Mocking Strategies in Python

Jordan Cahill


Quality
Reliability
Maintainability
Extendability


Quantity
More features + Less time
=
Quickly monetize

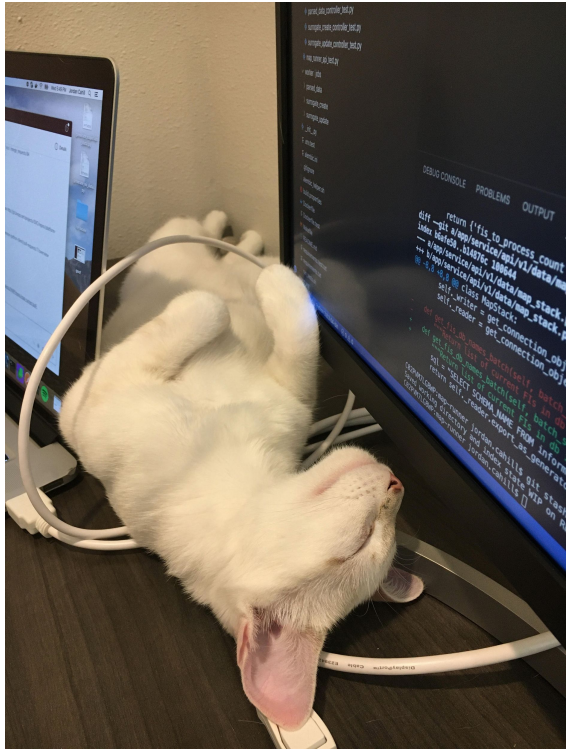
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.

Testing Flask Applications

Something that is untested is broken.

The origin of this quote is unknown and while it is not entirely correct, it is
applications make it hard to improve existing code and developers of untest
paranoid. If an application has automated tests, you can safely make change
breaks.

The University of Texas at Austin
Institute for Cellular and Molecular Biology
College of Natural Sciences



High Code Coverage
≠
Well Tested Codebase

queue_example.py > ...

```python
import json


def put_on_queue(data_1, data_2, queue):
    queue_data = dict(
        first_data=data_1,
        second_data=data_2
    )
    queue.put(json.dumps(queue_data))
    return True

```

tests > test_queue_example.py > ...

```python
import json
from queue import Queue

import pytest

from queue_example import put_on_queue


def test_put_on_queue_invaluable():
    # arrange
    test_queue = Queue()

    # act
    actual = put_on_queue(
        'some test data',
        'some more test data',
        test_queue
    )

    # assert
    assert actual == True

```

High Code Coverage
≠
Well Tested Codebase

queue_example.py > ...

```python
import json


def put_on_queue(data_1, data_2, queue):
    queue_data = dict(
        first_data=data_1,
        second_data=data_2
    )
    queue.put(json.dumps(queue_data))

```

tests > test_queue_example.py > ...

```python
import json
from queue import Queue

import pytest

from queue_example import put_on_queue


def test_put_on_queue_valuable():
    # arrange
    test_queue = Queue()
    test_data_1 = 'some test data'
    test_data_2 = 'some more test data'

    # act
    put_on_queue(
        test_data_1,
        test_data_2,
        test_queue
    )

    # assert
    expected = json.dumps(dict(
        first_data=test_data_1,
        second_data=test_data_2
    ))
    assert test_queue.get() == expected

```

High Code Coverage
≠
Well Tested Codebase

examples > sqs_queue.py > ...

You, a few seconds ago | 1 author (You)

```python
import json


def put_on_queue(data_1, data_2, queue):
    queue_data = dict(
        first_data=data_1,
        second_data=data_2
    )
    queue.send_message(MessageBody=json.dumps(queue_data))

```

tests > examples > test_sqs_queue.py > ...

