Getting started with Mocks

Ester Ramos Carmona

Kobalt Music Group

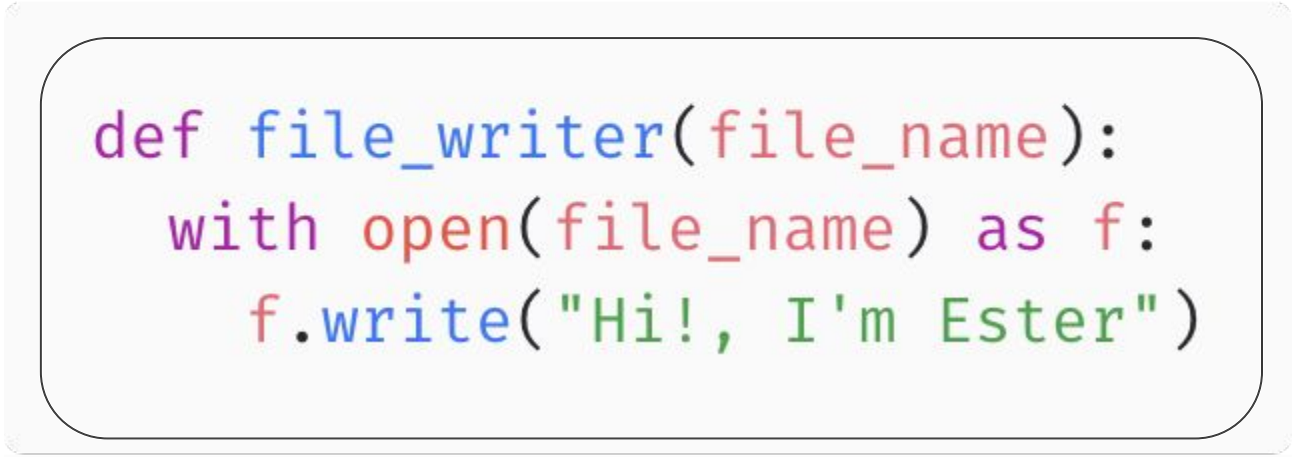
Kobalt

AWAL

A KOBALT COMPANY

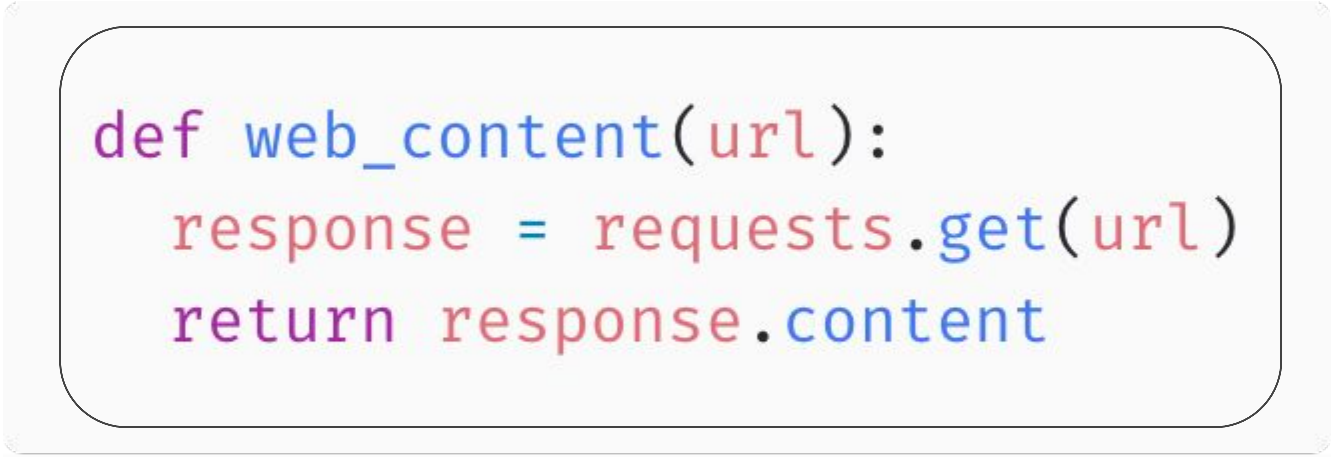
Why use Mocks?

Impossible to test?



