Introdução aos testes!

Live de Python #232

Roteiro

0. Como um programa funciona?

Uma visão para testes

2. Afinal, o que são testes?

Um ponto de partida

3. Isolando coisas

Para não ter que executar o código todo

4. O framework

Simplificando as validações

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Andre Azevedo, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bernardo At, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Christiano Morais, Claudemir Firmino, Damianth, Daniel Freitas, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Goivinho, Delton Porfiro, Denis Bernardo, Dgeison Peixoto, Diego Farias, Diego Guimarães, Dilenon Delfino, Diogo Paschoal, Edgar, Edinilson Bonato, Eduardo Tolmasquim, Elias Silva, Emerson Rafael, Eneas Teles, Érico Andrei, Everton Silva, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Firehouse, Flávio Meira, Francisco Neto, Gabriel Barbosa, Gabriel Espindola, Gabriel Mizuno, Gabriel Nascimento, Gabriel Simonetto, Geandreson Costa, Gênese Lessa, Gilberto Abrao, Giovanna Teodoro, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Guilherme Silva, Guionardo Furlan, Gustavo Pereira, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Hugo Cosme, Ismael Ventura, Italo Silva, Janael Pinheiro, Jhonatan Martins, Joelson Sartori, Jônatas Silva, Jon Cardoso, Jorge Silva, Jose Alves, José Gomes, Joseíto Júnior, Jose Mazolini, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leandro Silva, Leonardo Mello, Leonardo Nazareth, Leon Solon, Luancomputacao Roger, Lucas Adorno, Lucas Carderelli, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Lucas Valino, Luciano Filho, Luciano Ratamero, Luciano Silva, Luciano Teixeira, Luis Alves, Luiz Duarte, Luiz Lima, Luiz Paula, Luiz Perciliano, Mackilem Laan, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Maria Clara, Marina Passos, Mateus Lisboa, Matheus Silva, Matheus Vian, Mauricio Fagundes, Mauricio Nunes, Mlevi Lsantos, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Paulo D., Paulo Tadei, Pedro Gomes, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Pydocs Pro, Pytonyc, Rafael Lino, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Regis Santos, Renato Oliveira, Rene Bastos, Ricardo Silva, Riverfount, Rjribeiro, Robson, Robson Maciel, Rodrigo Freire, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Ribeiro, Rodrigo Vaccari, Rodrigo Vieira, Rogério Lima, Rogério Nogueira, Ronaldo Silva, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Thaynara Pinto, Thiago Araujo, Thiago Borges, Thiago Curvelo, Tiago Minuzzi, Tony Dias, Tyrone Damasceno, Uadson Emile, Valcilon Silva, Valdir Tegon, Vcwild, Vinicius Stein, Vladimir Lemos, Walter Reis, Willian Lopes, Wilson Duarte, Wilson Neto, Wilson Rocha, Xico Silvério, Yuri Fialho, Zeca Figueiredo

Obrigado você

O objetivo hoje é começar do absoluto **ZERO**

Aviso

O objetivo hoje é começar do absoluto **ZERO**

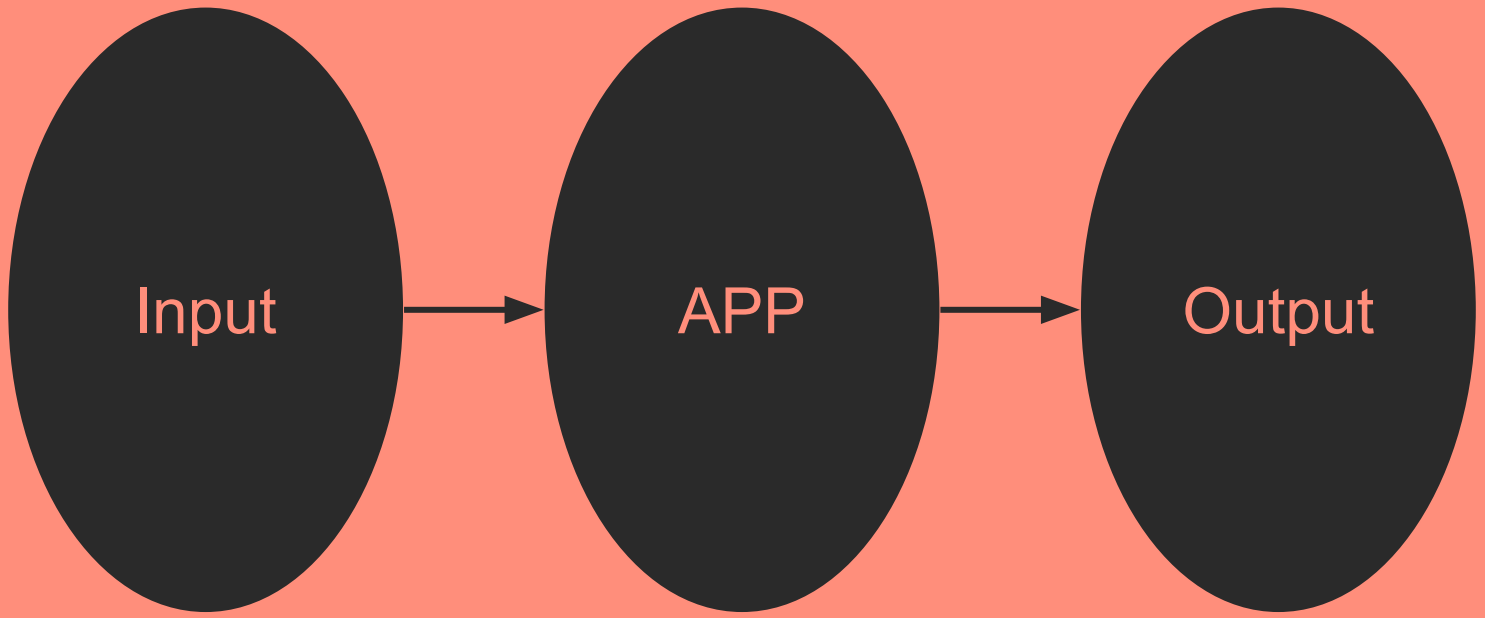
Todas as perguntas estão liberadas, me interrompa!

