Stub

Mock

Patch

# Bailo con tu sombra

María Andrea Vignau

**@mavignau** github: **marian-vignau**

# Bailo con tu sombra

- Porqué usar simuladores
- Cómo usar **patch**.
- Simuladores: **stub** y **mock**.
- Librerías para desarrollo web

# Introducción

Razones para testear

Testeo automatizado

Porqué usar simuladores

# Introducción

- Nunca (casi) un código funcionará de entrada
- Necesitamos testear
- Al crecer el código, hay más que modificar, mantener y volver a probar.

# Introducción

Tipos de tests automáticos

- **De integración**: una funcionalidad completa del sistema
- **Unitarios**: funciones específicas

# Simuladores - ¿Porqué?

- Testear partes del sistema por separado
- APIs de terceros, hardware especial.
- Operaciones de entrada y salida
- Eventos improbables

# Simuladores

- Suplantan funciones, clases, objetos, bibliotecas
- Predecibles, Repetibles, Veloces
- Livianos y baratos

Errores comunes

Alcance

Desventajas

Dependencia inversa

# Parchear

# Parchear – Errores comunes

- **SI**

  Elegir parchear el objeto importado

- **NO**

  Parchear desde dónde se hubiera importado

# Parchear – Errores comunes

Si usa en **b**.py

```
from a import Class
```

```
@patch("b.Class")
```

```
import a
```

```
@patch("a.Class")
```

Decorador

Manejador de contexto

Manualmente

# Parchear - Alcance

# Parchear - Decorador

```
@patch('module.ClassB')
@patch('module.functionA')
def test_some_func(self, mock_A, mock_B):
```

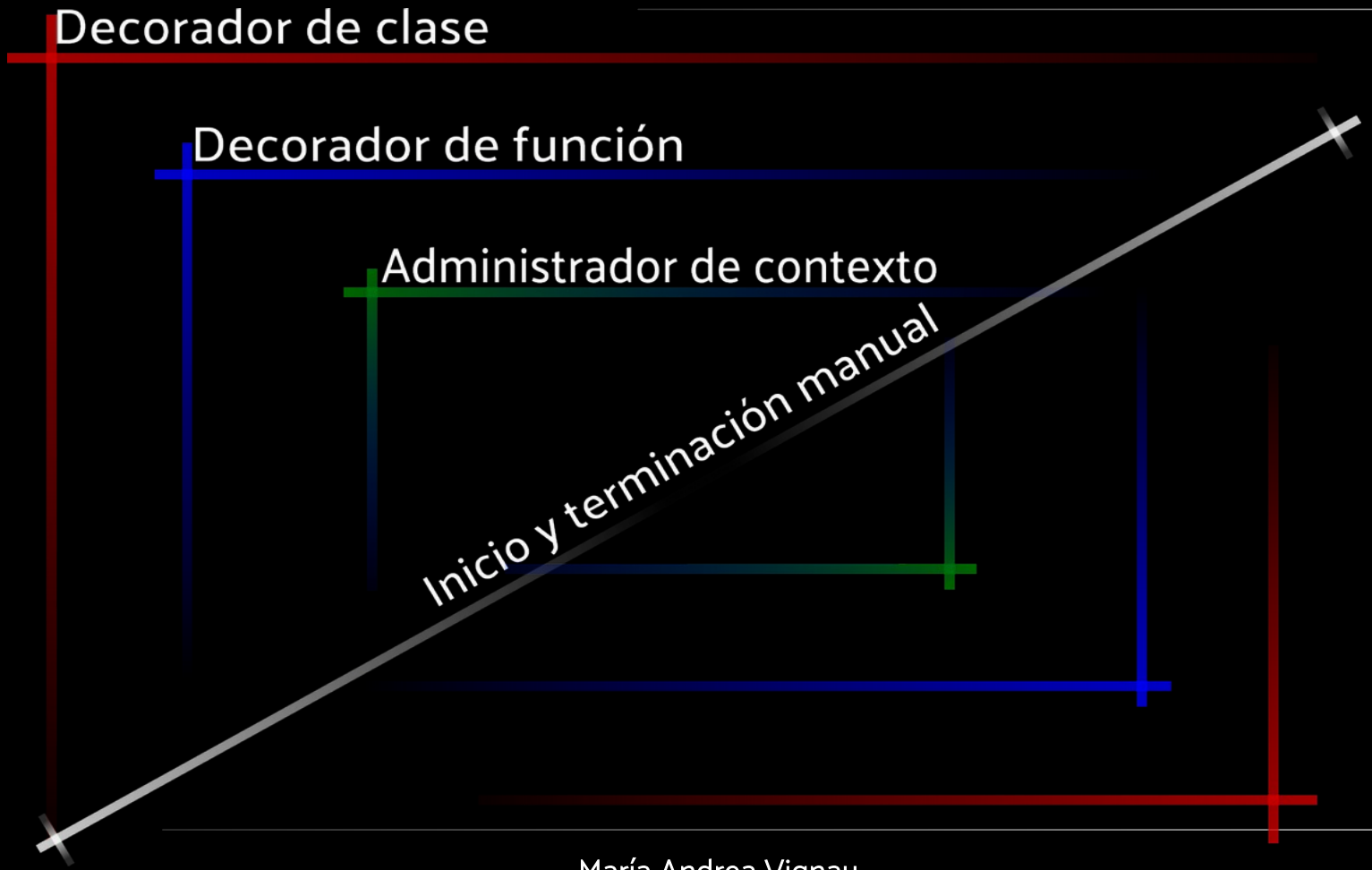
# Parchear – Administrador de contexto

```
import client
with patch('client.stats') as mock:
      mock.return_value=’some’
      response = client.get('/stats/')
```