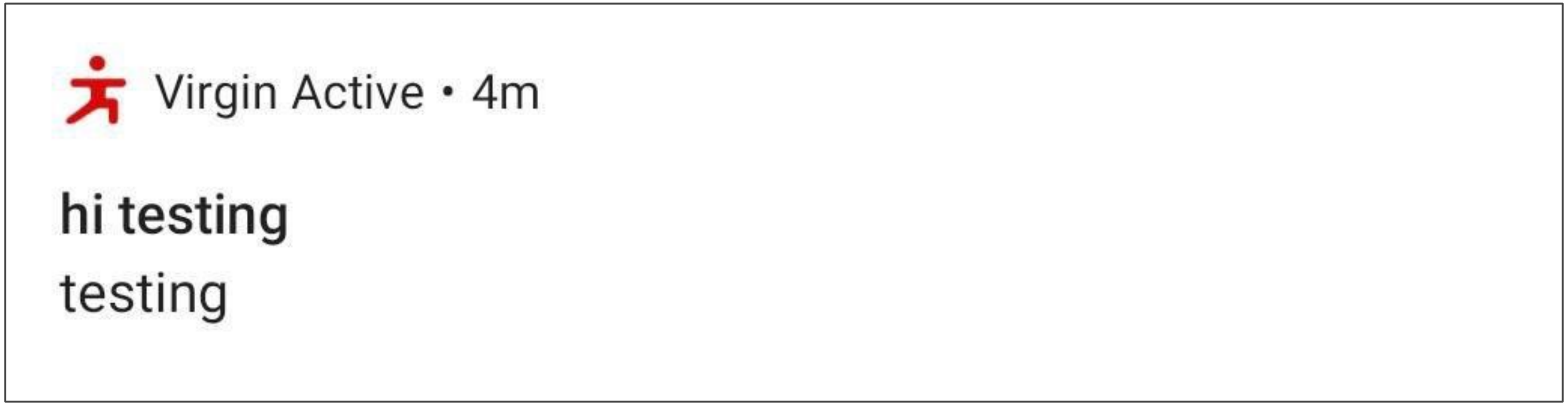
```
def file_writer(file_name):
  with open(file_name) as f:
    f.write("Hi!, I'm Ester")
```

Impossible to test?



```
def web_content(url):
  response = requests.get(url)
  return response.content
```

Impossible to test?


Virgin Active • 4m

hi testing

testing

What are Mocks?

Creating doubles

A mock creates an object that seems the same as the original but we control its behaviour.

```
from unittest.mock import Mock
```

Controlling behaviour

```python
def function(arguments, slack_channel):
    ...
    slack_channel(result)
    ...
```

```python
from unittest.mock import Mock
```

Controlling behaviour

```
def function(arguments, slack_channel):
    ...
    slack_channel(result)
    ...
```

```
double = Mock(return_value="SUCCESS")
```

```
from unittest.mock import Mock
```

Controlling behaviour

```python
def function(arguments, slack_channel):
    ...
    slack_channel(result)
    ...
```

```python
double = Mock(return_value="SUCCESS")
```

```python
function(arguments, double)
```

```python
from unittest.mock import Mock
```

Controlling the behaviour

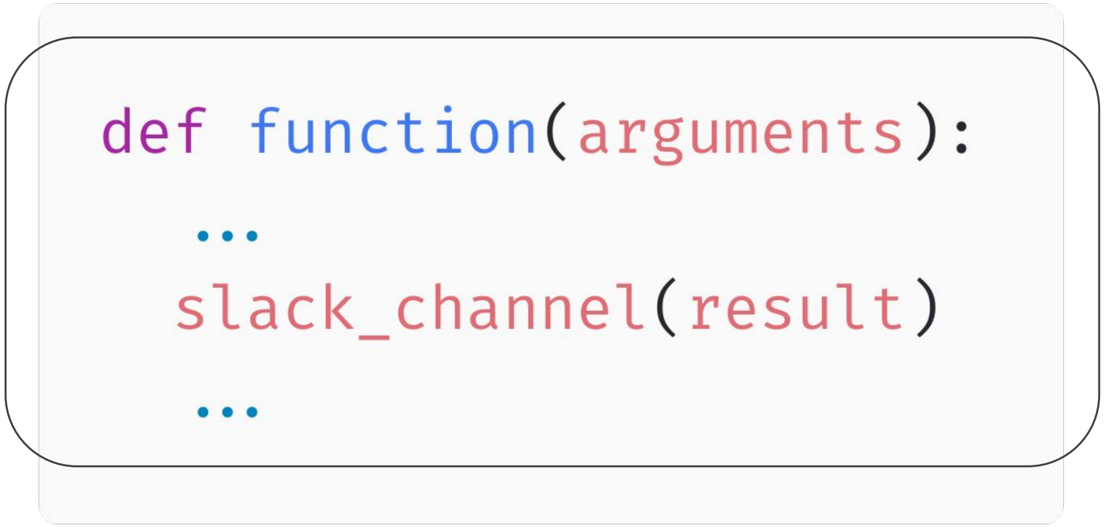
```python
def function(arguments, slack_channel):
    ...
    slack_channel(result)
    ...
```

```python
double = Mock(return_value="SUCCESS")
```

```python
function(arguments, double)
```

```python
from unittest.mock import Mock
```

If the object is inside the function



```
def function(arguments):
    ...
    slack_channel(result)
    ...
```

Patch is your friend

If the object is inside the function

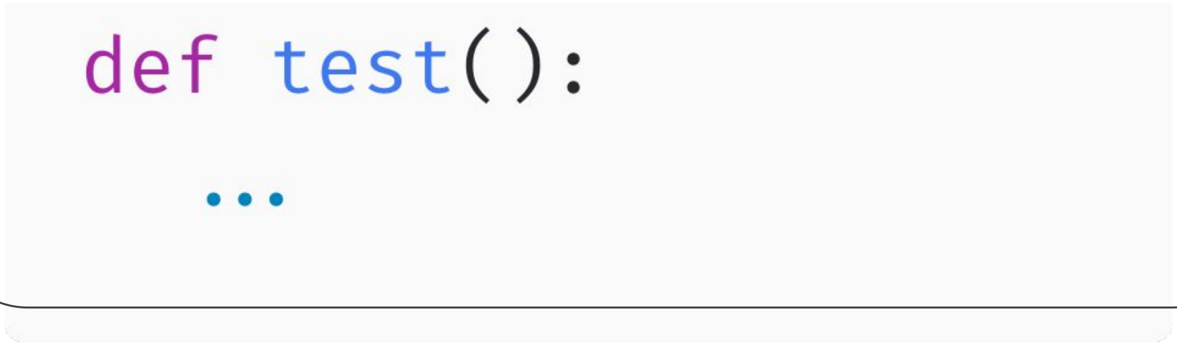
Patch replaces all target objects inside a function with a MagicMock (or inside a context manager)

