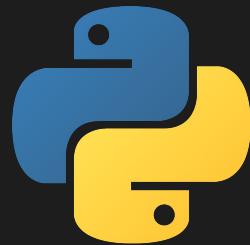


# Mocking Strategies in Python



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.



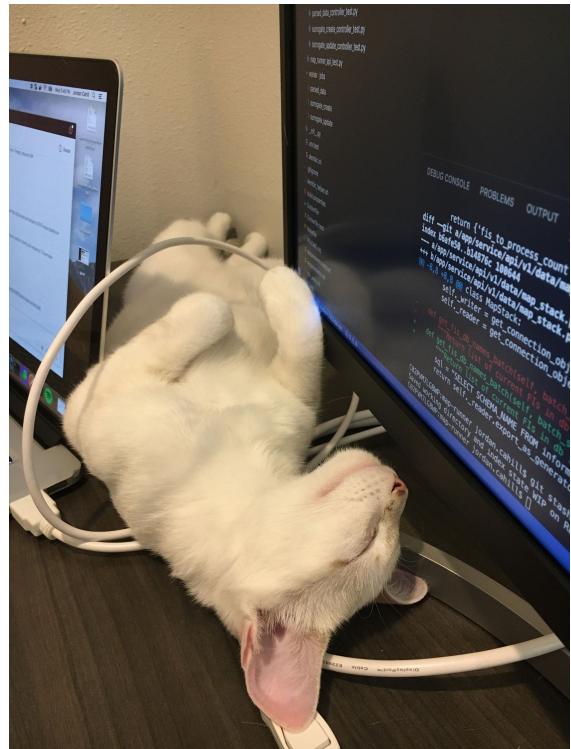
# Testing Flask Applications

**Something that is untested is broken.**

The origin of this quote is unknown and while it is not entirely correct, it is a good reminder. Untested code is more likely to have bugs and regressions. Automated tests help catch these issues early. In addition, well-written tests can serve as documentation and make it easier to understand the codebase. By running tests frequently, developers can quickly identify changes that break existing functionality and fix them before they become major problems. This leads to more reliable and maintainable applications over time.



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



# High Code Coverage ≠ Well Tested Codebase

```
py queue_example.py > ...
1 import json
2
3
4 def put_on_queue(data_1, data_2, queue):
5     queue_data = dict(
6         first_data=data_1,
7         second_data=data_2
8     )
9     queue.put(json.dumps(queue_data))
10    return True
11
```

```
tests > py test_queue_example.py > ...
1 import json
2 from queue import Queue
3
4 import pytest
5
6 from queue_example import put_on_queue
7
8
9 def test_put_on_queue_invaluable():
10     # arrange
11     test_queue = Queue()
12
13     # act
14     actual = put_on_queue(
15         'some test data',
16         'some more test data',
17         test_queue
18     )
19
20     # assert
21     assert actual == True
22
```

# High Code Coverage ≠

## Well Tested Codebase

queue\_example.py > ...

```
1 import json
2
3
4 def put_on_queue(data_1, data_2, queue):
5     queue_data = dict(
6         first_data=data_1,
7         second_data=data_2
8     )
9     queue.put(json.dumps(queue_data))
10
```

```
tests > test_queue_example.py > ...
1 import json
2 from queue import Queue
3
4 import pytest
5
6 from queue_example import put_on_queue
7
8
9 def test_put_on_queue_valuable():
10     # arrange
11     test_queue = Queue()
12     test_data_1 = 'some test data'
13     test_data_2 = 'some more test data'
14
15     # act
16     put_on_queue(
17         test_data_1,
18         test_data_2,
19         test_queue
20     )
21
22     # assert
23     expected = json.dumps(dict(
24         first_data=test_data_1,
25         second_data=test_data_2
26     ))
27     assert test_queue.get() == expected
28
```

# High Code Coverage ≠ Well Tested Codebase

```
examples > 🐍 sqs_queue.py > ...
You, a few seconds ago | 1 author (You)
1 import json
2
3
4 def put_on_queue(data_1, data_2, queue):
5     queue_data = dict(
6         first_data=data_1,
7         second_data=data_2
8     )
9     queue.send_message(MessageBody=json.dumps(queue_data))
10
```

```
tests > examples > 🐍 test_sqs_queue.py > ...
You, a few seconds ago | 1 author (You)
1 import json
2 from queue import Queue
3
4 import pytest
5
6 from examples.sqs_queue import put_on_queue
7
8
9 def test_put_on_queue(mocker):
10     # arrange
11     mock_queue = mocker.MagicMock()
12     mock_send_message = mocker.patch.object(mock_queue, 'send_message')
13
14     test_data_1 = 'some test data'
15     test_data_2 = 'some more test data'
16
17     # act
18     put_on_queue(
19         test_data_1,
20         test_data_2,
21         mock_queue
22     )
23
24     # assert
25     expected = json.dumps(dict(
26         first_data=test_data_1,
27         second_data=test_data_2
28     ))
29     mock_send_message.assert_called_once_with(
30         MessageBody=expected
31     )
```