Aviso

Como um programa funciona?

?

Uma aplicação hipotética



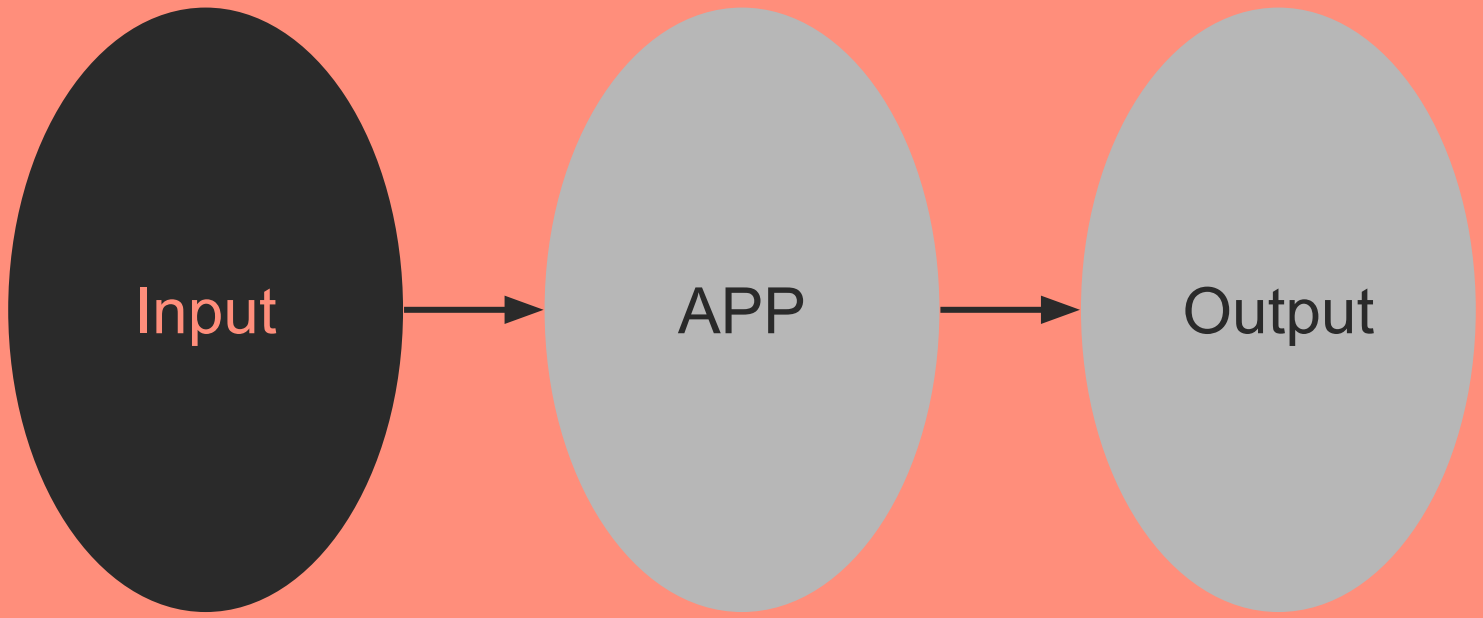
Dando um viés prático

“Faça um Programa que peça a temperatura em graus Fahrenheit, transforme e mostre a temperatura em graus Celsius"

C = 5 * ((F - 32) / 9).