The basic principle is that you patch where an object is *looked up*, which is not necessarily the same place as where it's defined

```
unittest.mock.patch(package.modulo.target)
```

Patch is your friend

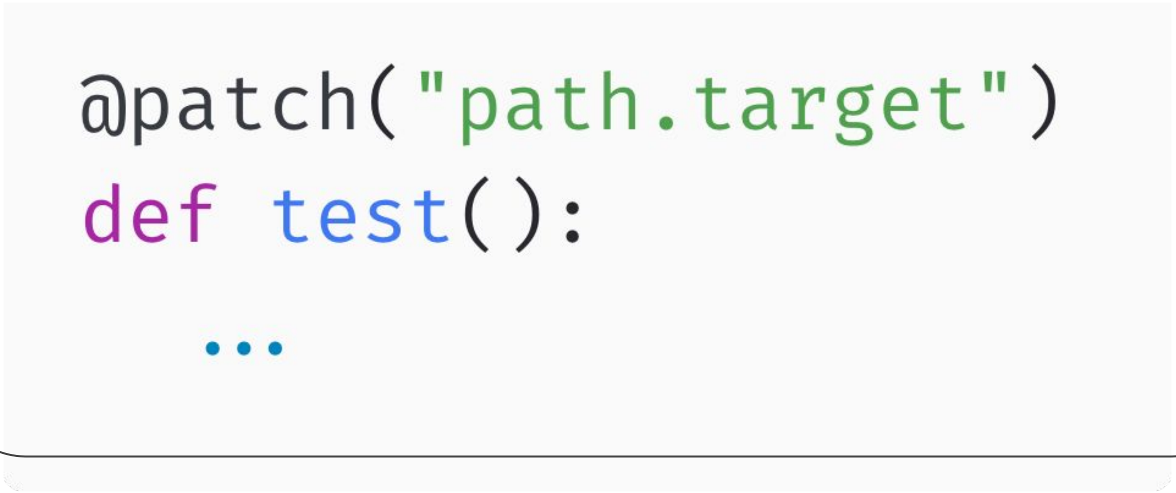
If the object is inside the function



```python
def test():
    ...
```

Patch is your friend

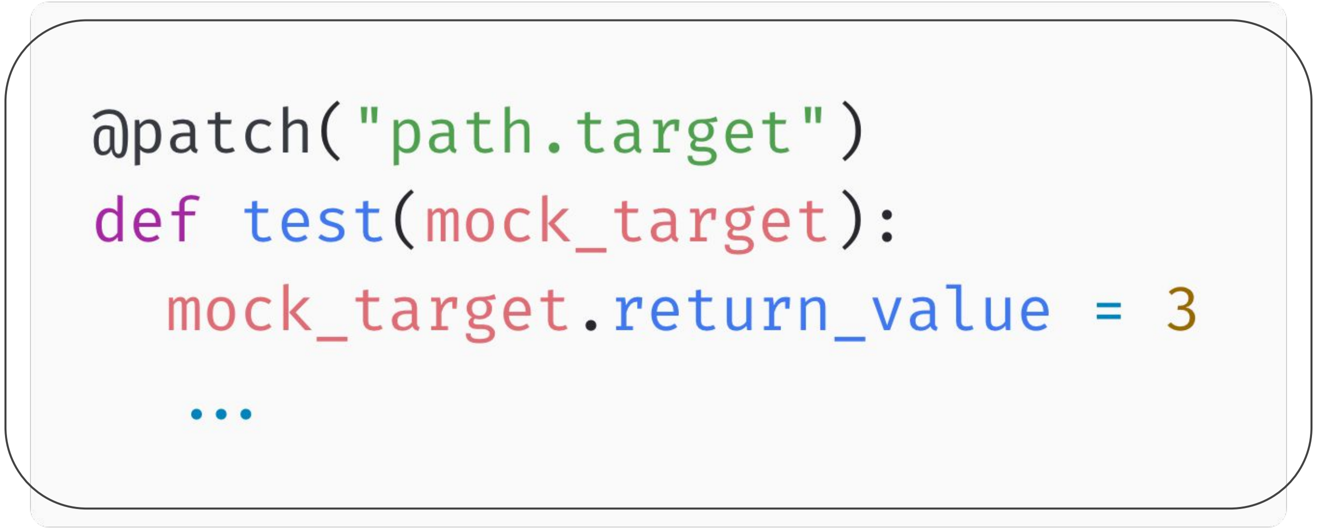
If the object is inside the function



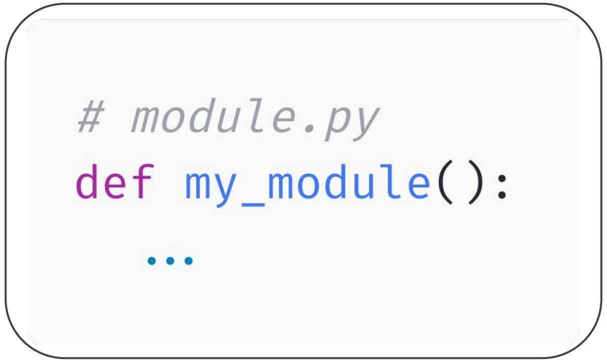
```python
@patch("path.target")
def test():
    ...
```