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

```
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

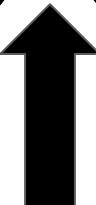
    # 5 - Mock environment variables
    mocker.patch.dict('path.to.code.being.tested.os.environ', {'MOCK': 'ENVVAR'}, clear=True)
```

# Quality

Reliability  
Maintainability  
Extendability

# Quantity

More features + Less time  
=  
Quickly monetize



*This could be you!*



# *Code Example:* Online Climbing Gear Comparison Site



## Directory Structure

```
└─ app
    └─ clients
        └─ interface.py
        └─ rei.py
    └─ controllers
        └─ rope.py
    └─ models
        └─ rope.py
    └─ __init__.py
    └─ create_app.py
    └─ resources.py
    └─ settings.py
```

## create\_app.py

```
app > ⚙ create_app.py > ...
You, a few seconds ago | 1 author (You)
1   from flask import Flask
2   from flask_restful import Api
3
4   from app.resources import RopeCollection
5
6
7   def create_app():
8       app = Flask(__name__)
9       api = Api(app)
10
11      api.add_resource(RopeCollection, '/ropes')
12
13      return app
14
15
16  if __name__ == "__main__":
17      app = create_app()
18      app.run(port=5000, debug=True)
19
```

## *settings.py*

```
app > 🐍 settings.py > ...
You, a few seconds ago | 1 author (You)
1 import os
2 from dataclasses import dataclass
3
4
5 def get_settings() -> dict:
6     return {
7         'rei': {
8             'base_url': os.environ.get('REI_BASE_URL', 'placeholder'),
9             'rope_path': os.environ.get('REI_ROPE_PATH', 'placeholder')
10        }
11    }
12
```

## \_\_init\_\_.py

```
app > 🐍 __init__.py > ...
You, a few seconds ago | 1 author (You)
1 from app.settings import get_settings
2
3 app_settings = get_settings()
4
```

## *resources.py*

```
app > 🐍 resources.py > ...
You, a few seconds ago | 1 author (You)
1 from flask_restful import Resource
2
3
4 from app import app_settings
5 from app.clients.rei import ReiClient
6
7
8 You, 24 minutes ago | 1 author (You)
9 class RopeCollection(Resource):
10
11     _clients_with_rope = [
12         ReiClient(**app_settings['rei'])
13     ]
14
15     def get(self):
16         ropes = RopeController().fetch_many(
17             self._clients_with_rope
18         )
19         return {'data': ropes}, 200
```

## *controllers/rope.py*

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
14         return [asdict(rope_model) for rope_model in rope_models]
```

app > models >  rope.py > ...

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

```
1 from dataclasses import dataclass
2
3
4 @dataclass
5 You, 3 hours ago | 1 author (You)
6 class RopeModel:
7     diameter: float
8     length: int
9     brand: str
10    color: str
11    price: float
```

## *models/rope.py*

app > clients >  rei.py > ...

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

```
1 import requests
2 from os import path
3 from typing import List
4
5 from app.clients.interface import HasRope
6 from app.models.rope import RopeModel
7
```

You, 29 minutes ago | 1 author (You)

```
9 class ReiClient(HasRope):
10
11     def __init__(self, base_url, rope_path):
12         self._rope_endpoint = path.join(base_url, rope_path)
13
14     def fetch_ropes(self) -> List[RopeModel]:
15         response = requests.get(self._rope_endpoint)
16         response_dict = response.json()
17         rope_models = []
18         for rope in response_dict.get('data', []):
19             rope_models.append(
20                 RopeModel(
21                     rope.get('diameter'),
22                     rope.get('length'),
23                     rope.get('brand'),
24                     rope.get('color'),
25                     rope.get('price')
26                 )
27             )
28         return rope_models
29
```

*clients/rei.py*

*clients/interface.py*

app > clients >  interface.py > ...

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

```
1 from abc import ABC, abstractmethod
2 from typing import List
3
4 from app.models.rope import RopeModel
5
6
```

You, 3 hours ago | 1 author (You)

```
7 class HasRope(ABC):
8
9     @abstractmethod
10    def fetch_ropes(self) -> List[RopeModel]:
11        pass
12
```

*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)
1  from flask_restful import Resource
2
3  from app import app_settings
4  from app.clients.rei import ReiClient
5  from app.controllers.rope import RopeController
6
7
You, 24 minutes ago | 1 author (You)
8  class RopeCollection(Resource):
9
10     _clients_with_rope = [
11         ReiClient(**app_settings['rei'])
12     ]
13
14     def get(self):
15         ropes = RopeController().fetch_many(
16             self._clients_with_rope
17         )
18         return {'data': ropes}, 200
19
```

```
tests > app > 🐍 test_resources.py > ...
1  import json
2  import pytest
3
4  from app.create_app import create_app
5  from app.controllers.rope import RopeController
6  from app.resources import RopeCollection
7
8
9  @pytest.fixture
10 def test_client():
11     app = create_app()
12     with app.test_client() as client:
13         yield client
14
15
16 def test_get_ropes(mocker, test_client):
17     # arrange
18     test_ropes = [
19         {
20             'test rope': 'test rope 1'
21         },
22         {
23             'test rope': 'test rope 2'
24         }
25     ]
26     mock_rope_controller = mocker.patch.object(RopeController, 'fetch_many', return_value=test_ropes)
27
28     # act
29     actual = test_client.get('/ropes')
30
31     # assert
32     assert json.loads(actual.data) == {'data': test_ropes}
33     assert actual.status_code == 200
34
35     mock_rope_controller.assert_called_once_with(
36         RopeCollection._clients_with_rope
37     )
38
```