Problema 9: https://wiki.python.org.br/EstruturaSequencial

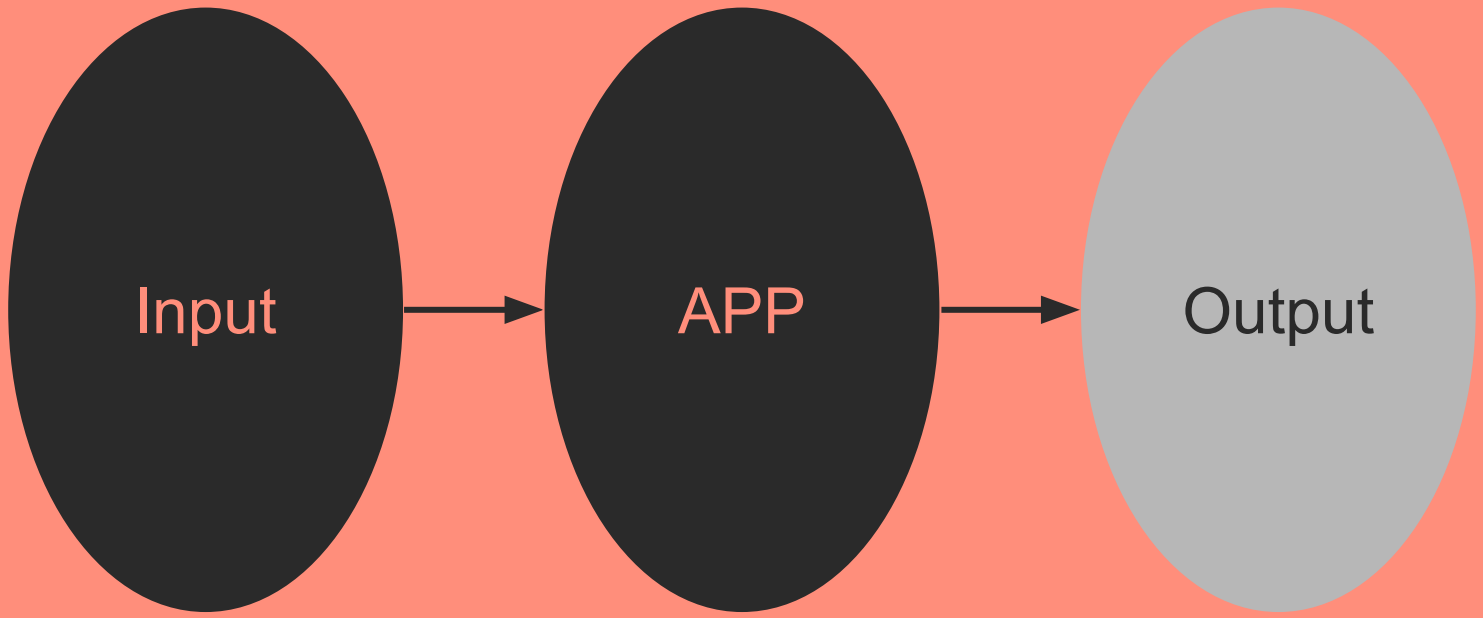
Botando em prática



A entrada de dados

```
1   temperatura = input('Digite a temperatura em Fahrenheit: ')
```