Patch is your friend

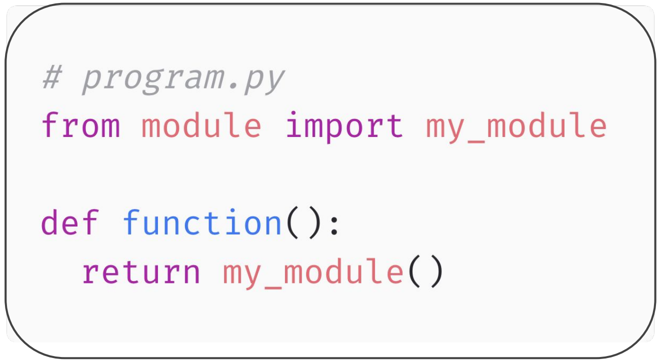
If the object is inside the function



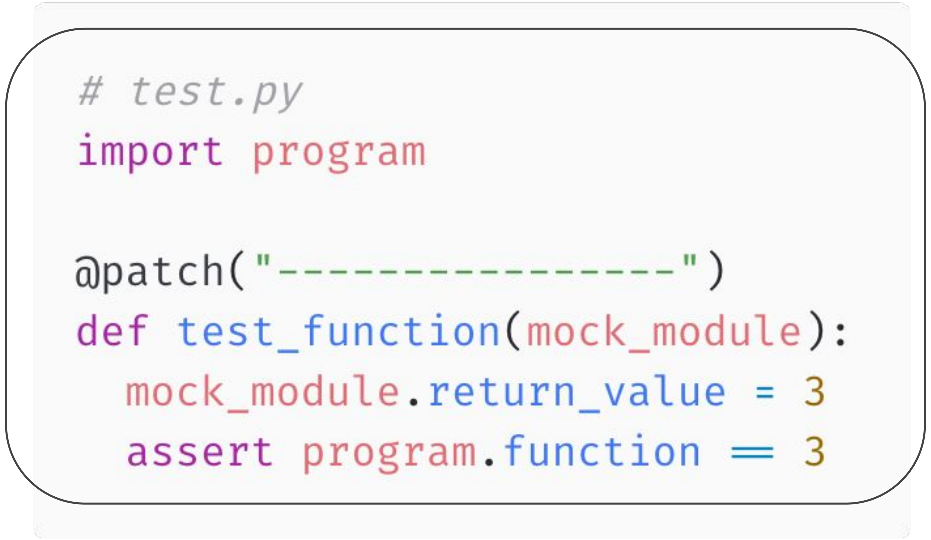
```python
@patch("path.target")
def test(mock_target):
  mock_target.return_value = 3
   ...
```



```python
# module.py
def my_module():
    ...
```



```python
# program.py
from module import my_module

def function():
  return my_module()
```



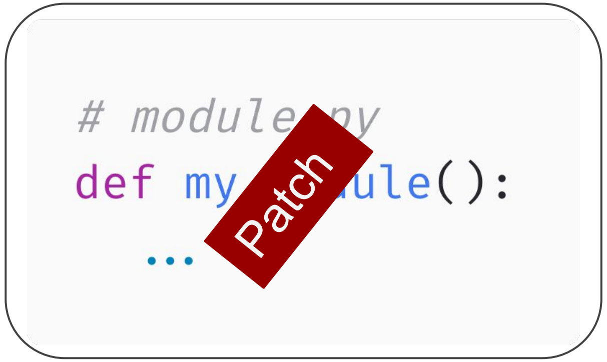
```python
# test.py
import program

@patch("----------------")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```

```python
# module.py
def my_module():
    ...
```

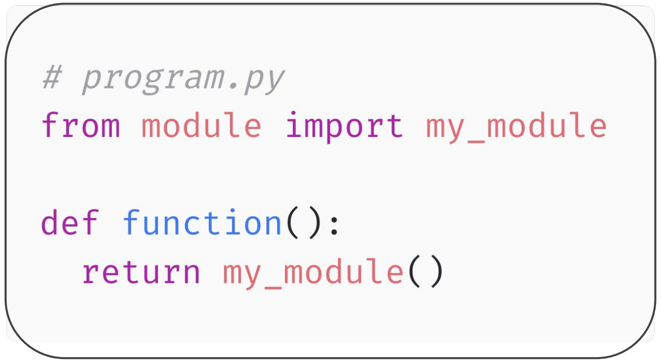
Patch

