Introdução ao Pytest

Live de Python # 167

Roteiro

1. Pytest
Uma introdução e instalação

2. Testes
Anatomia de um teste e diferentes tipos de asserts

3. Mark
Marcações, argumentos e metadados

4. Fixtures
Montado e desmontando estruturas para os testes

picpay.me/dunossauro

apoia.se/livedepython

PIX

Ajude o projeto

Ademar Peixoto, Alex Lima, Alex Lopes, Alexandre Harano, Alexandre Santos, Alexandre Tsuno, Alexandre Villares, Alynne Ferreira, Alysson Oliveira, Amaziles Carvalho, André Rocha, Arnaldo Turque, Bruno Batista, Bruno Oliveira, Caio Nascimento, Carlos Chiarelli, César Almeida, Davi Ramos, David Kwast, Diego Guimarães, Dilenon Delfino, Douglas Bastos, Elias Soares, Eugenio Mazzini, Everton Alves, Fabiano Gomes, Fabio Barros, Fabio Castro, Fabrícia Diniz, Fabrício Coelho, Fábio Serrão, Gabriel Simonetto, Gabriel Soares, Gabriela Santiago, Geandreson Costa, Guilherme Castro, Guilherme Felitti, Guilherme Marson, Guilherme Ostrock, Gustavo Chacon, Henrique Machado, Hélio Neto, Israel Fabiano, Italo Silva, Johnny Tardin, Jonatas Leon, Jonatas Oliveira, Jorge Plautz, Jose Mazolini, José Prado, João Lugão, João Schiavon, Juan Gutierrez, Julio Silva, Jônatas Silva, Júlia Kastrup, Kaneson Alves, Leonardo Cruz, Leonardo Galani, Leonardo Mello, Lidiane Monteiro, Lorena Ribeiro, Lucas Barros, Lucas Mello, Lucas Mendes, Lucas Teixeira, Lucas Valino, Luciano Ratamero, Maiquel Leonel, Maiquel Leonel, Marcela Campos, Marcelo Rodrigues, Maria Clara, Marina Passos, Matheus Vian, Melissa Mendonça, Natan Cervinski, Nicolas Teodosio, Patric Lacouth, Patricia Minamizawa, Patrick Gomes, Paulo Tadei, Pedro Pereira, Peterson Santos, Rafael Lino, Reinaldo Silva, Revton Silva, Rodrigo Ferreira, Rodrigo Mende, Rodrigo Vaccari, Ronaldo Silva, Sandro Mio, Silvio Xm, Thiago Araujo, Thiago Borges, Thiago Bueno, Tyrone Damasceno, Victor Geraldo, Vinícius Bastos, Vinícius Ferreira, Vítor Gomes, Wendel Rios, Wesley Mendes, Willian Lopes, Willian Lopes, Willian Rosa, Wilson Duarte, Érico Andrei

Obrigado você

Apresentação do

Pytest

Pytest

Pytest é um framework em python dedicado a testes. Uma alternativa mais "pythonica" ao unittest.

- Simples
- Escalável
- Rico em plugins
- Suporte ao pypy
- Primeira release em 2009
- Atualmente na versão 6.2.4

pip install pytest

poetry add pytest

Instalação

Testes

Uma introdução ao pytest

Qual a cara de um teste mínimo?

```python
# test_pytest.py
def test_meu_primeiro_teste():
    assert 1 == 1
```

Qual a cara de um teste mínimo?

Prefixo

```
1   # test_pytest.py
2   def test_meu_primeiro_teste():
3       assert 1 == 1
```

Qual a cara de um teste mínimo?

Nome do teste

```
1   # test_pytest.py
2   def test_meu_primeiro_teste():
3       assert 1 == 1
```

Qual a cara de um teste mínimo?

Com o que vamos praticar?

Vamos fazer uma adaptação abrasileirada do BuzzFizz que a Paty Mori apresentou em um tutorial de TDD no Capiyra de 2019.

Como funciona?

Vamos pegar um número inteiro, por exemplo: "1".

Quando o número foi **múltiplo de 3**, deve responder "**Queijo**"

Quando o número foi **múltiplo de 5**, deve responder "**Goiabada**"

Quando o número foi **múltiplo de 3 e de 5**, deve responder "**Romeu e Julieta**"

Mas então ...

