Python Mock Requests

Summary: In this tutorial, you'll learn how to mock the `requests` module in Python to test an API call using the unittest (https://www.pythontutorial.net/python-unit-testing/python-unittest/) module.

The `requests` module is an HTTP library that allows you to send HTTP requests easily. Typically, you use the `requests` module to call an API from a remote server.

The scenario

For the demo purposes, we'll use a public API provided by jsonplaceholder.typicode.com (https://jsonplaceholder.typicode.com/) :

```
https://jsonplaceholder.typicode.com/
```

To make an API call, you'll use the `requests` method to send an HTTP GET method to the following end-point:

```
https://jsonplaceholder.typicode.com/albums/1
```

It'll return JSON data in the following format:

```
{
  "userId": 1,
  "id": 1,
  "title": "quidem molestiae enim"
}
```

Since the `requests` module is not a built-in module, you need to install it by running the following pip command:

```
pip install requests
```

Making an API call using the requests module

The following defines a new module called `album.py` with a function `find_album_by_id()` that returns an album by an id:

```
import requests


def find_album_by_id(id):
    url = f'https://jsonplaceholder.typicode.com/albums/{id}'
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()['title']
    else:
        return None
```

How it works.

First, format the API end-point that includes the id parameter:

```
url = f'https://jsonplaceholder.typicode.com/albums/{id}'
```

Second, call the `get()` function of the requests module to get a `Response` object:

```
response = requests.get(url)
```

Third, call the `json()` method of the response object if the API call succeeds:

```
if response.status_code == 200:
    return response.json()['title']
else:
    return None
```

The `response.json()` returns a dictionary that represents the JSON data.

Creating a test module

We'll create a `test_album.py` test module that tests the functions in the `album.py` module:

```
import unittest

from album import find_album_by_id


class TestAlbum(unittest.TestCase):
    pass
```

Mocking the requests module

The `find_album_by_id()` function has two dependencies:

- The `get()` method of the `requests` module
- The `Response` object returned by the `get()` function.

So to test the `find_album_by_id()` function, you need to:

- First, mock the requests module and call the `get()` function (`mock_requests`)
- Second, mock the returned response object.

In other words, the mock_requests. `get()` returns a mock response object.

To mock the requests module, you can use the `patch()` function. Suppose that the `mock_requests` is a mock of the `requests` module.

The `mock_requests.get()` should return a mock for the response. To mock the response, you can use the `MagicMock` class of the `unittest.mock` module.

The following shows how to test the `find_album_by_id()` using the `test_find_album_by_id_success()` test method:

```python
import unittest
from unittest.mock import MagicMock, patch

from album import find_album_by_id


class TestAlbum(unittest.TestCase):

    @patch('album.requests')
    def test_find_album_by_id_success(self, mock_requests):
        # mock the response
        mock_response = MagicMock()
        mock_response.status_code = 200
        mock_response.json.return_value = {
            'userId': 1,
            'id': 1,
            'title': 'hello',
        }

        # specify the return value of the get() method
        mock_requests.get.return_value = mock_response

        # call the find_album_by_id and test if the title is 'hello'
        self.assertEqual(find_album_by_id(1), 'hello')
```

How it works.

First, patch the `requests` module as the `mock_requests` object:

```
@patch('album.requests')
def test_find_album_by_id_success(self, mock_requests):
    # ...
```

Second, mock the response of the `get()` function using the `MagicMock` class. In this test method, we specify the status code 200 and `return_value` of the `json()` function as a hard-coded value:

```
mock_response = MagicMock()
mock_response.status_code = 200
mock_response.json.return_value = {
    'userId': 1,
    'id': 1,
    'title': 'hello',
}
```

Third, use the mock_response as the return value of the `get()` function:

```
 mock_requests.get.return_value = mock_response
```

Finally, test if the title of the album is equal to the one that we specified in the `return_value` of the `mock_response` :

```
self.assertEqual(find_album_by_id(1), 'hello')
```

Run the test:

```
python -m unittest -v
```

Output:

```
test_find_album_by_id_success (test_album.TestAlbum) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.001s
```

```
OK
```

By using the same technique, you can also test the `find_album_by_id()` function in the failed case:

```
import unittest
from unittest.mock import MagicMock, patch

from album import find_album_by_id


class TestAlbum(unittest.TestCase):

    @patch('album.requests')
    def test_find_album_by_id_success(self, mock_requests):
        # mock the response
        mock_response = MagicMock()
        mock_response.status_code = 200
        mock_response.json.return_value = {
            'userId': 1,
            'id': 1,
            'title': 'hello',
        }

        # specify the return value of the get() method
        mock_requests.get.return_value = mock_response

        # call the find_album_by_id and test if the title is 'hello'
        self.assertEqual(find_album_by_id(1), 'hello')

    @patch('album.requests')
    def test_find_album_by_id_fail(self, mock_requests):
        mock_response = MagicMock()
        mock_response.status_code = 400
        mock_requests.get.return_value = mock_response
        self.assertIsNone(find_album_by_id(1))
```

Output:

```
test_find_album_by_id_fail (test_album.TestAlbum) ... ok
test_find_album_by_id_success (test_album.TestAlbum) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```

Summary

- Use the `patch()` function to mock the requests module (`mock_requests`)
- Use the `MagicMock` to mock the response returned by the `mock_requests.get()` function.