You, a few seconds ago | 1 author (You)

```python
import json
from queue import Queue

import pytest

from examples.sqs_queue import put_on_queue


def test_put_on_queue(mocker):
    # arrange
    mock_queue = mocker.MagicMock()
    mock_send_message = mocker.patch.object(mock_queue, 'send_message')

    test_data_1 = 'some test data'
    test_data_2 = 'some more test data'

    # act
    put_on_queue(
        test_data_1,
        test_data_2,
        mock_queue
    )

    # assert
    expected = json.dumps(dict(
        first_data=test_data_1,
        second_data=test_data_2
    ))
    mock_send_message.assert_called_once_with(
        MessageBody=expected
    )
```

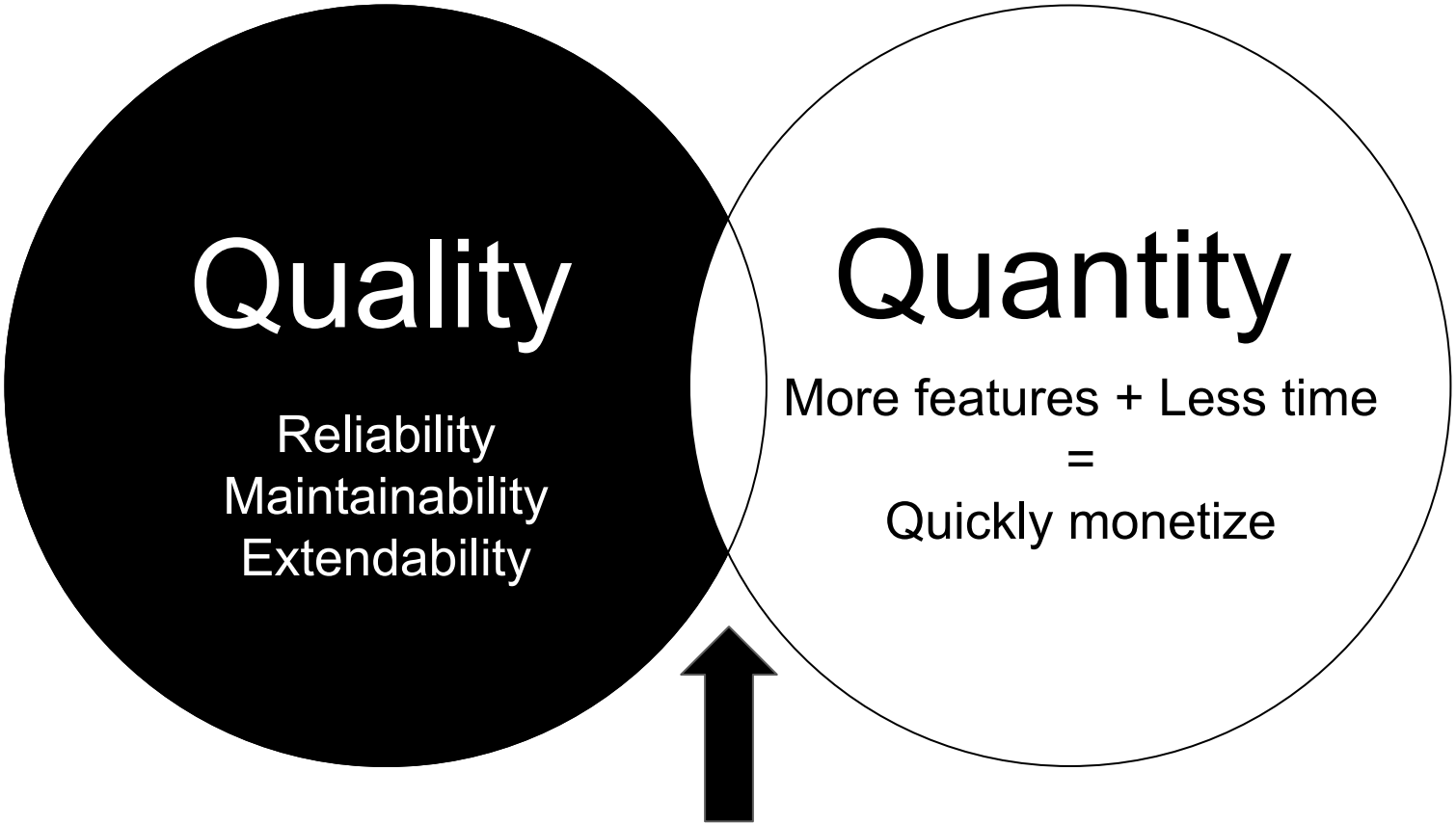
The main cases where mocking will be required in order to make the proper assertions:

1. Mocking an ***external service*** (or a func/class that calls an external service) so that you do not actually reach out to that service during the automated tests

2. Mocking a class or function called within the unit being tested to ***limit the unit being tested*** to a small unit

Python Mocking Tools

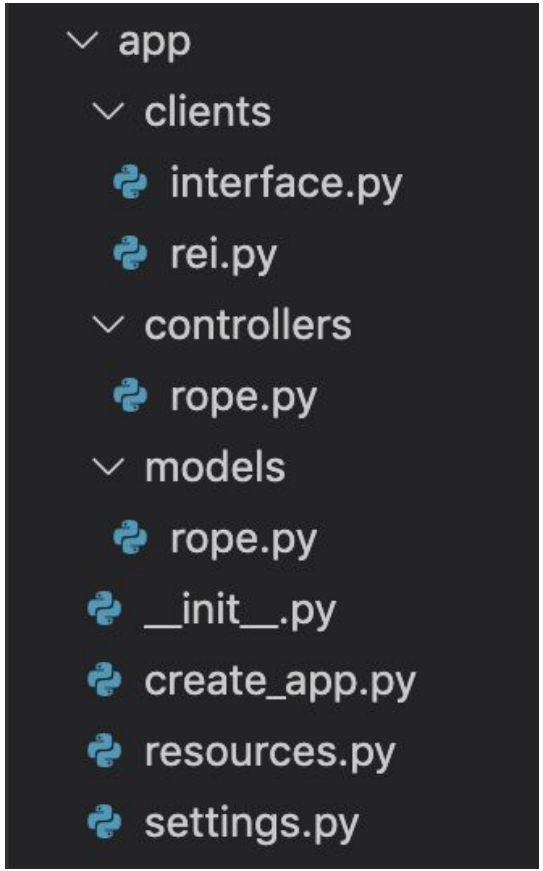
```python
def test_pretend(mocker):

    # 1 - Mock a function that is called within the code being tested
    mocker.patch('path.to.code.being.tested.func_being_mocked')

    # 2 - Mock a method on an object which is passed to the code being tested
    mocker.patch.object(my_obj, 'my_method')

    # 3 - Mock a method on a class which is instantiated within the code being tested
    mocker.patch.object(MyClass, 'my_method')

    # 4 - Use a mock object as input to a class or function since the actual input is “complicated”
    mocker.MagicMock(some_attr='my mock needs this attr')

    # ^^^ can pass return_value or side_effect to any of the above

    # 5 - Mock environment variables
    mocker.patch.dict('path.to.code.being.tested.os.environ', {'MOCK': 'ENVVAR'}, clear=True)
```



This could be you!

Code Example:
Online Climbing Gear Comparison Site

Directory Structure



create_app.py

app > create_app.py > ...

You, a few seconds ago | 1 author (You)

```python
from flask import Flask
from flask_restful import Api

from app.resources import RopeCollection


def create_app():
    app = Flask(__name__)
    api = Api(app)

    api.add_resource(RopeCollection, '/ropes')

    return app


if __name__ == "__main__":
    app = create_app()
    app.run(port=5000, debug=True)

```

settings.py

app > settings.py > ...