# Parchear - Manualmente

```
patcher = patch('package.module.ClassName')
from package import module
original = module.ClassName
new_mock = patcher.start()
patcher.stop()
```

# Parchear - Alcances

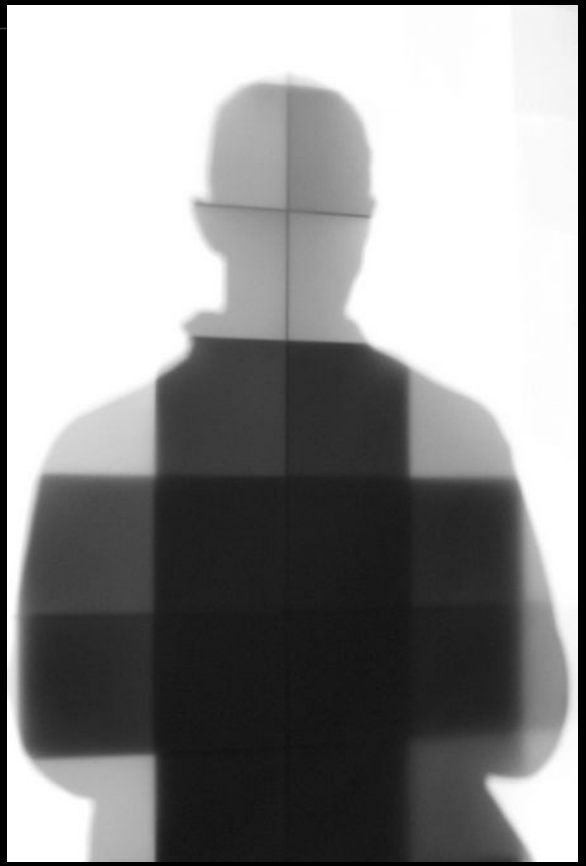


# Parchear – Desventajas

- Atamos fuertemente el test al código probado.
- Introduce modificaciones en tiempo de ejecución
- Complejo
- Alto costo