Com esse problema simples e divertido vamos aprender a usar as features básicas do pytest e com isso ir evoluindo e aprofundando nosso conhecimento sobre testes e sobre como fazer isso no pytest.

Como executar os testes?

```
$ pytest test_pytest.py
============================= test session starts ==============================
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 1 item

test_pytest.py .                                                         [100%]

============================== 1 passed in 0.01s ===============================
```

Como executar os testes?

```
$ pytest test_pytest.py
============================= test session starts ==============================
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 1 item

test_pytest.py .                                                         [100%]

============================== 1 passed in 0.01s ===============================
```

Como ficam as falhas?

```
========================================= test session starts ==========================================
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 1 item

test_pytest.py F                                                                              [100%]

=============================================== FAILURES ===============================================
_____________________________________ test_meu_primeiro_teste _____________________________________

    def test_meu_primeiro_teste():
>       assert False
E       assert False

test_pytest.py:2: AssertionError
======================================= short test summary info ========================================
FAILED test_pytest.py::test_meu_primeiro_teste - assert False
========================================== 1 failed in 0.02s ===========================================
```

Como ficam as falhas?

```
========================================= test session starts ==========================================
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 1 item

test_pytest.py F                                                                              [100%]

============================================== FAILURES ===============================================
_____________________________________ test_meu_primeiro_teste ______________________________________

    def test_meu_primeiro_teste():
>       assert False
E       assert False

test_pytest.py:2: AssertionError
======================================= short test summary info ========================================
FAILED test_pytest.py::test_meu_primeiro_teste - assert False
========================================== 1 failed in 0.02s ===========================================
```

Resumo

Como ficam as falhas?

Falhas

```
========================================= test session starts ==========
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.
rootdir: /home/dunossauro/git/code
collected 1 item

test_pytest.py F                                                  [100%]

================================================ FAILURES =================================================
_____________________________________ test_meu_primeiro_teste _____________________________________

    def test_meu_primeiro_teste():
>       assert False
E       assert False

test_pytest.py:2: AssertionError
======================================= short test summary info ========================================
FAILED test_pytest.py::test_meu_primeiro_teste - assert False
========================================== 1 failed in 0.02s ===========================================
```

Como ficam as falhas?

```
========================================= test session starts ==========================================
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 1 item

test_pytest.py F                                                                              [100%]

=============================================== FAILURES ===============================================
_____________________________________ test_meu_primeiro_teste ______________________________________

    def test_meu_primeiro_teste():
>       assert False
E       assert False

test_pytest.py:2: AssertionError
======================================= short test summary info ========================================
FAILED test_pytest.py::test_meu_primeiro_teste - assert False
========================================== 1 failed in 0.02s ===========================================
```

Resumo com nome dos testes que falham

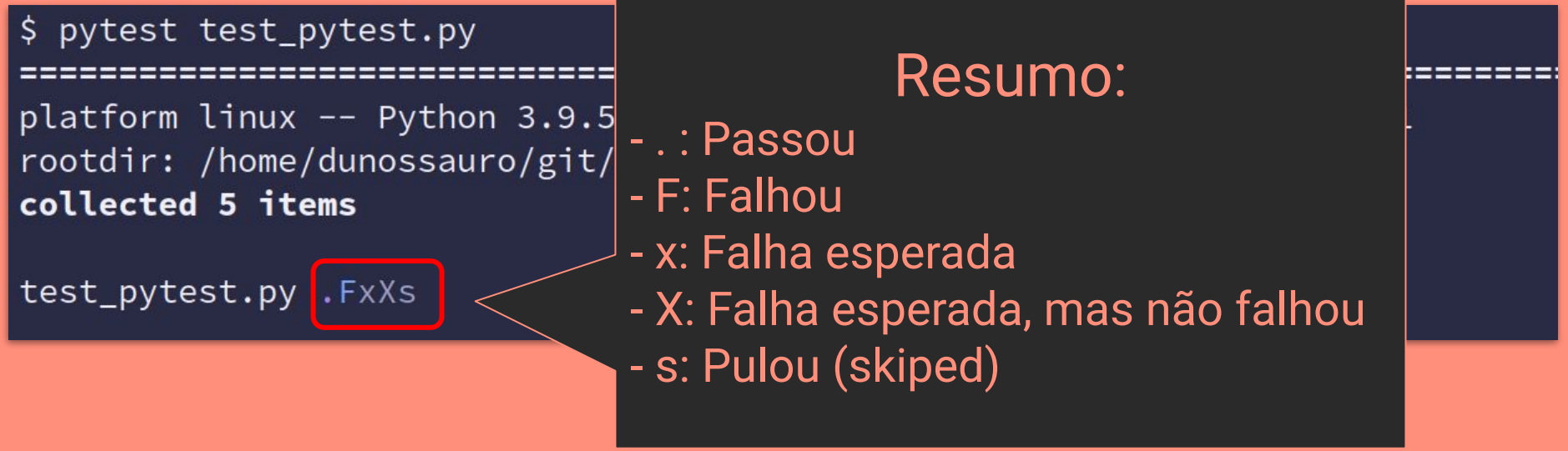
Olhando para as respostas do resumo

```
$ pytest test_pytest.py
========================================= test session starts ==========
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 5 items

test_pytest.py .FxXs
```