You, a few seconds ago | 1 author (You)

```python
import os
from dataclasses import dataclass


def get_settings() -> dict:
    return {
        'rei': {
            'base_url': os.environ.get('REI_BASE_URL', 'placeholder'),
            'rope_path': os.environ.get('REI_ROPE_PATH', 'placeholder')
        }
    }

```

__init__.py

app > __init__.py > ...

You, a few seconds ago | 1 author (You)

```python
from app.settings import get_settings

app_settings = get_settings()

```

resources.py

app > resources.py > ...

You, a few seconds ago | 1 author (You)

```python
from flask_restful import Resource

from app import app_settings
from app.clients.rei import ReiClient
from app.controllers.rope import RopeController


```

You, 24 minutes ago | 1 author (You)

```python
class RopeCollection(Resource):

    _clients_with_rope = [
        ReiClient(**app_settings['rei'])
    ]

    def get(self):
        ropes = RopeController().fetch_many(
            self._clients_with_rope
        )
        return {'data': ropes}, 200

```

controllers/rope.py

app > controllers > rope.py > ...

You, a few seconds ago | 1 author (You)

```python
from typing import List
from dataclasses import asdict

from app.models.rope import RopeModel


```

You, an hour ago | 1 author (You)

```python
class RopeController:

    def fetch_many(self, clients_with_rope) -> List[dict]:
        rope_models = []
        for client in clients_with_rope:
            rope_models += client.fetch_ropes()
        return [asdict(rope_model) for rope_model in rope_models]

```

app > models > rope.py > ...

You, a few seconds ago | 1 author (You)

```python
from dataclasses import dataclass


@dataclass
```

You, 3 hours ago | 1 author (You)

```python
class RopeModel:
    diameter: float
    length: int
    brand: str
    color: str
    price: float

```

models/rope.py

app > clients > rei.py > ...

You, a few seconds ago | 1 author (You)

```python
import requests
from os import path
from typing import List

from app.clients.interface import HasRope
from app.models.rope import RopeModel


```

You, 29 minutes ago | 1 author (You)

```python
class ReiClient(HasRope):

    def __init__(self, base_url, rope_path):
        self._rope_endpoint = path.join(base_url, rope_path)

    def fetch_ropes(self) -> List[RopeModel]:
        response = requests.get(self._rope_endpoint)
        response_dict = response.json()
        rope_models = []
        for rope in response_dict.get('data', []):
            rope_models.append(
                RopeModel(
                    rope.get('diameter'),
                    rope.get('length'),
                    rope.get('brand'),
                    rope.get('color'),
                    rope.get('price')
                )
            )
        return rope_models

```

clients/rei.py

clients/interface.py

app > clients > interface.py > ...

You, a few seconds ago | 1 author (You)

```python
from abc import ABC, abstractmethod
from typing import List

from app.models.rope import RopeModel


```

You, 3 hours ago | 1 author (You)

```python
class HasRope(ABC):

    @abstractmethod
    def fetch_ropes(self) -> List[RopeModel]:
        pass

```

Now let’s check out the tests!!!!

3 - Mock a method on a class which is instantiated within the code being tested

app > resources.py > ...

You, a few seconds ago | 1 author (You)

```python
from flask_restful import Resource

from app import app_settings
from app.clients.rei import ReiClient
from app.controllers.rope import RopeController


```

You, 24 minutes ago | 1 author (You)

```python
class RopeCollection(Resource):

    _clients_with_rope = [
        ReiClient(**app_settings['rei'])
    ]

    def get(self):
        ropes = RopeController().fetch_many(
            self._clients_with_rope
        )
        return {'data': ropes}, 200