Concepto

Ejemplo



# Inyección de dependencias

# Dependencia inversa

- Ingresamos las clases, objetos, etc que necesite un trozo de código como argumento
- Podemos reemplazar directamente

# Stub

Pequeñas funciones o fragmentos que imitan al código

```
class StubAnimal:
  def patas(self):
    return 4
```

# Dependencia inversa

```
from A import Animal

class Perro(Animal):

    def __init__(self):


mascota = Perro()
```

```
from A import Animal

class Perro():

    def __init__(self, class):


mascota = Perro(class=Animal)
```

# Dependencia inversa

Usar patch

```
from B import Perro
class StubAnimal:
    def patas(self):
        return 4


mascota = Perro(class=StubAnimal)
```

MagicMock vs Mock

Return value

Side Effect

Specs

# Simuladores: Mocks

# Mocks – MagicMock

MagicMock es un objeto Mock pero con los atributos mágicos ya incluidos.

```
» mock = MagicMock()
» int(mock)
1
» len(mock)
0
» list(mock)
[]
» object() in mock
False
```

# Mocks – Return Value

- Podemos establecer el valor que retornará el mock al ejecutarse.

```
> mock_fn.return_value = “valor deseado”
> mock_fn()
“valor deseado”
```

# Mocks – Side effect

Puede usarse para devolver excepciones o valores múltiples.

```
> mock.side_effect = [5, 4, 3, 2, 1]
> mock(), mock(), mock()
(5, 4, 3)
> mock.side_effect = Exception('Boom!')
```

# Mocks – Specs

- Mocks, por su diseño, nunca devuelven ERROR si accedemos a un atributo que no existe.

  Crea el mismo.

- Esto puede dar lugar a una **falsa seguridad**

# Mocks – Specs

- **Spec** permite copiar los atributos de una clase, y devolverá error si accedemos alguna que no existe.
- **Autospec** además hace spec de los objetos internos, de forma recursiva

# Mocks - Specs

```
> def function(a, b, c): pass
> mock_function = create_autospec(function, return_value='llamado ok')
> mock_function(1, 2, 3)
'llamado ok'
> mock_function('wrong arguments')
TypeError: <lambda>() takes exactly 3 arguments (1 given)
```

# Mocks – Espías (wrap)

- Al hacer reemplazo por parcheo, se llama realmente a la función.
- Útil para que el mock registre los argumentos de las llamadas especificadas.