Olhando para as respostas do resumo

```
$ pytest test_pytest.py
========================================= test session starts ==========
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/dunossauro/git/code
collected 5 items

test_pytest.py .FxXs
```

Olhando para as respostas do resumo



Como ver todos os testes?

Para isso podemos usar o modo verboso

```
$ pytest test_pytest.py -v
========================================= test session starts ==========================================
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1 -- /home/dunossauro/.cache/pypo
etry/virtualenvs/code-Cbk5BWGZ-py3.9/bin/python
cachedir: .pytest_cache
rootdir: /home/dunossauro/git/code
collected 5 items

test_pytest.py::test_meu_primeiro_teste PASSED                                                  [ 20%]
test_pytest.py::test_meu_segundo_teste FAILED                                                   [ 40%]
test_pytest.py::test_meu_terceiro_teste XFAIL                                                   [ 60%]
test_pytest.py::test_meu_quarto_teste XPASS                                                     [ 80%]
test_pytest.py::test_meu_quinto_teste SKIPPED (unconditional skip)                              [100%]
```

Como ver todos os testes?

Para isso podemos usar o modo verboso

```
$ pytest test_pytest.py -v
============================= test session starts ==============================
platform linux -- Python 3.9.5, py                    y-0.13.1 -- /home/dunossauro/.cache/pypo
etry/virtualenvs/code-Cbk5BWGZ-py3
cachedir: .pytest_cache
rootdir: /home/dunossauro/git/code
collected 5 items

test_pytest.py::test_meu_primeiro_teste PASSED                           [ 20%]
test_pytest.py::test_meu_segundo_teste FAILED                            [ 40%]
test_pytest.py::test_meu_terceiro_teste XFAIL                            [ 60%]
test_pytest.py::test_meu_quarto_teste XPASS                              [ 80%]
test_pytest.py::test_meu_quinto_teste SKIPPED (unconditional skip)       [100%]
```

pytest -v

Como ver todos os testes?

Para isso podemos usar o modo verboso

Nome do arquivo

```
$ pytest test_pytest.py -v
============================= starts ==========================================
platform linux -- Python 3                    0, pluggy-0.13.1 -- /home/dunossauro/.cache/pypo
etry/virtualenvs/code-Cb
cachedir: .pytest_cach
rootdir: /home/dunos      uro/git/code
collected 5 items

test_pytest.py::test_meu_primeiro_teste PASSED                                [ 20%]
test_pytest.py::test_meu_segundo_teste FAILED                                 [ 40%]
test_pytest.py::test_meu_terceiro_teste XFAIL                                 [ 60%]
test_pytest.py::test_meu_quarto_teste XPASS                                   [ 80%]
test_pytest.py::test_meu_quinto_teste SKIPPED (unconditional skip)            [100%]
```

Como ver todos os testes?