```python
# program.py
from module import my_module

def function():
  return my_module()
```

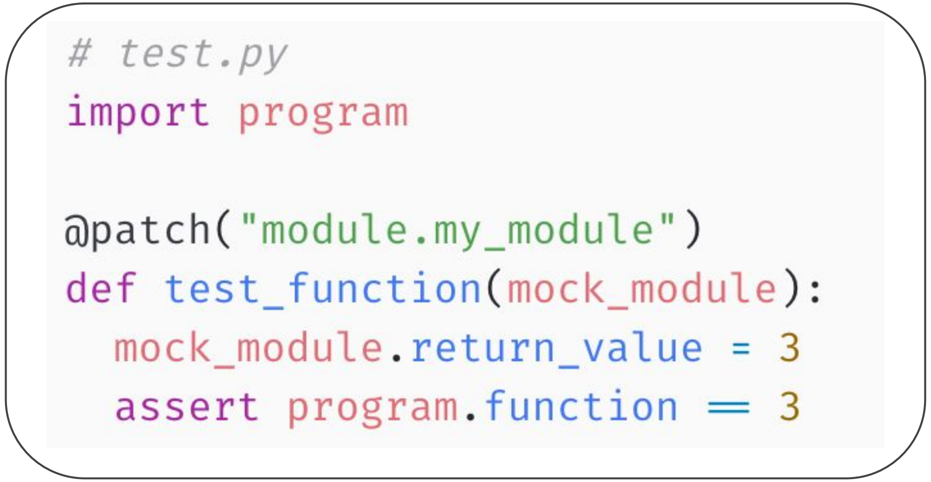
```python
# test.py
import program

@patch("module.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```



```
# module  py
def my     ule():
    ...
```



```
# program.py
from module import my_module

def function():
  return my_module()
```



```
# test.py
import program

@patch("module.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```

```
# mo
def                ():
    ..
```

```python
# program.py
from module import my_module

def function():
  return my_module()
```

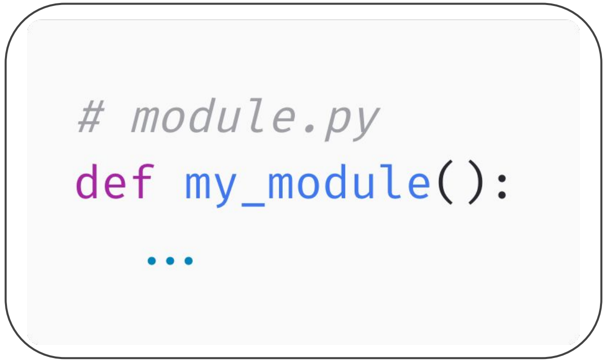
```python
# test.py
import program

@patch("module.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```

```python
# module.py
def my_module():
    ...
```

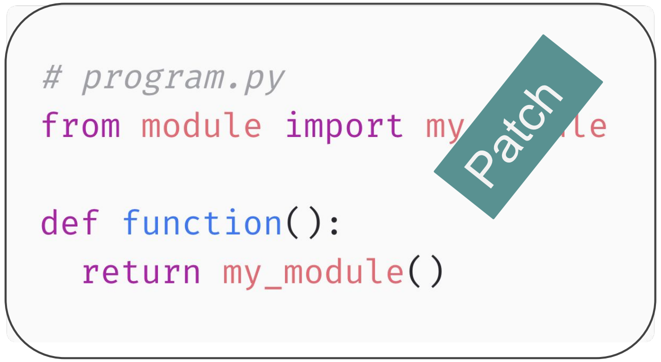
```python
# program.py
from module import my_module

def function():
  return my_module()
```

Patch

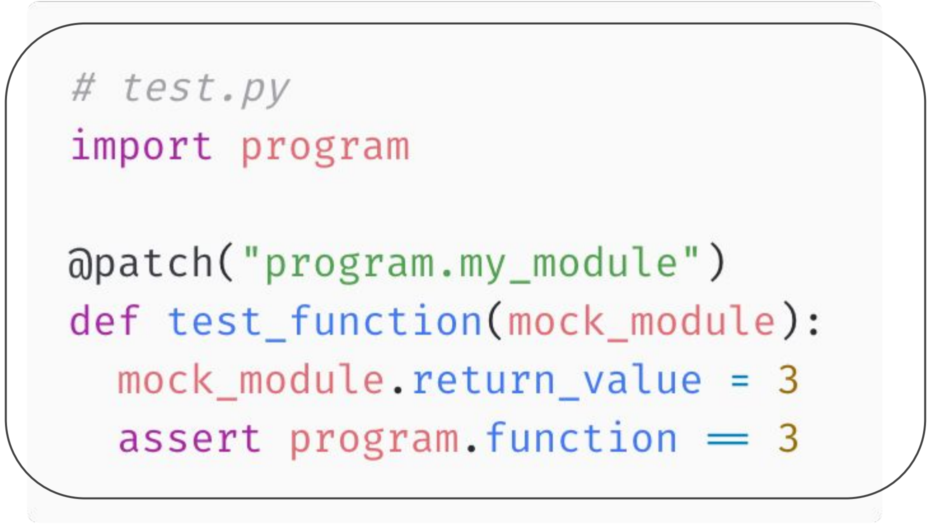
```python
# test.py
import program

@patch("---------------")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```



```python
# module.py
def my_module():
    ...
```



```python
# program.py
from module import my[illegible]le

def function():
  return my_module()
```



```python
# test.py
import program

@patch("program.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```

```python
# module.py
def my_module():
    ...
```

```python
# program.py
from module import my[illegible]le

def function():
  return my_module()
```

