pytest: How to mock in Python

If you have trouble understanding mocks for testing in Python like me, then this post is for you.

Requirements

To follow the post, please install

- `pytest`: https://pypi.org/project/pytest/
- `pytest-mock`: https://pypi.org/project/pytest-mock/

Both can be installed via `pip`.

The code in this post can be found in

- https://github.com/changhsinlee/pytest-mock-examples

What is mock?

Here's an example. Imagine that you have a function called `compute()`. Part of its code contains an `expensive_api_call()` that takes 1,000 seconds to run

```
import time

def compute(x):
    response = expensive_api_call()
    return response + x


def expensive_api_call():
```

```
    time.sleep(1000) # takes 1,000 seconds
    return 123
```

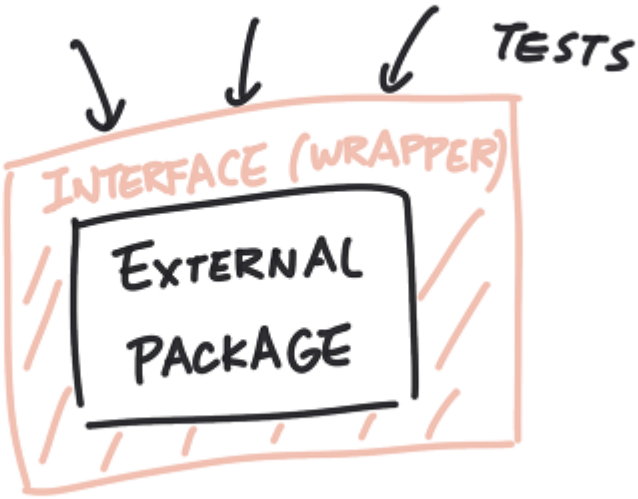
I would expect that `compute(1)` returns `124`, so I would write a test in Python:

```
def test_compute():
    expected = 124
    actual = compute(1)
    assert expected == actual
```

Because of the API call, this test also takes 1,000 seconds to run. This is too slow for a simple test.

When I write this test, I don't really care whether the API call runs or not. What I want to know when I develop is that my code works as expected when API returns correct data.

If I can provide fake data without calling the API, then I don't have to sit there are wait for the test to complete. This is where mocks come in.



Test an interface if possible

Shorten the feedback loop

In other words, it is a trick to shorten development feedback loop.

Let's review again: I have two options of writing a test for `compute()`.

1. Write a single test on `compute()` that contains both the api call `expensive_api_call()` and the computation `result + x`. Takes 1,000 seconds to run.
2. Write two tests: mock the API call in the test for `compute()`, and write another test to test that the API call returns correct data. The first test will be instant, and the second test will take 1,000 seconds.

Option 2 is better because the developer can choose run only the fast tests when she is developing. She can now run the integration tests elsewhere, for example, on a CI/CD server as part of the build process, that does not interfere with her flow.

So how do I replace the expensive API call in Python?

Mocking in pytest

In Python, the solution is a library called `mock`:

- https://docs.python.org/3/library/unittest.mock.html

The definition of `mock` in Merriam-Webster

```
to imitate (someone or something) closely : MIMIC
e.g. a mockingbird was mocking a cardinal
```

In Python, you use mocks to replace objects for testing purposes. In the next section, I am going to show you how to mock in `pytest`.

The workhorse: MagicMock

The most important object in `mock` is the `MagicMock` object. Playing with it and understanding it will allow you to do whatever you want.

The basic idea is that `MagicMock` a placeholder object with placeholder attributes that can be passed into any function.

I can

- mock a constant,
- mock an object with attributes,
- or mock a function, because a function is an object in Python and the attribute in this case is its return value.

Let's go through each one of them.

Recipes for using mocks in pytest

We will use pytest-mock to create the mock objects.

The `mocker` fixture is the interface in `pytest-mock` that gives us `MagicMock`.

Before diving in: what confused me

Before I go into the recipes, I want to tell you about the thing that confused me the most about Python mocks: where do I apply the mocks?

In general, when you mock an object, you want to mock where **the object is imported into** not **where the object is imported from**.

This caused so many lost time on me so let me say it again: mock where **the object is imported into** not **where the object is imported from**.

For more details, see the offical docs on this topic.

I will also demonstrate this point in the recipes.

Recipes

The code used in this post can be found in

- https://github.com/changhsinlee/pytest-mock-examples

1. Mocking a constant

The function `double()` reads a constant from another file and doubles it.

```
# functions.py
from .constants import CONSTANT_A  # CONSTANT_A = 1

def double():
    return CONSTANT_A * 2
```

Because `CONSTANT_A=1`, each call to `double()` is expected to return 2.