## # 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()
```

```
44 def test_rope_controller_fetch_many_2(mocker):
45     # arrange
46     test_client_with_rope_1 = mocker.MagicMock()
47     test_client_with_rope_2 = mocker.MagicMock()
48     test_rope_model_1 = RopeModel(
49         diameter=9.4,
50         length=70,
51         brand='mammut',
52         color='blue',
53         price=264.95
54     )
55     test_rope_model_2 = RopeModel(
56         diameter=9.9,
57         length=60,
58         brand='black diamond',
59         color='green',
60         price=199.95
61     )
62     test_rope_model_3 = RopeModel(
63         diameter=9.4,
64         length=40,
65         brand='edelrid',
66         color='pink',
67         price=199.95
68     )
69     test_rope_model_4 = RopeModel(
70         diameter=9.9,
71         length=60,
72         brand='mammut',
73         color='yellow',
74         price=264.95
75     )
```

**# 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**

```
76     mock_fetch_ropes_1 = mocker.patch.object(test_client_with_rope_1, 'fetch_ropes', return_value=[
77         test_rope_model_1,
78         test_rope_model_2
79     ])
80     mock_fetch_ropes_2 = mocker.patch.object(test_client_with_rope_2, 'fetch_ropes', return_value=[
81         test_rope_model_3,
82         test_rope_model_4
83     ])
84
85     # act
86     actual = RopeController().fetch_many([test_client_with_rope_1, test_client_with_rope_2])
87
88     # assert
89     assert actual == [
90         asdict(test_rope_model_1),
91         asdict(test_rope_model_2),
92         asdict(test_rope_model_3),
93         asdict(test_rope_model_4)
94     ]
95     mock_fetch_ropes_1.assert_called_once_with()
96     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)
1 import requests
2 from os import path
3 from typing import List
4
5 from app.clients.interface import HasRope
6 from app.models.rope import RopeModel
7
8
You, 29 minutes ago | 1 author (You)
9 class ReiClient(HasRope):
10
11     def __init__(self, base_url, rope_path):
12         self._rope_endpoint = path.join(base_url, rope_path)
13
14     def fetch_ropes(self) -> List[RopeModel]:
15         response = requests.get(self._rope_endpoint)
16         response_dict = response.json()
17         rope_models = []
18         for rope in response_dict.get('data', []):
19             rope_models.append(
20                 RopeModel(
21                     rope.get('diameter'),
22                     rope.get('length'),
23                     rope.get('brand'),
24                     rope.get('color'),
25                     rope.get('price')
26                 )
27             )
28         return rope_models
29
```

```
tests > app > clients > test_rei_client.py > ...
You, a few seconds ago | 1 author (You)
1 import pytest
2
3 from app.clients.rei import ReiClient
4 from app.models.rope import RopeModel
5
6
7 def test_rei_client_fetch_ropes(mocker):
8     # arrange
9     test_rope_1 = dict(
10         diameter=9.4,
11         length=70,
12         brand='mammut',
13         color='blue',
14         price=264.95
15     )
16     test_rope_2 = dict(
17         diameter=9.9,
18         length=60,
19         brand='black diamond',
20         color='green',
21         price=199.95
22     )
23     test_response_dict = {
24         'data': [
25             test_rope_1,
26             test_rope_2
27         ]
28     }
29     mock_response = mocker.MagicMock()
30     mocker.patch.object(mock_response, 'json', return_value=test_response_dict)
31     mock_requests_get = mocker.patch('app.clients.rei.requests.get', return_value=mock_response)
32
33     # act
34     client = ReiClient('test-rei-url.com', 'random-rope-path')
35     actual = client.fetch_ropes()
36
37     # assert
38     assert actual == [
39         RopeModel(**test_rope_1),
40         RopeModel(**test_rope_2)
41     ]
42     mock_requests_get.assert_called_once_with(
43         'test-rei-url.com/random-rope-path'
44     )
```

# 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 → create a flowchart in your mind of when to use what mocking tools so that your testing process becomes a **standard operating procedure**

*Thx for attending the presentation!*



All code samples & these slides available at:  
[github.com/chord-memory/python-mocking-presentation](https://github.com/chord-memory/python-mocking-presentation)

Listen to Namespace Podcast on any podcast app, YouTube, or our website  
[namespace-pod.buzzsprout.com](https://namespace-pod.buzzsprout.com)

Feel free to connect w me on LinkedIn or send remarks to  
[jordancahill88@gmail.com](mailto:jordancahill88@gmail.com)