pytest

What?

- Framework to make testing easier in Python
- Opinionated:
  - Provides several defaults to make it easier to write tests
- Helpful features:
  - Can automatically set up environment, tear down after test etc.
  - Test fixtures, monkeypatching etc.

Note: python standard library includes `unittest` - pytest is an alternative with some more features

Example

```python
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

Example

```python
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

```
$ pytest
=============== test session starts =====================
platform linux -- Python 3.x.y, pytest-6.x.y, py-1.x.y,
pluggy-1.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 1 item

test_sample.py F
[100%]

=============== FAILURES =================================
________________________________ test_answer ______________

    def test_answer():
>       assert func(3) == 5
E       assert 4 == 5
E        +  where 4 = func(3)

test_sample.py:6: AssertionError
=============== short test summary info ==================
FAILED test_sample.py::test_answer - assert 4 == 5
=============== 1 failed in 0.12s ========================
```

Test for exceptions

```
# content of test_sysexit.py
import pytest


def f():
    raise SystemExit(1)


def test_mytest():
    with pytest.raises(SystemExit):
        f()
```

Temporary directory etc.

```
# content of test_tmpdir.py
def test_needsfiles(tmpdir):
    print(tmpdir)
    assert 0
```

Temporary directory etc.

```
# content of test_tmpdir.py
def test_needsfiles(tmpdir):
    print(tmpdir)
    assert 0
```

```
$ pytest -q test_tmpdir.py
F                                                                    [100%]
================================= FAILURES =================================
_____________________________ test_needsfiles ______________________________

tmpdir = local('PYTEST_TMPDIR/test_needsfiles0')

    def test_needsfiles(tmpdir):
        print(tmpdir)
>       assert 0
E       assert 0

test_tmpdir.py:3: AssertionError
--------------------------- Captured stdout call ---------------------------
PYTEST_TMPDIR/test_needsfiles0
========================= short test summary info ==========================
FAILED test_tmpdir.py::test_needsfiles - assert 0
1 failed in 0.12s
```

Test Fixtures

- Set up some data before test
- Remove after test
- Examples:
  - initialize dummy database
  - Create dummy users, files

Example: test fixture

```python
import pytest

@pytest.fixture
def setup_list():
    return ["apple", "banana"]

def test_apple(setup_list):
    assert "apple" in setup_list

def test_banana(setup_list):
    assert "banana" in setup_list

def test_mango(setup_list):
    assert "mango" in setup_list
```

Result: test fixture

```
test_fruit.py ..F                                      [100%]

========== FAILURES ============================
__________ test_mango __________________________

setup_list = ['apple', 'banana']

    def test_mango(setup_list):
>       assert "mango" in setup_list
E       AssertionError: assert 'mango' in ['apple', 'banana']

test_fruit.py:14: AssertionError
========== short test summary info ==========
FAILED test_fruit.py::test_mango - AssertionError: assert 'mango' in
['apple', 'banana']
========== 1 failed, 2 passed in 0.01s ======
```

Conventions

- Test discovery starts from current dir or **`testpaths`** variable
  - Recurse into subdirectories unless specified not to
- Search for files name `test_*.py` or `*_test.py`
- From those files:
  - `test` prefixed test functions or methods outside of class
  - `test` prefixed test functions or methods inside `Test` prefixed test classes (without an `__init__` method)
- Also supports standard python `unittest`

Testing Flask applications

- Create a `client` fixture - known to Flask
- Set up dummy database, temp dir etc. in fixture
- Use `requests` library to generate queries

Fixture setup

```python
import os
import tempfile

import pytest

from flaskr import create_app
from flaskr.db import init_db


@pytest.fixture
def client():
    db_fd, db_path = tempfile.mkstemp()
    app = create_app({'TESTING': True, 'DATABASE': db_path})

    with app.test_client() as client:
        with app.app_context():
            init_db()
        yield client

    os.close(db_fd)
    os.unlink(db_path)
```

Test example

```python
def test_empty_db(client):
    """Start with a blank database."""

    rv = client.get('/')
    assert b'No entries here so far' in rv.data
```

Testing login and other features

```python
def login(client, username, password):
    return client.post('/login', data=dict(
        username=username,
        password=password
    ), follow_redirects=True)


def logout(client):
    return client.get('/logout', follow_redirects=True)
```

```python
def test_login_logout(client):
    """Make sure login and logout works."""

    username = flaskr.app.config["USERNAME"]
    password = flaskr.app.config["PASSWORD"]

    rv = login(client, username, password)
    assert b'You were logged in' in rv.data

    rv = logout(client)
    assert b'You were logged out' in rv.data

    rv = login(client, f"{username}x", password)
    assert b'Invalid username' in rv.data

    rv = login(client, username, f'{password}x')
    assert b'Invalid password' in rv.data
```

Evaluation

```
import pytest
import os.path

class TestWeek1PublicCases:
    # Test case to check if the contact.html file exists
    def test_public_case1(self, student_assignment_folder):
        file_path = student_assignment_folder + "contact.html"
        assert os.path.isfile(file_path) == True

    # Test case to check if the resume.html file exists
    def test_public_case5(self, student_assignment_folder):
        file_path = student_assignment_folder + "resume.html"
        assert os.path.isfile(file_path) == True
```

Summary

- Automated testing is essential to get confidence in design
- Regression testing:
  - ensure previously passed tests do not start failing
- Test generation process:
  - mix of manual and automated

Continuous testing essential for overall system stability