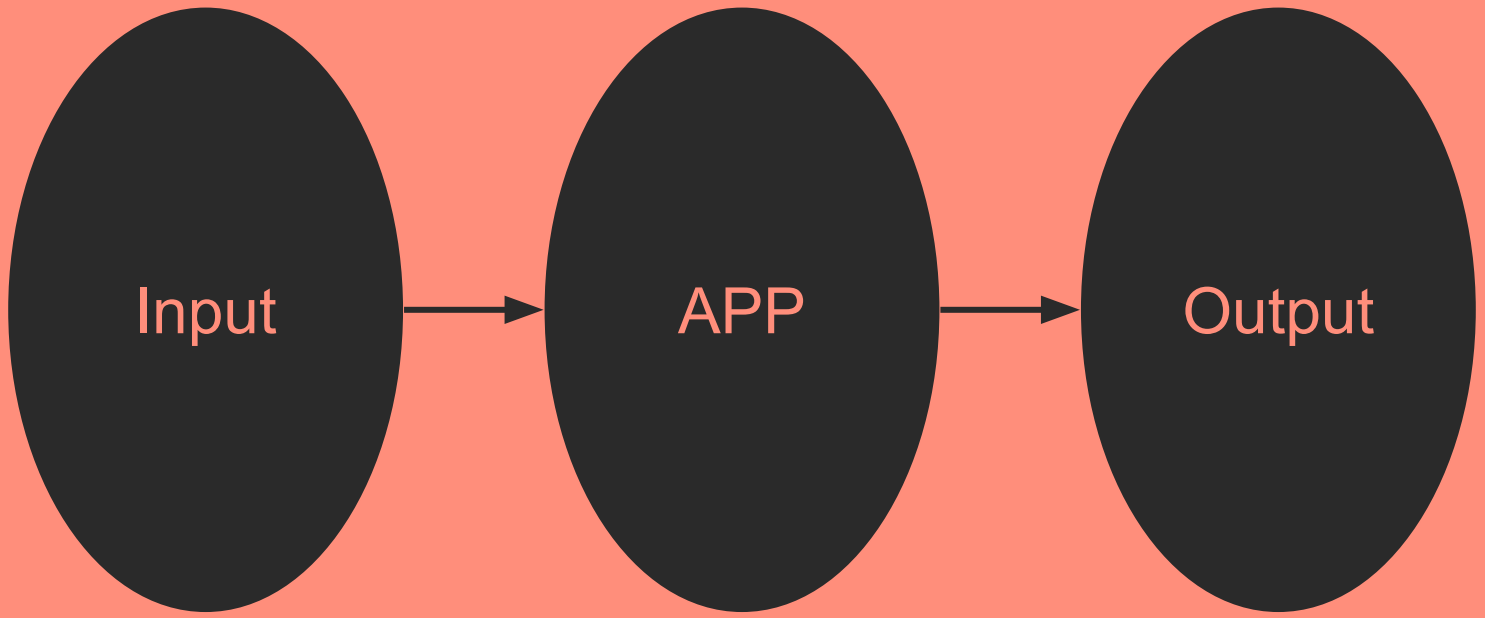
Botando em prática



Entrada e processamento

```python
# input
temperatura = float(input('Digite a temperatuda em Fahrenheit: '))

# Processamento
celsius = 5 * ((temperatura - 32) / 9)
```

Todo o caminho sendo percorrido



Entrada / Processamento / Saída

```python
# input
temperatura = float(input('Digite a temperatura em Fahrenheit: '))

# Processamento
celsius = 5 * ((temperatura - 32) / 9)

# Output
print(f'A temperatura em Celsius é: {celsius}')
```