Para isso podemos usar o modo verboso

```
$ pytest test_pytest.py -v
=====================================                                    =====================================
platform linux -- Python 3.9.5, pyt                                    -0.13.1 -- /home/dunossauro/.cache/pypo
etry/virtualenvs/code-Cbk5BWGZ-py3
cachedir: .pytest_cache
rootdir: /home/dunossauro/git/c    e
collected 5 items

test_pytest.py::test_meu_primeiro_teste PASSED                                    [ 20%]
test_pytest.py::test_meu_segundo_teste FAILED                                     [ 40%]
test_pytest.py::test_meu_terceiro_teste XFAIL                                     [ 60%]
test_pytest.py::test_meu_quarto_teste XPASS                                       [ 80%]
test_pytest.py::test_meu_quinto_teste SKIPPED (unconditional skip)                [100%]
```

Nome do teste

Como ver todos os testes?

Para isso podemos usar o modo verboso

```
$ pytest test_pytest.py -v
=====================================                                    =====================================
platform linux -- Python 3.9.5, pyt                                    -0.13.1 -- /home/dunossauro/.cache/pypo
etry/virtualenvs/code-Cbk5BWGZ-py3.
cachedir: .pytest_cache
rootdir: /home/dunossauro/git/code
collected 5 items

test_pytest.py::test_meu_primeiro_teste PASSED                                                    [ 20%]
test_pytest.py::test_meu_segundo_teste FAILED                                                     [ 40%]
test_pytest.py::test_meu_terceiro_teste XFAIL                                                     [ 60%]
test_pytest.py::test_meu_quarto_teste XPASS                                                       [ 80%]
test_pytest.py::test_meu_quinto_teste SKIPPED (unconditional skip)                                [100%]
```

Resultado do teste

Resultado dos testes?

O pytest já conta com uma ferramenta de report incluída para o formato Junit (padrão dos frameworks de teste)

```
pytest --junitxml report.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<testsuites>
  <testsuite name="pytest" errors="0" failures="1" skipped="2" tests="5" time="0.033" timestamp="2021-06-07T14:56:35.412219" hostname="babbage">
    <testcase classname="test_pytest" name="test_meu_primeiro_teste" time="0.001" />
    <testcase classname="test_pytest" name="test_meu_segundo_teste" time="0.001">
      <failure message="assert False">def test_meu_segundo_teste():
&gt;       assert False
E       assert False

test_pytest.py:9: AssertionError
      </failure>
    </testcase>
    <testcase classname="test_pytest" name="test_meu_terceiro_teste" time="0.000">
      <skipped type="pytest.xfail" message="" />
    </testcase>
    <testcase classname="test_pytest" name="test_meu_quarto_teste" time="0.000" />
    <testcase classname="test_pytest" name="test_meu_quinto_teste" time="0.000">
      <skipped type="pytest.skip" message="unconditional skip">/home/dunossauro/git/code/test_pytest.py:22: unconditional skip
      </skipped>
    </testcase>
  </testsuite>
</testsuites>
```

Linha de comando

- -v: Mostra o nome dos testes executados
- -s: Mostra as saídas no console
- -k "nome_dos_testes": Filtra resultados
- -x: Saida rápida

- --pdb: Para debugar quando falhar

Marcações, argumentos e metadados

Mark

Mark

A funcionalidade de marcação pode nos ajudar a montar "tags" ou "grupos" para testes específicos. Podemos simplificar chamadas ou rodar testes específicos para casos específicos.

```python
from pytest import mark

@mark.tag
def test_meu_quinto_teste():
    assert True
```

Mark

O marcador pode ter qualquer nome

A funcionali
pode nos aju
ou "grupos"
específicos. Podemos simplificar chamadas ou rodar testes específicos para casos específicos.

```
1  from pytest import mark
2
3  @mark.tag
4  def test_meu_quinto_teste():
5      assert True
```

Filtro por tag

As tags pode ser filtradas pelo argumento **-m marcador** na linha de comando. Você também pode fazer a marcação invertida **-m "not marcador"**.

Assim você pode ter controle de quais testes vão ser executados de maneira simples.

```
$ pytest test_pytest.py -m xpto -v
============================================= test session starts ======
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
ualenvs/code-Cbk5BWGZ-py3.9/bin/python
cachedir: .pytest_cache
rootdir: /home/dunossauro/git/code
collected 5 items / 4 deselected / 1 selected

test_pytest.py::test_meu_quinto_teste PASSED

=================================== 1 passed, 4 deselected in 0.00s
```

Tags embutidas

O pytest fornece um grupo de tags que facilitam o nosso dia a dia em coisas que são comuns em várias suites de teste.

