will discuss results in 20ish minutes.

Mocking

Consider this class.

```
class UserManager:
    def get_user_name(self, user_id):
        user  = db.get_user(user_id)
        return user.name
```

This class is hard to test since it relies on an external resource, `db`.

Mocking

`unittest.mock` helps with this problem.

```
from unittest.mock import Mock

testuser = Mock()
testuser.name = 'Joe Bloggs'
db = Mock()
db.get_user.return_value = testuser

class UserManager:
    def get_user_name(self, user_id):
        user = db.get_user(user_id)
        return user.name
```

`Mock()` provides all-purpose stand in objects for use in testing and development.

Mocking in Unit Tests

We can also use mocks in unit tests

```
from unittest.mock import patch
import user_manager
class TestUserManager:

    ...

    @patch('user_manager.db')
    def test_get_user_name(self, mock_db):
        testuser = Mock()
        testuser.name = 'Joe Bloggs'
        mock_db.get_user.return_value = testuser
        assertEqual(self.usermanager.get_user_name(1),
            'Joe Bloggs')
```

The mock objects are used in the test without any modification to the `UserManager` code.

References

- unittest: `https://docs.python.org/3/library/unittest.html`
- unittest.mock: `https://docs.python.org/3/library/unittest.mock.html`
- RealPython article about mock: `https://realpython.com/python-mock-library/`