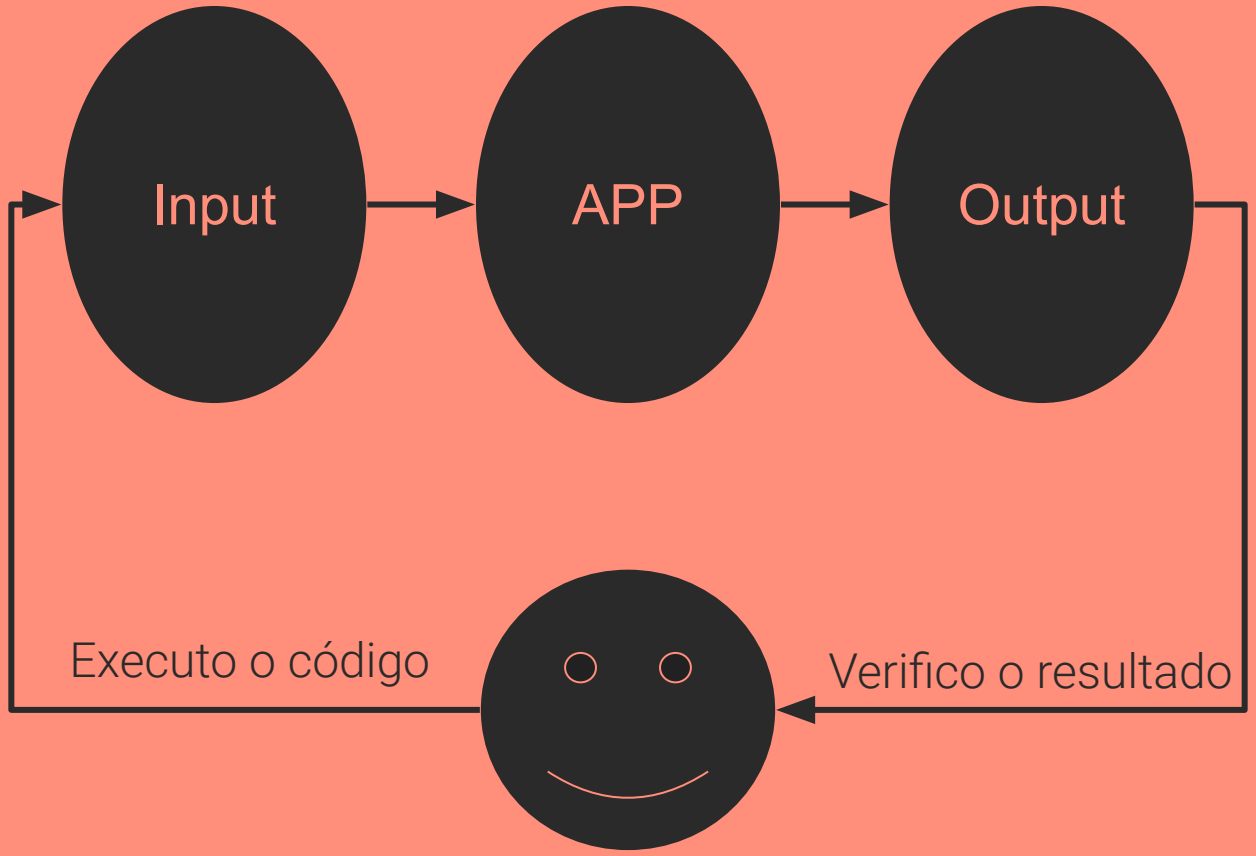
Testes

Um ponto de partida!

Afinal, o que são testes?

- Um ciclo de feedback
- Uma garantia de que funciona

A dinâmica dos testes



Geralmente, testamos assim

```
1   # Shell ou Play do editor
2   python exemplo_01.py
3
4   Digite a temperatuda em Fahrenheit: 80
5   A temperatura em Celsius é: 26.666666666666664
6
```

Isso nada mais é que um teste

Porém, nós sabemos programar!

Não precisamos fazer isso na mão!

Isolar para testar

Vamos pensar um pouco.

"O que podemos fazer para que esse código consiga ser testado sem a nossa interferência?"

Decomposição

Basicamente, se nosso software tem 3 linhas ou 5.000 linhas. Ele é composto por diversos elementos.

Um software não é nada mais que um monte de blocos de código. Cada bloco executa uma função específica de código.

Decompondo nosso exemplo

