

CÁCH VIẾT MỘT BỘ TEST CƠ BẢN

PHẦN BỔ SUNG CHO
KIỂM TRA TÍNH ĐÚNG ĐẮN VÀ HIỆU NĂNG CỦA CHƯƠNG TRÌNH
BẰNG BỘ TEST

TABLE OF CONTENT

Cấu trúc cơ bản của một project

Bộ test đơn giản, lệnh assert, chạy test

Cách sử dụng `@pytest.mark.parametrize`

Cách đảm bảo tính độc lập, mock

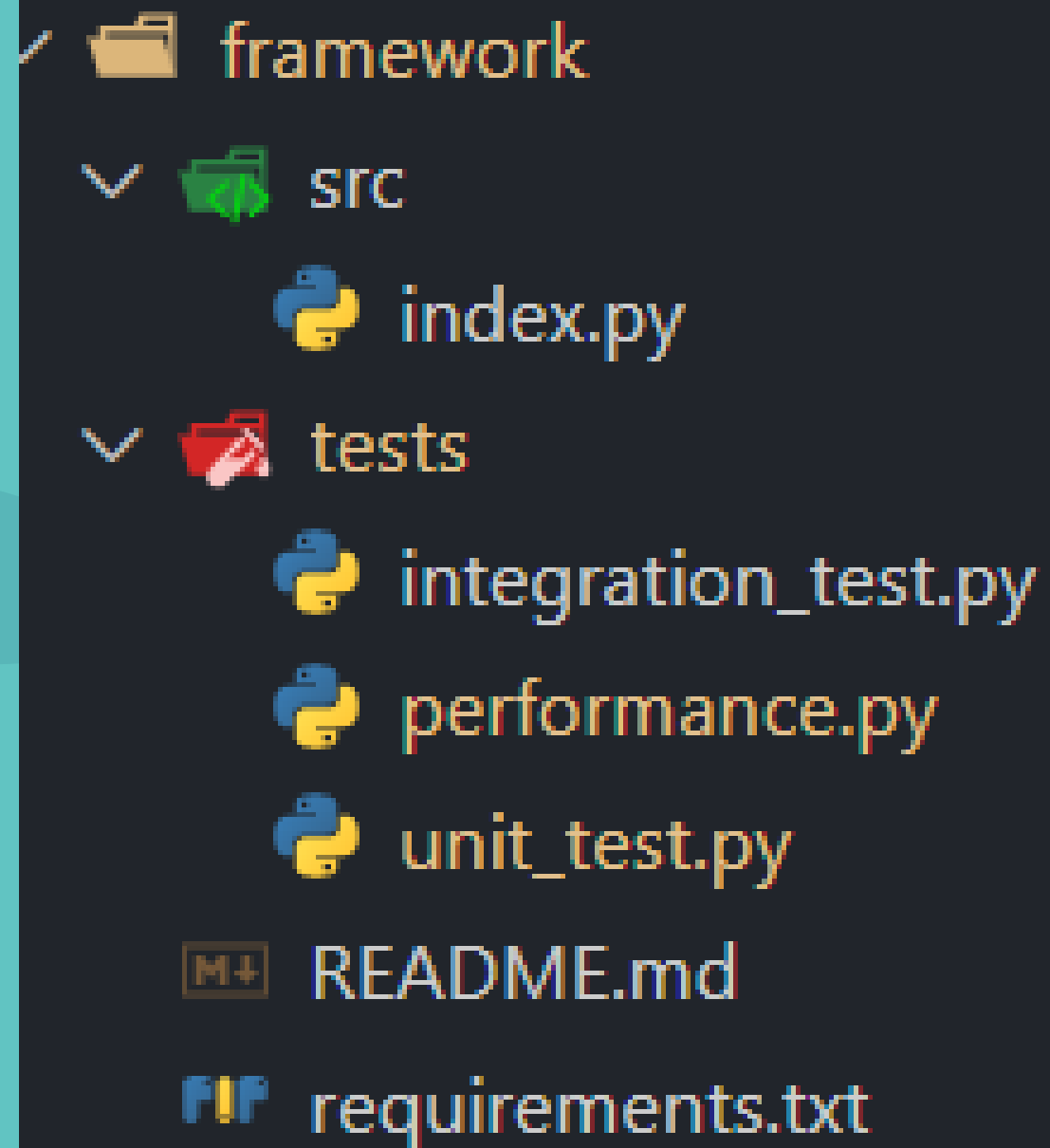
Kiểm tra hiệu năng

CHÚ Ý

Tất cả code trong phần này đã được đưa lên github [tại đây](#), đây cũng là framework chung, mọi người có thể tải về và dùng cho việc test các bài toán của mình

Mọi người có thể xem ví dụ chi tiết [tại đây](#).

