PyTest

PyGRN

Avantatges respecte unittest

- Codi més “pythonic”
  - Plain asserts vs self.assert*
- Llibreria vs framework
- Fixtures més potents
  - en unittest només pots tenir una per test
  - son més reutilizables que a unittest, p.ex. una fixture pot fer servir un altre fixture
  - parametrització de fixtures
- Tooling més potent
  - (marks, executar nomes per text match, etc)
  - Stop after N failures
  - Run on multiple cpus (pytest-xdist)

Simple test

```
class TestOperators(unittest.TestCase):
    def test_addition():
        self.assertEqual(operator.add(1, 4),  5)
```

VS:

```
def test_addition():
    assert operator.add(1, 4) == 5
```

Migrar de unittest a pytest

- pytest pot correr tests escrits en unittest i sense fer res ja estaras beneficiante de moltes de les millores del tooling
- Unittest2pytest (transforma els asserts de self.assert* a assert simple)

Fitxers configuració basics

pytest.ini:

- un per tot el projecte
- opcions de cli
- opcions de plugins pytest
- env vars per tests
- markers propis
- etc

conftest.py:

- pot haver varis (per mòdul, app, etc)
- fixtures compartides
- configuració programàtica de tests
- configuració programàtica del pytest

Markers (decoradors)

Integrats:

- Skip
- Skipif
- Xfail

Propis:

- exemples: smoke, unit, integració, funcional, internet...

Fixtures

- scope
  - function, class, module, package or session
- autouse
- multiple fixtures per test
- anidar fixtures
- parametrització fixtures

Fixtures built-in

- tmpdir
- capsys (stderr/stdout)
- mock/monkeypatch
- cache (usada internament per --last-failed i --failed-first)

Testing stdout/stderr

```
def test_my_function(capsys):
    my_function()  # function that prints stuff
    captured = capsys.readouterr()  # Capture output
    assert f"Received invalid message ..." in captured.out  # Test stdout
    assert f"Fatal error ..." in captured.err  # Test stderr
```

Patching amb pytest-mock

```
def test_my_func(mocker):
    mocked_method = mocker.patch('backend.programs.models.Podcast.cut_and_mark')
    mocked_method.return_value = None
    mocked_method.not_called()
```

Fixtures parametrize

```
@pytest.mark.parametrize("test_input,expected", [
        ("3+5", 8),
        ("2+4", 6),
        ("6*9", 42)])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

Tools extra / tips & tricks

- coverage / pytest-cov
- hypothesis
- env
- xdist
- timeout
- assert 2.2 == pytest.approx(2.3)
- with pytest.raises(...)
- sugar

Doc tests (--doctest-modules)

```
def gigabytes2bytes(size_gigabytes: int) -> int:
    """
    >>> gigabytes2bytes(30)
    32212254720
    """
    if size_gigabytes < 0:
        raise Exception("gigabytes should be positive")
    p = math.pow(1024, 3)
    s = round(size_gigabytes * p, 2)
    return int(s)
```

Running tests

Correr un fitxer:
$ pytest test_file.py

Filtrar per text en nom del test:
$ pytest -k with_pandas

Correr tests amb una marca específica:
$ pytest -m smoketest

Imprimer stdout i trace context:
$ pytest -s --showlocals

Aturar en primer error:
$ pytest -x

Executar darrers tests fallats:
$ pytest -lf

Executar primer els fallats:
$ pytest -ff

Integració amb django (pytest-django)

```
def test_foobar(client):
    assert client.get(‘/foobar’) == ‘foobar’

def test_foobar_admin(admin_client):
    assert admin_client.get(‘/foobar’) == ‘super foobar’
```

Docker playground

- https://github.com/pygrn/pytest
- /conftest.py => global: configuració i fixtures

```
docker-compose up

docker-compose exec db /app/docker/db/db_init.sh

docker-compose exec django bash

pytest
```

Fixtures - compartint amb conftest.py

```
@pytest.fixture(scope='function')
def reset_sqlite_db(request):
    path = request.param  # Path to database file
    with open(path, 'w'): pass
    yield None
    os.remove(path)
```

Code sample test més complert

```
@pytest.mark.smoketest
@pytest.mark.parametrize(“a”, “b”, “expected, [
        (1, 3, 4),
        (8, 20, 28)])
def test_addition():
        assert operator.add(1, 4) == 5
```