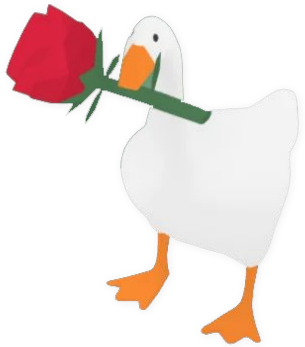
Patch

```python
# test.py
import program

@patch("program.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```

```python
# module.py
def my_module():
  ...
```

```python
# program.py
from module import my Patch le

def function():
  return my_module()
```

```python
# test.py
import program

@patch("program.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```

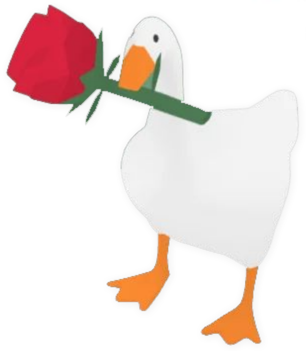


```python
# module.py
def my_module():
    ...
```

```python
# program.py
from module import my[illegible]le

def function():
  return my_module()
```

Patch

```python
# test.py
import program

@patch("program.my_module")
def test_function(mock_module):
  mock_module.return_value = 3
  assert program.function == 3
```



Patch: more ways to use it

Context Manager

```
def test_function():
  with patch('path.target') as mock_target:
    mock_target.return_value = 3
     ...
```

Side effects

Exceptions

Good for exception testing.

Be careful when testing very generic exceptions.

```
mock = Mock()
mock.side_effect = Exception("¯\_(ツ)_/¯")
mock()

Traceback (most recent call last):
    ...
 Exception: ¯\_(ツ)_/¯
```

Side effects

Multiple function calls

The basic mock returns the same value every time

With ‘side_effect’ you can create different responses for each call.

```
mock = Mock()
mock.side_effect = [3, 2, 1]
mock()
```

Side effects

Multiple function calls

The basic mock can only be called once but with ‘side_effect’ you can create different responses for each call.

```
mock = Mock()
mock.side_effect = [3, 2, 1]
mock()
   3
```

Side effects

Multiple function calls

The basic mock can only be called once but with ‘side_effect’ you can create different responses for each call.

```
mock = Mock()
mock.side_effect = [3, 2, 1]
mock()
  3
mock()
  2
```

Side effects

Multiple function calls

The basic mock can only be called once but with ‘side_effect’ you can create different responses for each call.

```
mock = Mock()
mock.side_effect = [3, 2, 1]
mock()
  3
mock()
  2
mock()
  1
```

Dependency Injection

It's best if the target you want to mock is one of the dependencies.

You don't need to know how that object is implemented.

1.

```
def do_thing(url):
   ...
  response = requests.get(url)
   ...
```

2.

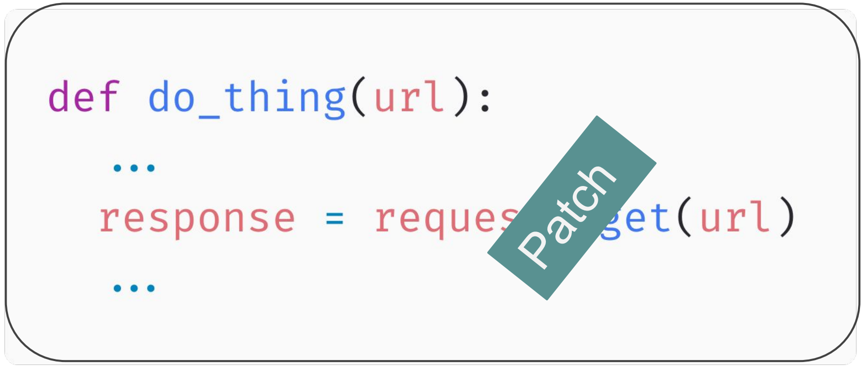
```
def do_thing(url, session):
   ...
  response = session.get(url)
   ...
```

Dependency Injection

It’s best if the target you want to mock is one of the dependencies.

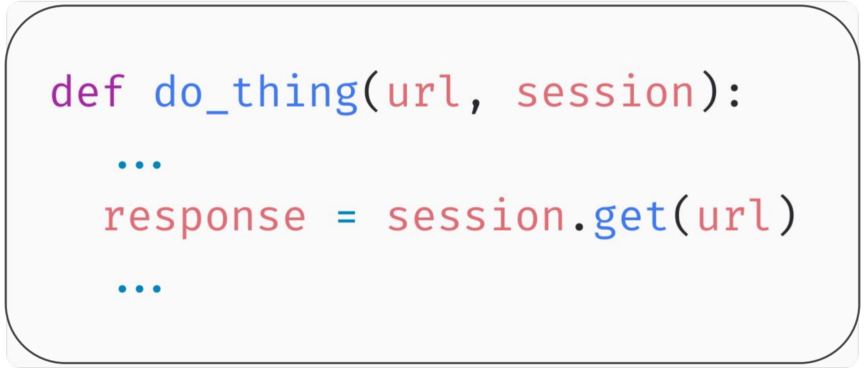
You don’t need to know how that object is implemented.