```

tests > app > test_resources.py > ...

```python
import json
import pytest

from app.create_app import create_app
from app.controllers.rope import RopeController
from app.resources import RopeCollection


@pytest.fixture
def test_client():
    app = create_app()
    with app.test_client() as client:
        yield client


def test_get_ropes(mocker, test_client):
    # arrange
    test_ropes = [
        {
            'test rope': 'test rope 1'
        },
        {
            'test rope': 'test rope 2'
        }
    ]
    mock_rope_controller = mocker.patch.object(RopeController, 'fetch_many', return_value=test_ropes)

    # act
    actual = test_client.get('/ropes')

    # assert
    assert json.loads(actual.data) == {'data': test_ropes}
    assert actual.status_code == 200

    mock_rope_controller.assert_called_once_with(
        RopeCollection._clients_with_rope
    )

```

2 - Mock a method on an object which is passed to the code being tested

app > controllers > rope.py > ...

```
You, a few seconds ago | 1 author (You)
1  from typing import List
2  from dataclasses import asdict
3
4  from app.models.rope import RopeModel
5
6
   You, an hour ago | 1 author (You)
7  class RopeController:
8
9      def fetch_many(self, clients_with_rope) -> List[dict]:
10         rope_models = []
11         for client in clients_with_rope:
12             rope_models += client.fetch_ropes()
13         return [asdict(rope_model) for rope_model in rope_models]
14
```

Not the best solution for this case however …

tests > app > controllers > test_rope_controller.py > ...

```
   You, a few seconds ago | 1 author (You)
1  from dataclasses import asdict
2
3  import pytest
4
5  from app import app_settings
6  from app.clients.rei import ReiClient
7  from app.controllers.rope import RopeController
8  from app.models.rope import RopeModel
9
10
11 def test_rope_controller_fetch_many_1(mocker):
12     # arrange
13     test_rei_client = ReiClient(**app_settings['rei'])
14     test_rope_model_1 = RopeModel(
15         diameter=9.4,
16         length=70,
17         brand='mammut',
18         color='blue',
19         price=264.95
20     )
21     test_rope_model_2 = RopeModel(
22         diameter=9.9,
23         length=60,
24         brand='black diamond',
25         color='green',
26         price=199.95
27     )
28     mock_fetch_ropes = mocker.patch.object(test_rei_client, 'fetch_ropes', return_value=[
29         test_rope_model_1,
30         test_rope_model_2
31     ])
32
33     # act
34     actual = RopeController().fetch_many([test_rei_client])
35
36     # assert
37     assert actual == [
38         asdict(test_rope_model_1),
39         asdict(test_rope_model_2)
40     ]
41     mock_fetch_ropes.assert_called_once_with()
```

```python
def test_rope_controller_fetch_many_2(mocker):
    # arrange
    test_client_with_rope_1 = mocker.MagicMock()
    test_client_with_rope_2 = mocker.MagicMock()
    test_rope_model_1 = RopeModel(
        diameter=9.4,
        length=70,
        brand='mammut',
        color='blue',
        price=264.95
    )
    test_rope_model_2 = RopeModel(
        diameter=9.9,
        length=60,
        brand='black diamond',
        color='green',
        price=199.95
    )
    test_rope_model_3 = RopeModel(
        diameter=9.4,
        length=40,
        brand='edelrid',
        color='pink',
        price=199.95
    )
    test_rope_model_4 = RopeModel(
        diameter=9.9,
        length=60,
        brand='mammut',
        color='yellow',
        price=264.95
    )
```

 # 4 - Use a mock object as input to a class or function since the actual input is “complicated”

&

 # 2 - Mock a method on an object which is passed to the code being tested

```python
    mock_fetch_ropes_1 = mocker.patch.object(test_client_with_rope_1, 'fetch_ropes', return_value=[
        test_rope_model_1,
        test_rope_model_2
    ])
    mock_fetch_ropes_2 = mocker.patch.object(test_client_with_rope_2, 'fetch_ropes', return_value=[
        test_rope_model_3,
        test_rope_model_4
    ])

    # act
    actual = RopeController().fetch_many([test_client_with_rope_1, test_client_with_rope_2])

    # assert
    assert actual == [
        asdict(test_rope_model_1),
        asdict(test_rope_model_2),
        asdict(test_rope_model_3),
        asdict(test_rope_model_4)
    ]
    mock_fetch_ropes_1.assert_called_once_with()
    mock_fetch_ropes_2.assert_called_once_with()
```

1 - Mock a function that is called within the code being tested

& # 3 & # 4 … Nested mocking!

app > clients > rei.py > ...