# Mock - Aserciones

Sólo llamadas

Ver argumentos

# Mock - Aserciones

**`assert_called`:** Fué llamado

**`assert_called_once`:** Llamado una vez

**`assert_not_called`:** Nunca llamado

# Mock – Aserciones con argumentos

**`assert_called_with:`** Último llamado de tal forma

**`assert_called_once_with:`** Llamado sólo de tal forma

**`assert_any_call:`** Alguna vez llamado de tal forma.

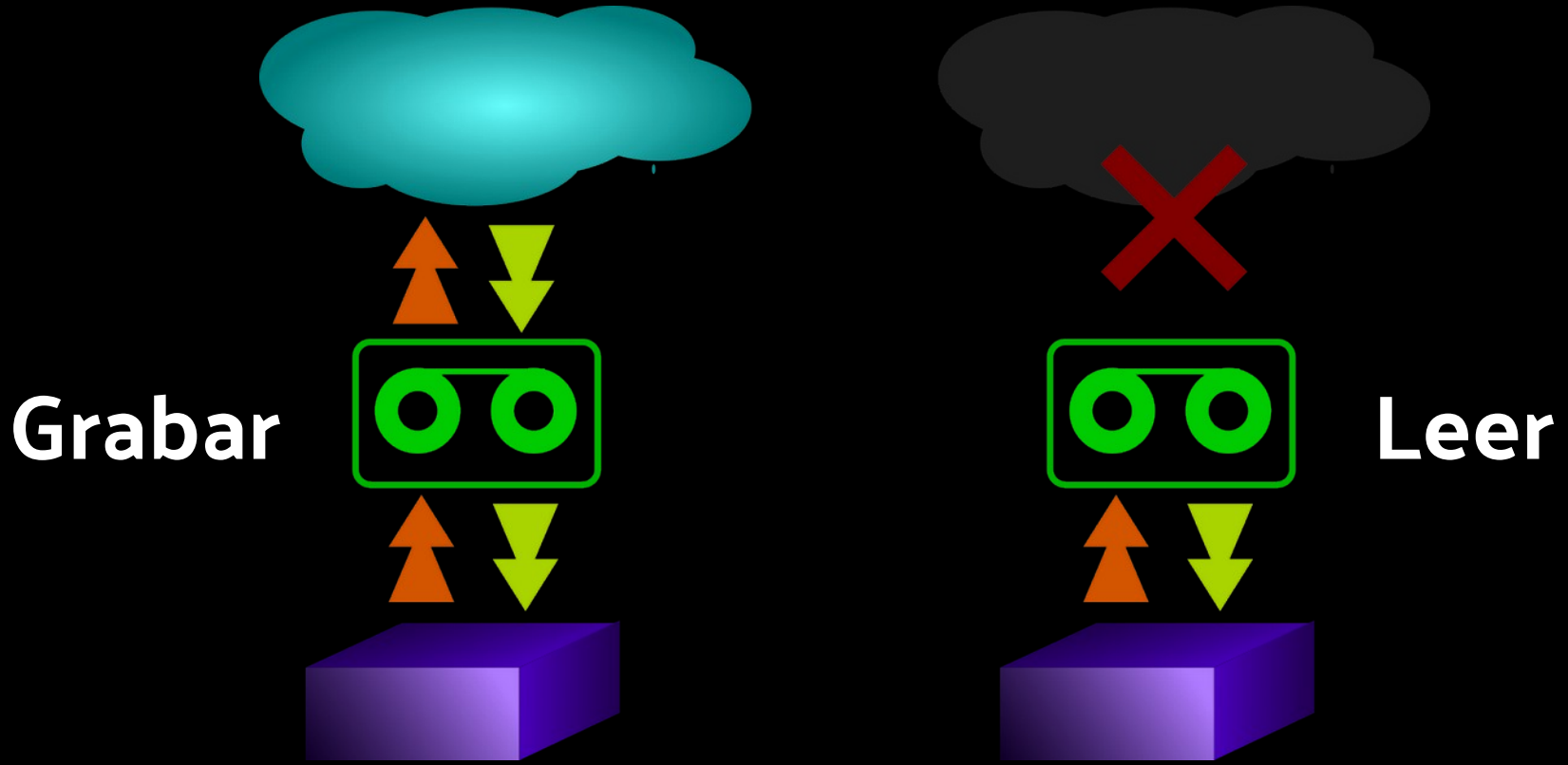
**`assert_has_calls:`** Que cumpla esta lista de llamados

VCR

Moto

# Especiales para internet

# VCR – Un grabador



# VCR – Un grabador

- ¿Porqué? Velocidad o Integración contínua **CI**.
- Graba una o más interacciones.

```
@pytest.mark.vcr()
def test_one():
    response = urlopen('http://www.one.org').read()
```

# Moto – Simular AWS Boto

```
import boto3

class MyModel(object):

    def save(self):

        s3 = boto3.client('s3',...)

        s3.put_object(...)
```

```
from moto import mock_s3

from mymodule import MyModel

@mock_s3

def test_my_model_save():

    model_instance = MyModel()

    model_instance.save()
```

# Bailo con tu sombra

- Para qué simular
- Como usar: patch.
- Simuladores: stub y mock.
- Especiales para internet

Contacto

mavignau

marian-vignau

# Bailo con tu sombra

**@mavignau** github: **marian-vignau**