- @mark.skip: Para pular um teste
- @mark.skipif: Para pular um teste em determinado contexto
- @mark.xfail: É esperado que esse teste falhe em algum contexto
- @mark.usefixture: Falaremos depois sobre isso
- @mark.parametrize: Para parametrizar testes (próximo slide)

mark.parametrize

Imagine que você gostaria de fazer uma gama de testes somente alterando os valores e checando seus resultados? O parametrize cria um esquema de "sub testes" onde cada parâmetro será executa uma única vez, mas o teste será executado múltiplas vezes.

```
1  @mark.parametrize(
2    'parametro,resultado_esperado',
3    [(1, 3), (3, 5), (5, 7)]
4  )
5  def test_soma_mais_2(parametro, resultado_esperado):
6    assert soma_mais_dois(parametro) == resultado_esperado
```

Exemplo do resultado

```
$ pytest test_pytest.py -v
============================================= test session starts
platform linux -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.
ualenvs/code-Cbk5BWGZ-py3.9/bin/python
cachedir: .pytest_cache
rootdir: /home/dunossauro/git/code
collected 3 items

test_pytest.py::test_soma_mais_2[1-3] PASSED
test_pytest.py::test_soma_mais_2[3-5] PASSED
test_pytest.py::test_soma_mais_2[5-7] PASSED

============================================== 3 passed in 0.01s =
```

xfail e skipif

xfail e **skipif** são metadados da função que podem dizer quando algum teste deve falhar, pois é esperado, e quando um teste não deve ser executado.

```python
@mark.skipif(
  sys.platform == "win32",
  reason="Não funciona no windows",
)
def test_soma_2_linux():
  numero_do_pinguim = 42
  assert soma_mais_dois(numero_do_pinguim) == 42

```

Linha de comando

- -v: Mostra o nome dos testes executados
- -s: Mostra as saídas no console
- -k "nome_dos_testes": Filtra resultados
- -x: Saida rápida

- --pdb: Para debugar quando falhar

- -m "meu_marcador": Marcadores
- -rs: Mostra o motivo do teste ter skipado

Fixtures

Uma introdução muito, MAS MUITO, sutil.

Afinal o que são fixtures?

A fixture é basicamente uma maneira de "entrar" em um contexto. Ou prover uma ferramenta que precisa ser executada "antes" dos testes.

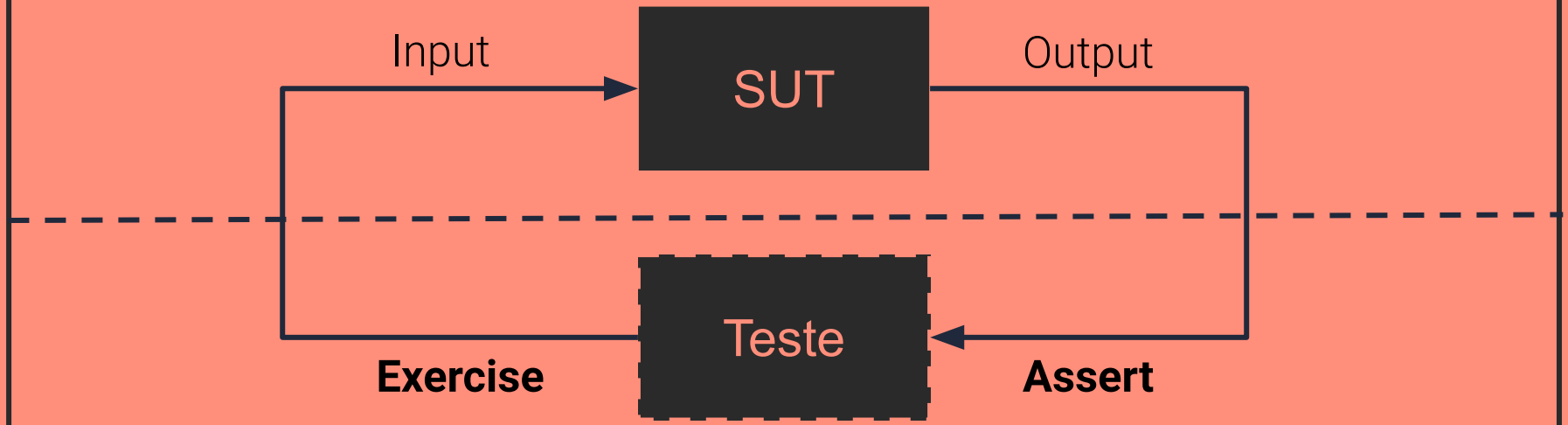
Afinal o que são fixtures?