To replace `CONSTANT_A` in tests, I can use `patch.object()` and replace the `CONSTANT_A` object with another constant.

```
import mock_examples.functions
from mock_examples.functions import double


# note that I'm mocking the module when it is imported, not where
def test_mocking_constant_a(mocker):
    mocker.patch.object(mock_examples.functions, 'CONSTANT_A', 2)
    expected = 4
    actual = double()  # now it returns 4, not 2

    assert expected == actual
```

2. Mocking a function

In `main.py`, I have a slow function

```
from mock_examples.slow import api_call

def slow_function():
    api_result = api_call()
    # do some more stuff here
    return api_result
```

where it is slow because in `slow.py`,

```
def api_call():
    time.sleep(3)
    return 9
```

So each test will take at least 3 seconds to run.

When I mock a function, what I really care about is its return value, so I can patch the function with

```
def test_slow_function_mocked_api_call(mocker):
    mocker.patch(
        # api_call is from slow.py but imported to main.py
        'mock_examples.main.api_call',
        return_value=5
    )

    expected = 5
    actual = slow_function()
    assert expected == actual
```

This removes the dependency of the test on an external API or database call and makes the test instantaneous.

3. Mocking a class

For classes, there are many more things that you can do. Remembering that `MagicMock` can imitate anything with its attributes is a good place to reason about it.

I will only show a simple example here. For more complex ones, I recommend reading the references in the next section.

I have a class `Dataset` that has a slow method,

```
# slow.py
class Dataset:

    def __init__(self):
        self.data = None

    def load_data(self):
        time.sleep(4)
        self.data = 'slow data'
```

It is called as part of the `main()` function

```
# main.py
def slow_dataset():
```

```
    dataset = Dataset()
    return dataset.load_data()
```

For the test example, I am using `patch.object` to replace the method with a tiny function that returns the data that I want to use for testing:

```
from mock_examples.main import slow_dataset

def test_mocking_class_method(mocker):
    expected = 'xyz'

    def mock_load(self):
        return 'xyz'

    mocker.patch(
        # Dataset is in slow.py, but imported to main.py
        'mock_examples.main.Dataset.load_data',
        mock_load
    )
    actual = slow_dataset()
    assert expected == actual
```

Useful reference for mocking a class

There are many scenarios about mocking classes and here are some good references that I found:

- Mocking class instance and method at the same time
- Mocking class attributes

FAQ

Should I replace every API call with mocks?

No. I would combine integration tests and unit tests but not replace.

For developers, unit tests boost productivity. But for product development, integration tests are *absolutely necessary*. I still want to know when APIs external to the project start sending data that breaks my code. The testing can happen outside of developer's machine, however.

When should I mock?

In my opinion, the best time to mock is when you find yourself refactoring code or debugging part of code that runs slow but has zero test.

Trying to make changes without a test means you are incurring technical debt for the future and making teammates pay for it.

In this case, if my goal is making changes to the computations, I would figure out how to mock the data connectors and start writing tests.

The tests seem to be tied to how I implement the code. Isn't that a bad thing?

Mocks are always white-box tests. You can't use them without peeking into the code, so they are most useful for developers and not so much for testing specifications. It is a tradeoff that the developer has to accept.

Can you replace the return value in the same test twice??

Answer: yes. I don't know how to do this with the Python base library `mock` but it can be done with `pytest-mock`:

```
def test_mocking_constant_twice_in_same_test(mocker):
    mocker.patch.object(mock_examples.functions, 'CONSTANT_A', 3)
    expected_1 = 6
    actual_1 = double()

    mocker.patch.object(mock_examples.functions, 'CONSTANT_A', 10)
    expected_2 = 20
    actual_2 = double()

    assert expected_1 == actual_1
    assert expected_2 == actual_2
```

Learn form my mistakes

The most common mistake that I make when I write tests with mocks is… that I mock after I make the method call I want to patch:

```
actual = compute(x)
mocker.patch('some.function', fake_function)
# And I wonder why compute() wasn't patched :(
```

More than once I spent more than 15 minutes trying to figure out what was wrong 🤦. If you are having trouble getting mocks to work,

1. Make sure you are mocking where it is imported into
2. Make sure the mocks happen before the method call, not after

Written on May 2, 2020

ALSO ON **CHANG HSIN LEE**

PyJanitor and Pandas Method Chaining

a year ago • 1 comment

I want to share a Python package that makes your pandas code more …

Don't Wait, Schedule and Relax …

2 years ago • 2 comments

The purpose of automation is to let machine do things while us humans rest. In …

Becoming a Better Data Scientist: …

2 years ago • 1 comment

This story has happened to me too many times: First, I started a project. Then I …

Making Exc Formulas, a

2 years ago • 1

In this last pos I will continue Excel with Pyt

What do you think?

22 Responses

Upvote

1 Comment **Chang Hsin Lee** **Disqus' Privacy Policy** **Login**

Recommend Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Riteek Srivastava • 2 months ago

In the section `Mocking a class` if dataset is being passed as dependency injection instead of creating a new one inside `slow  dataset`. Then how will