```python
# input
temperatura = float(input('Digite a temperatura em Fahrenheit: '))

# Processamento
celsius = 5 * ((temperatura - 32) / 9)

# Output
print(f'A temperatura em Celsius é: {celsius}')
```

Decompondo nosso exemplo

```python
# input
temperatura = float(input('Digite a temperatura em Fahrenheit: '))

# Processamento
celsius = 5 * ((temperatura - 32) / 9)

# Output
print(f'A temperatura em Celsius é: {celsius}')
```

Decompondo nosso exemplo

```python
# input
temperatura = float(input('Digite a temperatura em Fahrenheit: '))

# Processamento
celsius = 5 * ((temperatura - 32) / 9)

# Output
print(f'A temperatura em Celsius é: {celsius}')
```

Como podemos isolar isso?

Devemos separar as preocupações (Dijkstra).

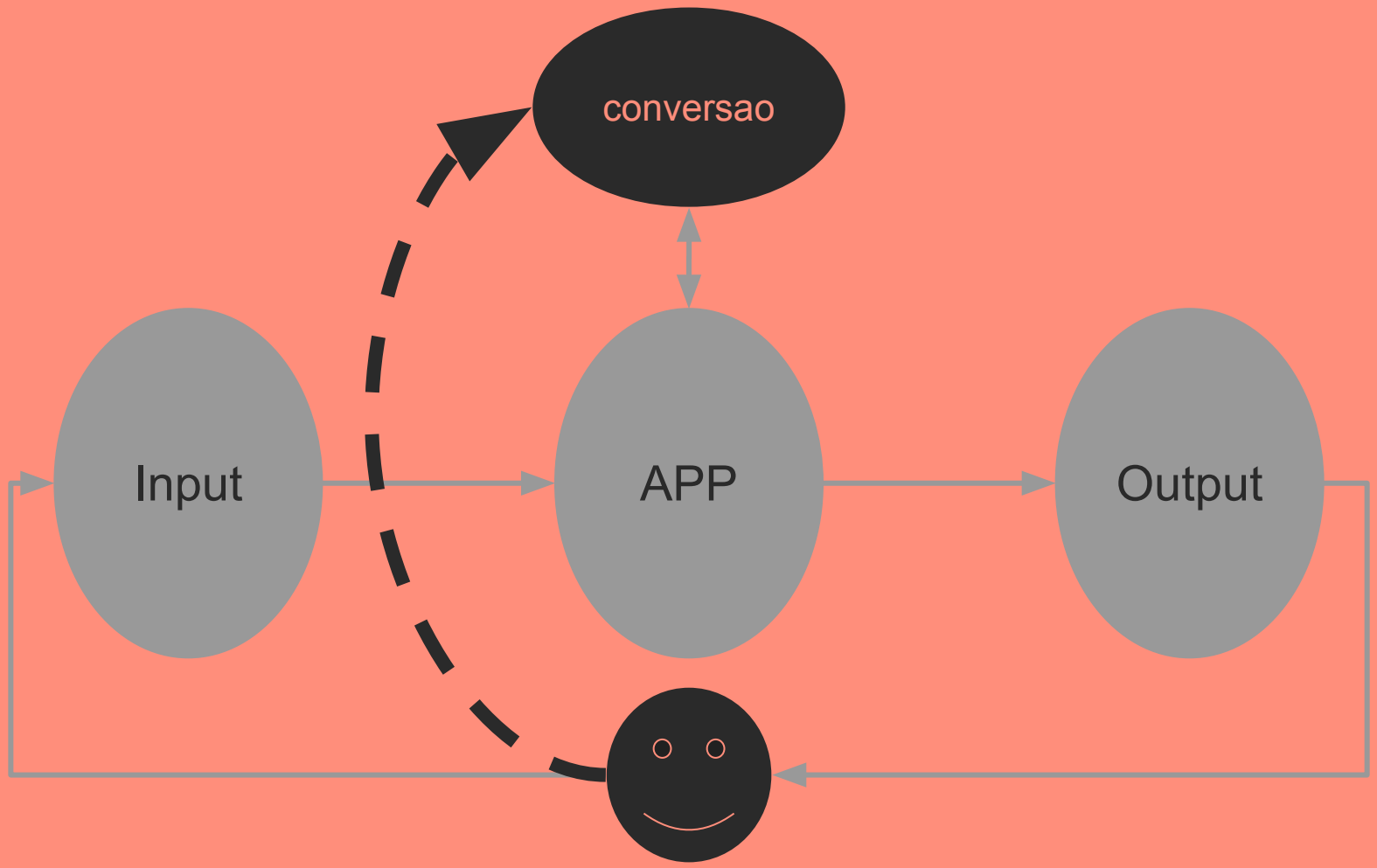
Em pequenos blocos isolados, que podem ser chamados de forma independente.