Precisamos entender uma coisa. Todos os testes, mesmo os de uma linha são formados por 4 fases (em outras literaturas 3):

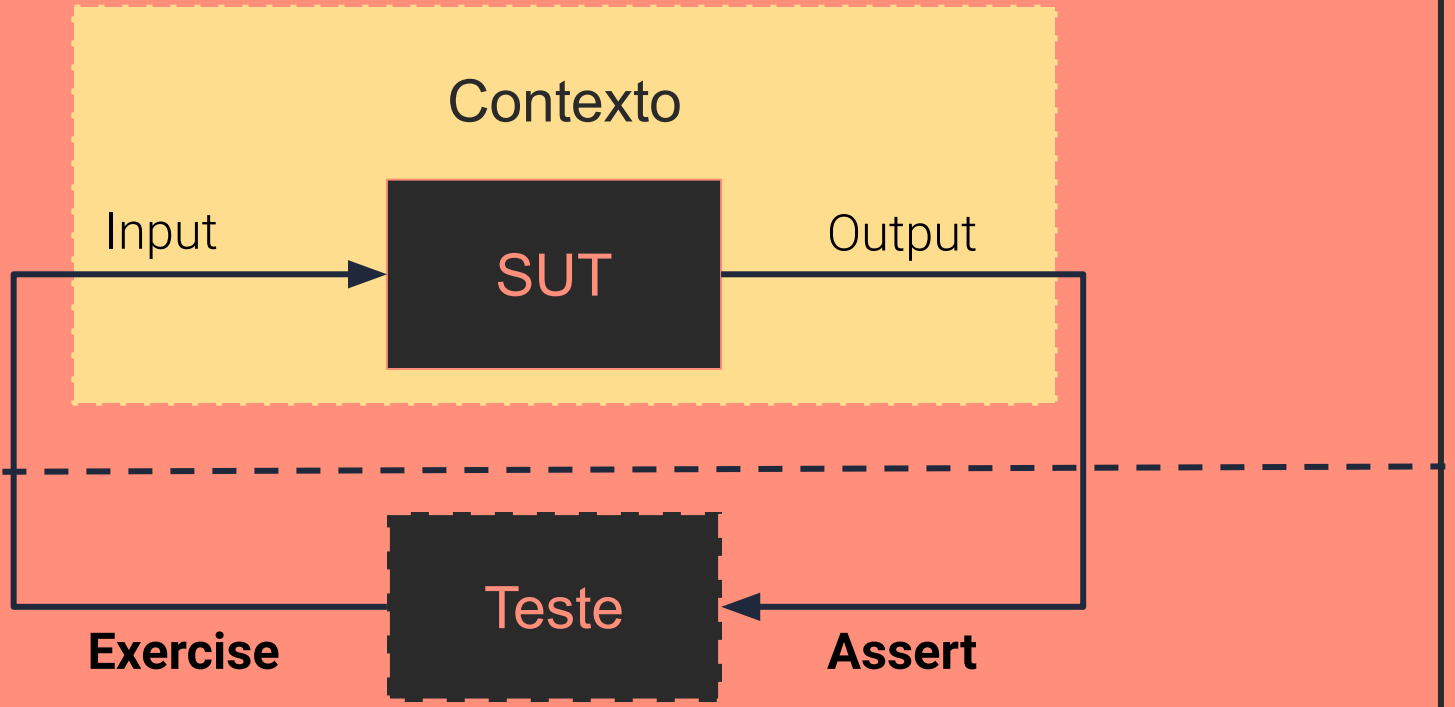
- Setup: Onde montamos as coisas
- Exercise: Onde chamamos as coisas
- Assert: Onde verificamos as coisas
- TearDown: Onde desmontamos as coisas

Afinal o que são fixtures?

