Documentation

A Large portion of our unit-tests use mocking that is why it is crucial to explain what mocking is and how I used it in the tests.

# What is Mocking ?

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object you want to replace the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test. In short, mocking is creating objects that simulate the behavior of real objects.

The centerpoint of the pytest-mock module is, of course, the Mock fixture. The main characteristic of a Mock object is that it will return another Mock instance when:

* accessing one of its attributes
* calling the object itself

# Why Mocking is used?

Mocking is performed using Pytest-mock library for python which provides the mocker fixture that can used as parameter to function that can patch/separate a functionality from its dependencies.

## Example:

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():

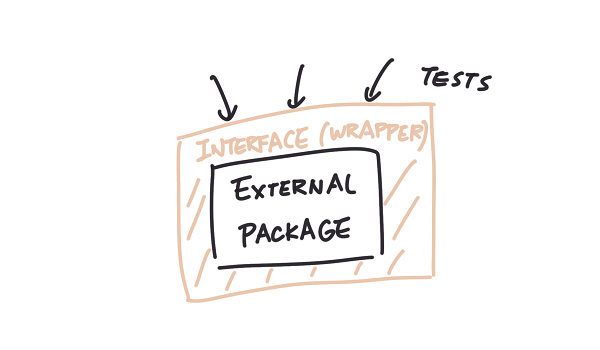
expected = 124

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.

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().

* 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.
* 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

Consider an example:

-----------------------------------------------------------------------

# functions.py

from .constants import CONSTANT\_A # CONSTANT\_A = 1

def double():

return CONSTANT\_A \* 2

---------------------------------------------------------------------------

import mock\_examples.functions

from mock\_examples.functions import double

# note that I'm mocking the module when it is imported, not where CONSTANT\_A is from

def test\_mocking\_constant\_a(mocker):

mocker.patch.object(mock\_examples.functions, 'CONSTANT\_A', 2)

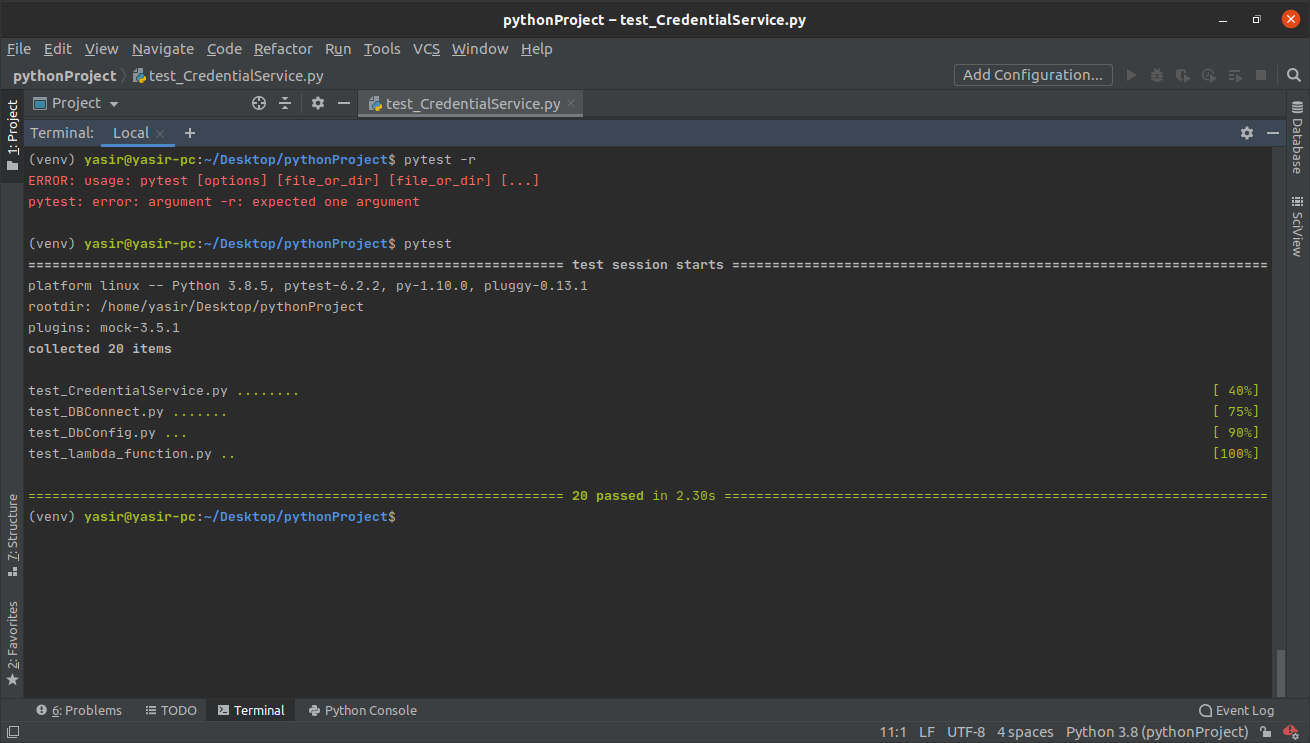
expected = 4

actual = double() # now it returns 4, not 2

assert expected == actual

our test\_mocking\_constant recieves a fixture called mocker, and it has method like patch which receives the first parameter as what to mock and second parameter is the name of the attribute, since functions are objects in python, and the third one is the return value. So whenever, the python interpreter sees line after the patch like ‘‘actual = double()’’, it will not run the original basis function that has dependencies but rather provide handler to our patch and the patch will return the value we have provided, so we actually imitate the return value provided by the dependencies.

# Test Results



# Coverage of the code

As you can see in the above picture, the total lines coverage in the end is 100