You, a few seconds ago | 1 author (You)

```python
import requests
from os import path
from typing import List

from app.clients.interface import HasRope
from app.models.rope import RopeModel


You, 29 minutes ago | 1 author (You)
class ReiClient(HasRope):

    def __init__(self, base_url, rope_path):
        self._rope_endpoint = path.join(base_url, rope_path)

    def fetch_ropes(self) -> List[RopeModel]:
        response = requests.get(self._rope_endpoint)
        response_dict = response.json()
        rope_models = []
        for rope in response_dict.get('data', []):
            rope_models.append(
                RopeModel(
                    rope.get('diameter'),
                    rope.get('length'),
                    rope.get('brand'),
                    rope.get('color'),
                    rope.get('price')
                )
            )
        return rope_models

```

tests > app > clients > test_rei_client.py > ...

You, a few seconds ago | 1 author (You)

```python
import pytest

from app.clients.rei import ReiClient
from app.models.rope import RopeModel


def test_rei_client_fetch_ropes(mocker):
    # arrange
    test_rope_1 = dict(
        diameter=9.4,
        length=70,
        brand='mammut',
        color='blue',
        price=264.95
    )
    test_rope_2 = dict(
        diameter=9.9,
        length=60,
        brand='black diamond',
        color='green',
        price=199.95
    )
    test_response_dict = {
        'data': [
            test_rope_1,
            test_rope_2
        ]
    }
    mock_response = mocker.MagicMock()
    mocker.patch.object(mock_response, 'json', return_value=test_response_dict)
    mock_requests_get = mocker.patch('app.clients.rei.requests.get', return_value=mock_response)

    # act
    client = ReiClient('test-rei-url.com', 'random-rope-path')
    actual = client.fetch_ropes()

    # assert
    assert actual == [
        RopeModel(**test_rope_1),
        RopeModel(**test_rope_2)
    ]
    mock_requests_get.assert_called_once_with(
        'test-rei-url.com/random-rope-path'
    )
```

Other tricks

`side_effect` as optional keyword argument to mocker.patch or mocker.patch.object

- *Exception Class* → to raise error and assert proper measures are taken in case of that error
- *List* → to return a different value on each call to the mock object
- *Function* → to execute a function with the parameters passed to the mock as the parameters passed to the function

Other tricks

Other assertions to make on a mock object based on its calls:

- `assert_has_calls`
  - Pass a list of call objects to `assert_has_calls` to ensure your mock object has been called more than once with different parameters
- `call_count`
  - `assert_has_calls` does not ensure that those are the only calls on the mock object so it is good to pair `assert_has_calls` with `assert mock_obj.call_count == <int>`
- `called`
  - Can simply assert a mock has been called with `assert mock_obj.called is True`
- `assert_called_with`
  - Can assert a mock has been called with parameters without specifying one call

Summary

- Why ***mocking is valuable*** → allows you to make ***valuable assertions***
- When to mock → to ***avoid external services*** or ***reduce the scope of the unit*** being tested
- What ***mocking tools*** are available → there are 5 main types of Python mocks which will account for almost all use cases
- When to ***assert on what your mock object was called with*** → always
- How can I be a more ***efficient and reliable*** developer → use the proper mocking tools for each situation so that your testing process becomes a ***standard operating procedure***

Thanks for attending the presentation!

All code samples & these slides available at
github.com/chord-memory/python-mocking-presentation

Listen to Namespace Podcast on any podcast app, YouTube, or our website
namespace-pod.buzzsprout.com

Feel free to connect with me on LinkedIn or send remarks to
jordancahill88@gmail.com