Mọi người vui lòng đọc hướng dẫn chi tiết tại README của các thư mục đó để biết chi tiết cách cài đặt và chạy chương trình nha



```
✓ framework
  ✓ src
    index.py
  ✓ tests
    integration_test.py
    performance.py
    unit_test.py
  README.md
  requirements.txt
```

A screenshot of a file explorer interface with a dark background. It shows a project structure starting with a 'framework' folder. Inside 'framework', there is a 'src' folder containing 'index.py', and a 'tests' folder containing 'integration_test.py', 'performance.py', and 'unit_test.py'. At the root level of the project, there are also 'README.md' and 'requirements.txt' files. Icons for folders, Python files, and a README file are visible next to their respective names.

CẤU TRÚC CƠ BẢN CỦA PROJECT

Toàn bộ source code sẽ được đặt trong thư mục src/
Các file test sẽ được viết riêng trong thư mục tests

HÀM TEST CƠ BẢN

- Một hàm test cơ bản sẽ được đặt tên có dạng `test_*` hoặc `*_test`, mục đích của việc đặt tên này là để pytest nhận diện và chạy những hàm này khi test
- Lệnh `assert` có tác dụng là nếu biểu thức phía sau lệnh `assert` trả về `False` thì sẽ thông báo lỗi, chúng ta sẽ sử dụng `assert` để đảm bảo nếu kết quả trả về của hàm cần test khác với kết quả mong đợi thì ta sẽ được thông báo

```
def test_independentFunction_basic():  
    assert independentFunction(1) == 1
```

CHẠY TEST

Chúng ta có thể chạy test bằng lệnh

```
python -m pytest --cov=src/ tests/ --cov-branch --cov-report html
```

Lệnh này có tác dụng chạy tất cả các test có trong thư mục tests/, đồng thời tính branch coverage và đưa kết quả vào thư mục htmlcov

KẾT QUẢ CHẠY TEST THÀNH CÔNG

Tổng số test case và tỉ lệ thành công sẽ được hiển thị

```
PS D:\Learning\Python\CS112.N22.KHCL-Team-3\framework> python -m pytest --cov=src/ tests/ --cov-branch --cov-report html
===== test session starts =====
platform win32 -- Python 3.11.1, pytest-7.2.2, pluggy-1.0.0
rootdir: D:\Learning\Python\CS112.N22.KHCL-Team-3\framework
rootdir: D:\Learning\Python\CS112.N22.KHCL-Team-3\framework
plugins: cov-4.0.0, mock-3.10.0
collected 7 items

tests\unit_test.py ..... [100%]

----- coverage: platform win32, python 3.11.1-final-0 -----
Coverage HTML written to dir htmlcov

===== 7 passed in 0.12s =====
```

KẾT QUẢ CHẠY TEST THẤT BẠI

```
PS D:\Learning\Python\CS112.N22.KHCL-Team-3\framework> python -m pytest --cov=src/ tests/ --cov-branch --cov-report html
===== test session starts =====
platform win32 -- Python 3.11.1, pytest-7.2.2, pluggy-1.0.0
rootdir: D:\Learning\Python\CS112.N22.KHCL-Team-3\framework
plugins: cov-4.0.0, mock-3.10.0
collected 7 items

tests\unit_test.py F..... [100%]

===== FAILURES =====
----- test_independentFunction_basic -----

    def test_independentFunction_basic():
>     assert independentFunction(2) == 1
E       assert 2 == 1
E       + where 2 = independentFunction(2)

tests\unit_test.py:16: AssertionError

----- coverage: platform win32, python 3.11.1-final-0 -----
Coverage HTML written to dir htmlcov

===== short test summary info =====
FAILED tests/unit_test.py::test_independentFunction_basic - assert 2 == 1
===== 1 failed, 6 passed in 0.15s =====
```

- Các test case bị fail sẽ được hiển thị, đồng thời giá trị trong assert sẽ được hiện ra trong dòng màu đỏ
- Số lượng test failed, passed sẽ được hiển thị ở dưới cùng

KẾT QUẢ COVERAGE

Xem trong <htmlcov/index.html>

Coverage report: 86%

coverage.py v7.2.2, created at 2023-04-01 05:58 +0700

Module	statements	missing	excluded	branches	partial	coverage
src\index.py	7	1	0	0	0	86%
Total	7	1	0	0	0	86%

Coverage for src\index.py: 86%

7 statements 6 run 1 missing 0 excluded 0 partial

« prev ^ index » next coverage.py v7.2.2, created at 2023-04-01 05:54 +0700

```
1
2 # Hàm này không gọi hàm nào khác
3 def independentFunction(a):
4     return (a)
5
6 # Hàm này gọi hàm foo nên khi test bằng unit test cần phải mock
7 def functionThatCallOther(a):
8     result = foo(a)
9     return result
10
11
12 def foo(a):
13     return a
```

- Tỷ lệ branch coverage sẽ được ghi ở đây
- Nhấn vào một file xác định để xem branch nào chưa được cover