1.



```
def do_thing(url):
    ...
    response = reques    get(url)
    ...
```

2.



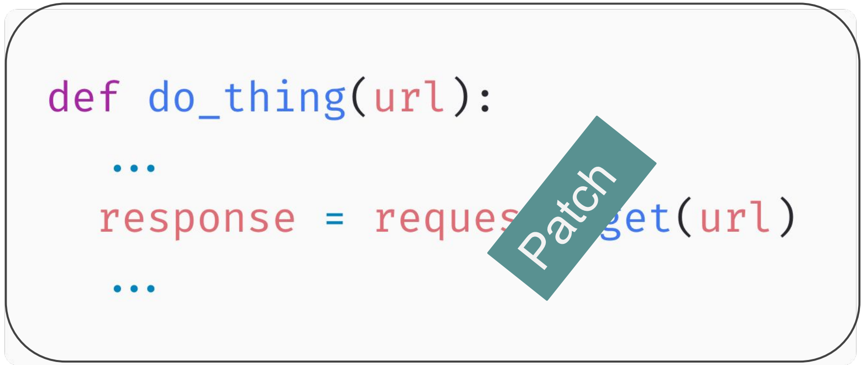
```
def do_thing(url, session):
    ...
    response = session.get(url)
    ...
```

Dependency Injection

It’s best if the target you want to mock is one of the dependencies.

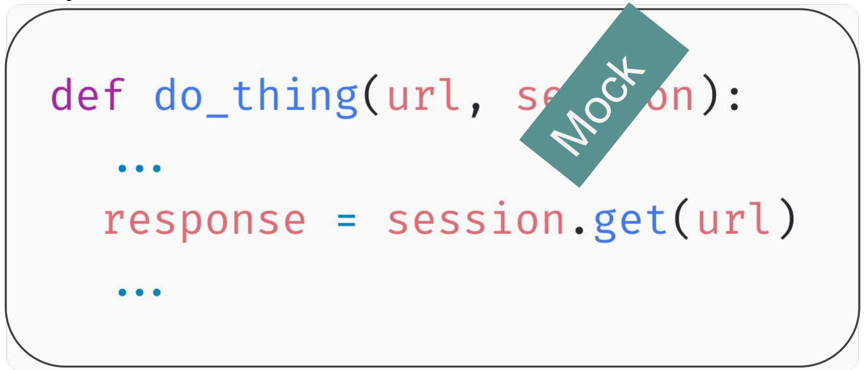
You don’t need to know how that object is implemented.

1.



```
def do_thing(url):
    ...
    response = reques   get(url)
    ...
```

2.



```
def do_thing(url, se     on):
    ...
    response = session.get(url)
    ...
```

Dependency Injection

Using ‘patch’ means that you need to know what’s inside that function.

1.

```
def do_thing(url):
    ...
    response = requests.get(url)
    ...
```

Mocking a dependency allows testing without having to know what’s happening inside the function.

2.

```
def do_thing(url, session):
```

[redacted]

Dependency Injection

Using ‘patch’ means that you need to know what’s inside that function.

Mocking a dependency allows testing without having to know what’s happening inside the function.

1.

```
def do_thing(url):
    ...
    response = requests.get(url)
    ...
```

2.

```
def do_thing(url, session):
```

[redacted]

Dependency Injection

Using ‘patch’ means that you need to know what’s inside that function.

1.

```
def do_thing(url):
  ...
  response = requests.get(url)
  ...
```

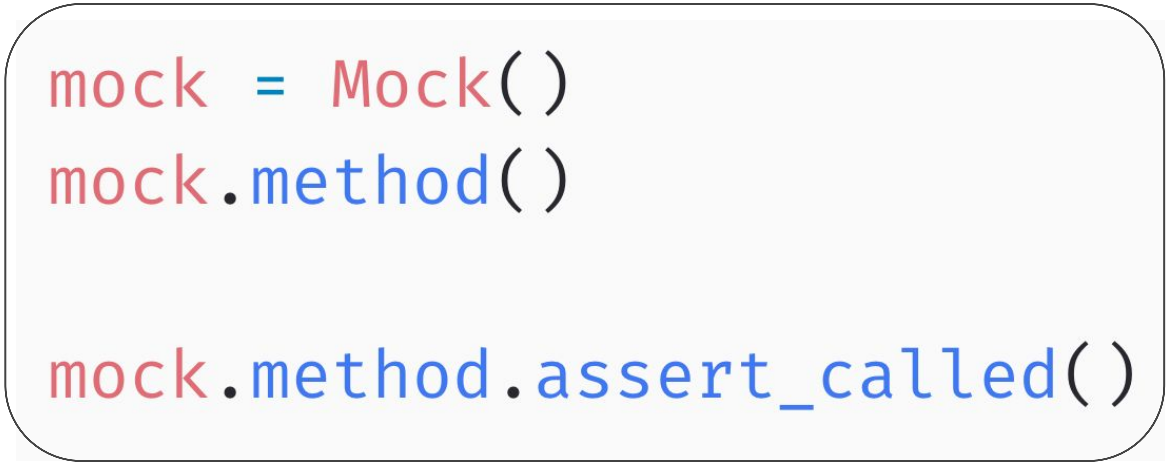
Mocking a dependency allows testing without having to know what’s happening inside the function.