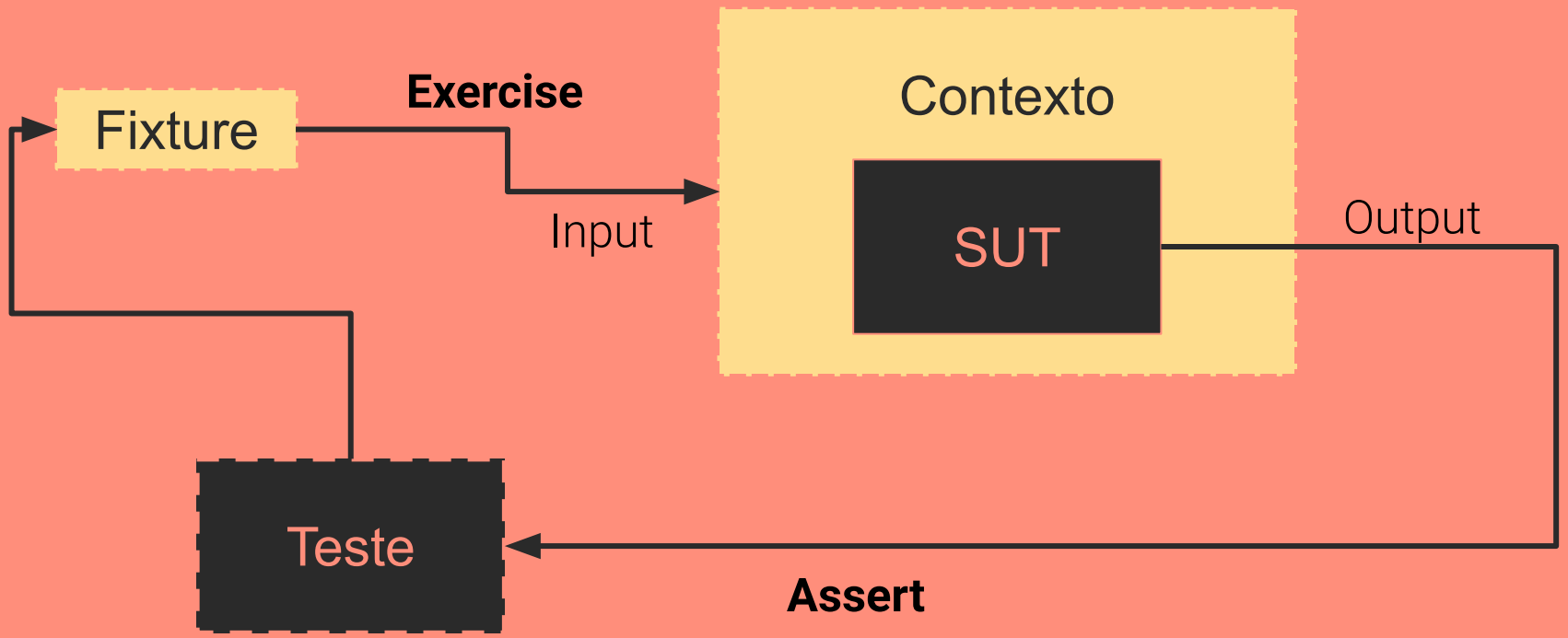
Todos os testes que fizemos até agora, exercitam duas partes disso



Afinal o que são fixtures?



Afinal o que são fixtures?



Um exemplo básico

Vamos supor que o que define o sucesso de execução de um teste é um **print**. É bem menos trivial que parece. Pois precisamos ficar "ouvindo" a saída padrão do sistema (sys.stdout).

Como faríamos isso?