Uma função pode ser uma forma de isolar isso.

Conversão isolada

```python
def conversao(temperatura):
    return 5 * ((temperatura - 32) / 9)


# input
temperatura = float(input('Digite a temperatura em Fahrenheit: '))

# Processamento
celsius = conversao(temperatura)

# Output
print(f'A temperatura em Celsius é: {celsius}')
```

O que faremos pra saber se funciona?



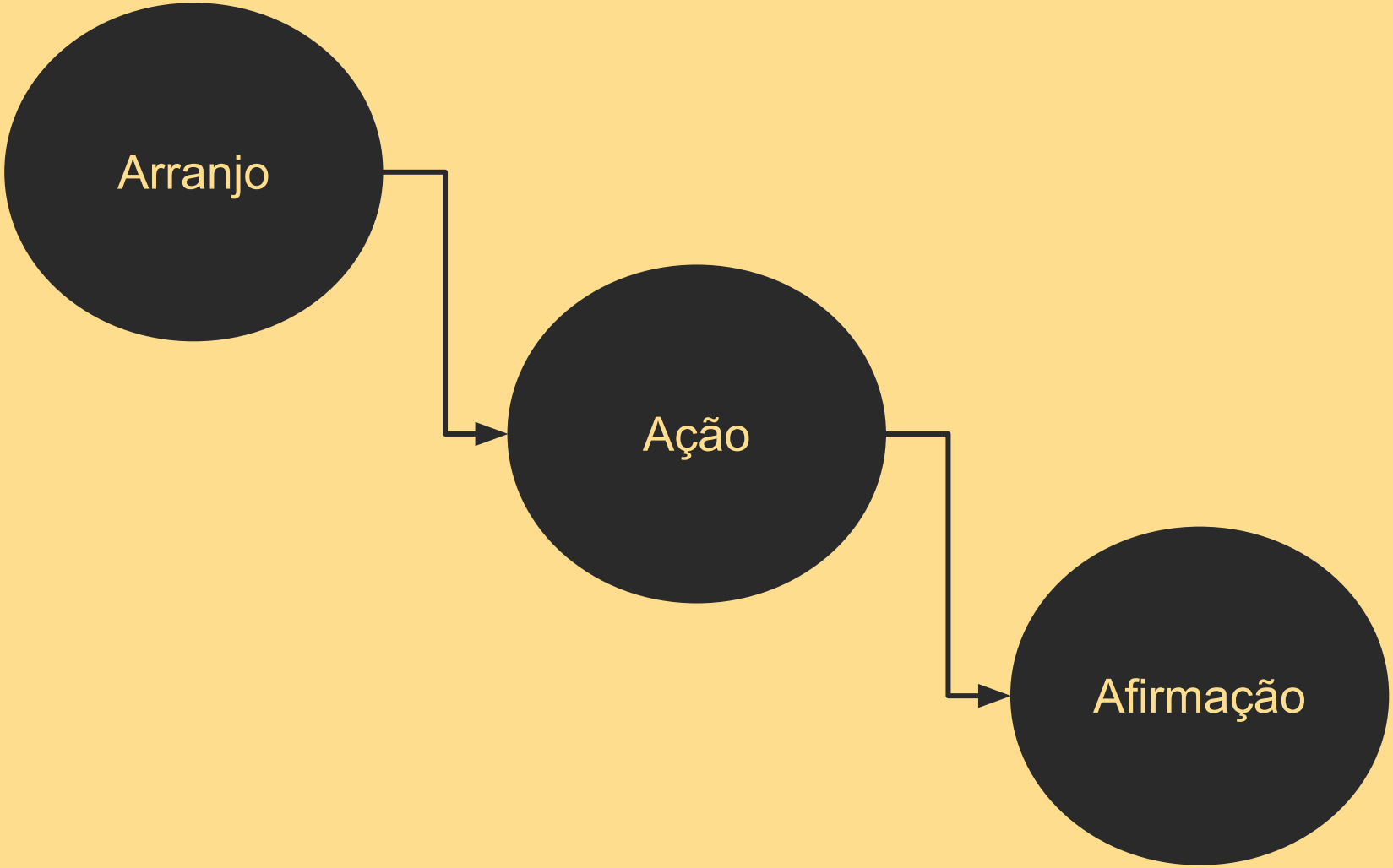
Tornando as coisas testáveis

Isola mento

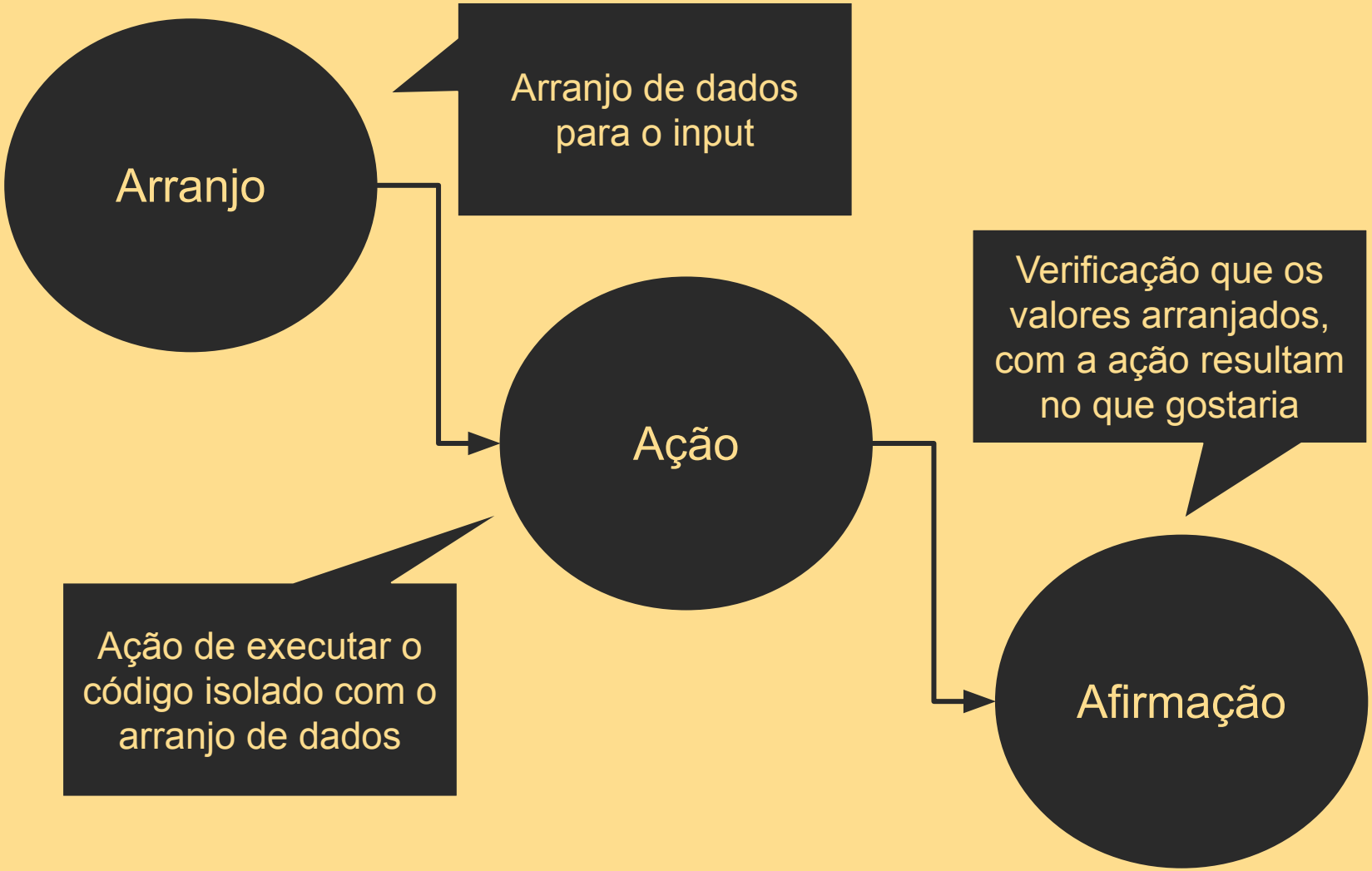
Separando as definições do fluxo de execução

```python
def conversao(temperatura):
    return 5 * ((temperatura - 32) / 9)


if __name__ == '__main__':
    # input
    temperatura = float(input('Digite a temperatura em Fahrenheit: '))

    # Processamento
    celsius = conversao(temperatura)

    # Output
    print(f'A temperatura em Celsius é: {celsius}')
```