2.

```
def do_thing(url, session):
```

Asserts

You can also check if the method of a mocked target has been called.



```
mock = Mock()
mock.method()

mock.method.assert_called()
```

Asserts

Or if the call has been done only once

```
mock = Mock()
mock.method()

mock.method.assert_called_once()
```

Asserts

Or if the call has been done with specific arguments.

```
mock = Mock()
mock.method(1, 2, 3)

mock.method.assert_called_with(1, 2, 3)
```

```python
from unittest.mock import call
```

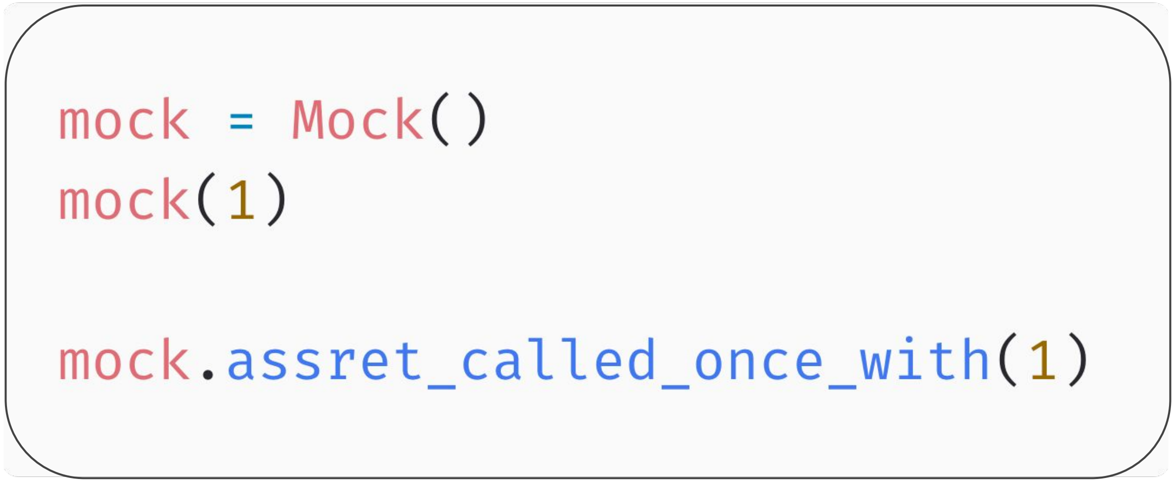
Asserts

You can also check if the mock is called several times.

```python
mock = Mock()
mock(1)
mock(2)
mock(3)

calls = [call(1), call(2), call(3)]

mock.assert_has_calls(calls)
```

Spec and Autospec

Mock creates a method and calls it’s own assert method the same way and this can create issues.



```
mock = Mock()
mock(1)

mock.assret_called_once_with(1)
```

Spec

If we use ‘spec’, the mock will be only able to access the existing attributes of the chosen class.

```
mock = Mock(spec=os.listdir)
mock(".")
mock.assret_called()
```

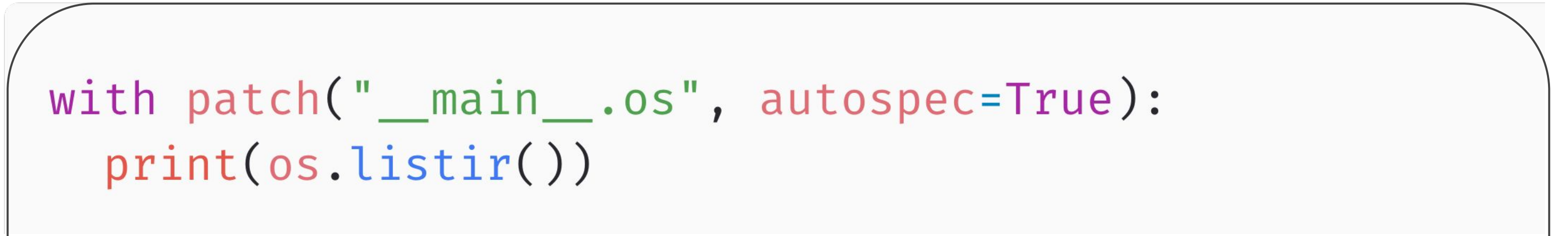
Spec

If we use ‘spec’, the mock will be only able to access the existing attributes of the chosen class.

```
mock = Mock(spec=os.listdir)
mock(".")
mock.assret_called()

Traceback (most recent call last):
   ...
AttributeError: Mock object has no attribute 'assret_called"
```

Autospec

‘Autospec’ will automatically check the attributes of the patched class.



```python
with patch("__main__.os", autospec=True):
  print(os.listir())
```

Autospec

‘Autospec’ will automatically check the attributes of the patched class.

```
with patch("__main__.os", autospec=True):
  print(os.listir())

Traceback (most recent call last):
   ...
AttributeError: Mock object has no attribute 'listir'
```

Mock vs MagicMock

MagicMock has default implementation of most of the magic methods.

You run the risk of a test *succeeding* when it should have failed.

Tests should be minimal and mock objects should be minimally functional so that you are sure exactly what you’re testing.

Mock vs MagicMock

Only use MagicMock if you want to implement magic methods

Gracias