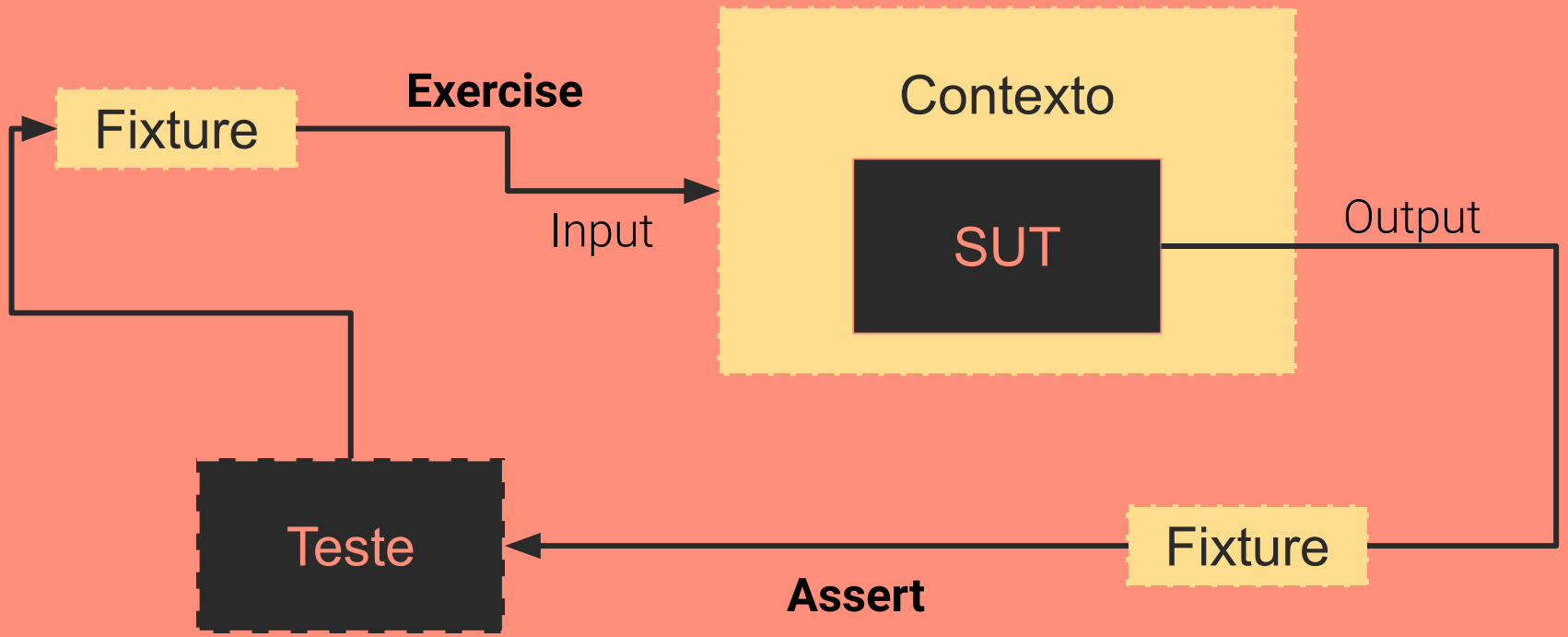
Um exemplo básico

Vamos supor que o que define o sucesso de execução de um teste é um **print**. É bem menos trivial que parece. Pois precisamos ficar "ouvindo" a saída padrão do sistema (sys.stdout).

Como faríamos isso?

Teríamos que começar a "espionar" o stdout antes do teste e durante o teste checar se o valor foi efetivamente "printado"

Afinal o que são fixtures?



Quais fixtures temos disponíveis?

- capsys e variações: "espiona" o stdout
- tempdir: Cria um diretório temporário
- caplog: "espiona" logs
- mokeypatch: Adiciona atributos e métodos a objetos em runtime
- ...

```
1  def test_output(capsys):
2    print('meu print bolado')
3    captured = capsys.readouterr()
4    assert captured.out == "meu print bolado\n"
```

Posso criar minhas próprias fixtures?

Mas e agora?

SIIM!

```python
from pytest import fixture
from app import create_app

@fixture
def flask_app():
  return create_app()

def test_com_app(flask_app):
  ...
```

Fixtures são o assunto pra próxima live, é o assunto mais extenso do pytest!

MAAAASSSS???

picpay.me/dunossauro

apoia.se/livedepython

PIX

Ajude o projeto