CÁCH SỬ DỤNG `@PYTEST.MARK.PARAMETRIZE`

- Có nhiều trường hợp, ta muốn sử dụng một hàm test nhưng cho nhiều test case khác nhau, ta có thể sử dụng `parametrize` để truyền nhiều test case vào hàm test qua các tham số

```
@pytest.mark.parametrize(
    "a, expected_result",
    [
        (1, 1), # Test case 1
        (2, 2), # Test case 2
        (3, 3), # Test case 3
    ],
)
def test_independentFunction(a, expected_result):
    assert independentFunction(a) == expected_result
```

@pytest.mark.parametrize nhận vào 2 tham số

- Tham số thứ nhất là danh sách các tham số sẽ được truyền vào hàm test
- Tham số thứ hai là list các test case

Các test case sẽ được lần lượt truyền vào hàm test để chạy, các giá trị trong một test case cũng sẽ được truyền lần lượt vào các tham số theo thứ tự

Trong hàm test ta sẽ dùng các giá trị thông qua các tham số ta đã xác định trước đó

CÁCH ĐẢM BẢO TÍNH ĐỘC LẬP

- Một trong những yêu cầu bắt buộc đối với unit test là ta phải kiểm tra nó trong môi trường độc lập
- Tuy nhiên, trong một hàm chúng ta sẽ có nhiều khả năng gọi một hàm khác mà hàm đó chưa được test đầy đủ hoặc nếu đã test thì cũng không thể đảm bảo chính xác 100%
- Để đảm bảo các lỗi trong các thành phần con không ảnh hưởng đến việc test thành phần chính, ta cần phải làm giả các thành phần đó (mock)

```
def functionThatCallOther(a):  
    result = foo(a)  
    return result
```

Generate tests for the below function

```
def foo(a):  
    return a
```

Trong ví dụ này, mình sẽ test hàm `functionThatCallOther`, hàm này có gọi hàm `foo`, do đó khi test hàm `functionThatCallOther` cần phải mock hàm `foo` trước

VIẾT HÀM MOCK

- Để mock hàm foo ta làm như hình dưới
- Khi viết hàm này, hàm foo vẫn chưa được mock, chỉ khi ta sử dụng fixture này cho các hàm test khác thì hàm foo mới được mock trong phạm vi của hàm test đó
- `foo_return_value` là giá trị ta mong muốn hàm foo sẽ trả về, chúng ta sẽ truyền các giá trị đó vào sau
- Ở đây, ta sẽ mock giá trị trả về của hàm foo bằng giá trị ta truyền trực tiếp vào

```
@pytest.fixture
def mockFoo(mocker, foo_return_value):
    mocker.patch("src.index.foo", return_value=foo_return_value)
```

VIẾT HÀM TEST CÓ SỬ DỤNG MOCK

- Ta sử dụng `@pytest.mark.usefixtures` để đánh dấu là sử dụng fixture `mockFoo`, khi đó hàm `foo` sẽ được mock
- Ta chú ý trong `parametrize`, ở danh sách các tham số có thêm `foo_return_value` đây là giá trị ta mong muốn hàm `foo` sẽ trả về trong từng test case, và giá trị này sẽ được truyền vào `mockFoo` để mock giá trị trả về của hàm `foo` bằng giá trị này

```
@pytest.mark.parametrize(
    "a, foo_return_value, expected_result",
    [
        (1, 1, 1), # Test case 1
        (2, 2, 2), # Test case 2
        (3, 3, 3), # Test case 3
    ],
)

@pytest.mark.usefixtures("mockFoo")
def test_functionThatCallOther(a, expected_result):
    assert functionThatCallOther(a) == expected_result
```

KIỂM TRA HIỆU NĂNG

- Ta sử dụng template này để kiểm tra hiệu năng một hàm bất kì, sau khi chạy chúng ta sẽ có kết quả về thời gian chạy chương trình và lượng bộ nhớ đã sử dụng
- Template này được khai báo trong `tests/performance.py`

```
@profile
def metricFunction():
    independentFunction(random.randrange(1, 28))

if __name__ == "__main__":
    execution_time = timeit.timeit(metricFunction, number=1)
    print(f"Execution time: {execution_time}")
```


KIỂM TRA HIỆU NĂNG

Chúng ta có thể kiểm tra hiệu năng bằng lệnh

```
mprof run tests/performance.py
```

Kết quả

```
PS D:\Learning\Python\CS112.N22.KHCL-Team-3\framework> mprof run tests/performance.py
mprof: Sampling memory every 0.1s
running new process
running as a Python program...
Filename: tests/performance.py
```

Line #	Mem usage	Increment	Occurrences	Line Contents
7	22.6 MiB	22.6 MiB	1	@profile
8				def metricFunction():
9	22.6 MiB	0.0 MiB	1	independentFunction(random.randrange(1, 28))

```
Execution time: 0.02206839999416843
```