Taxonomia de testes



Taxonomia de testes



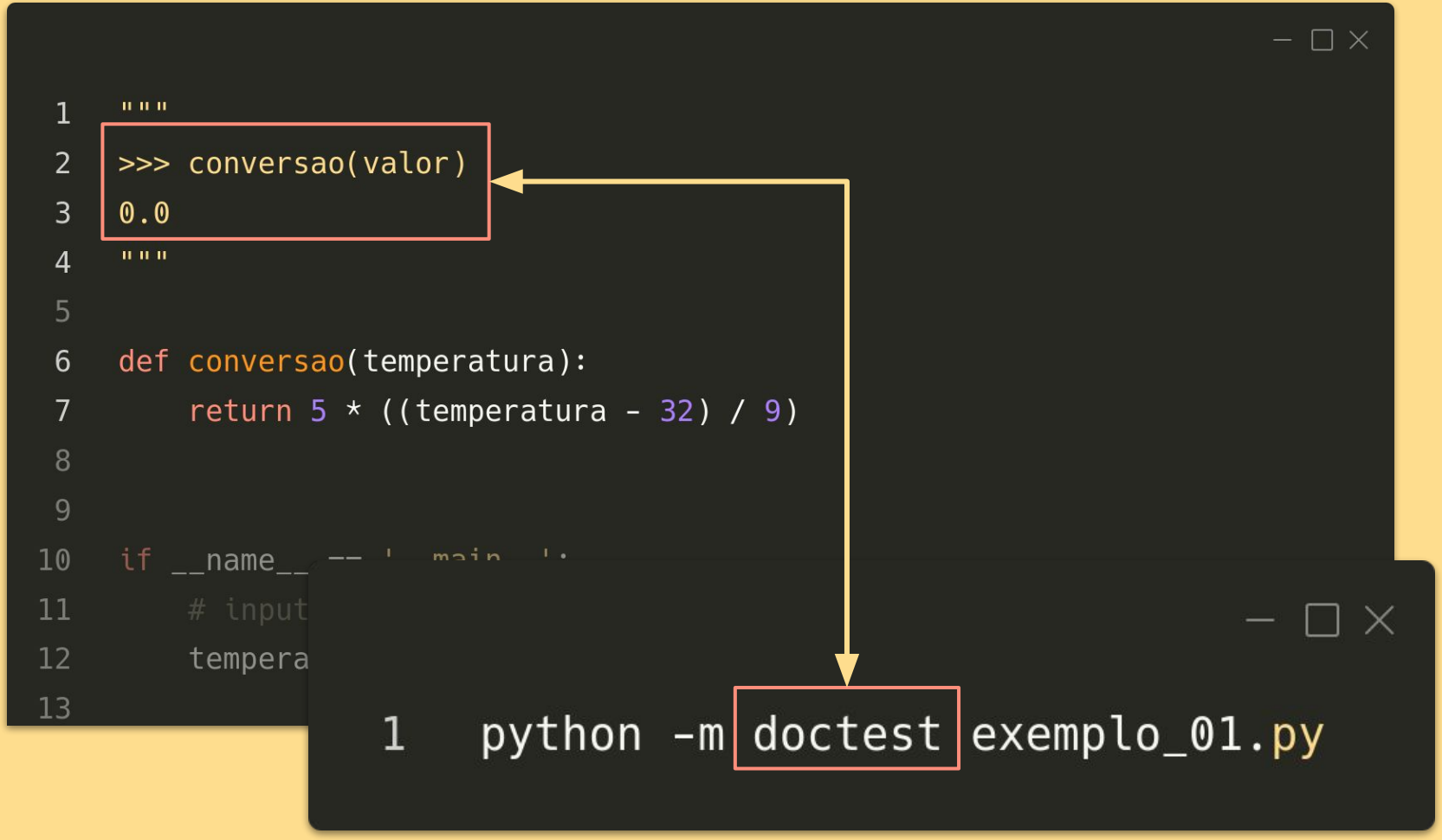
Um roteiro para um teste

Quero afirmar que quando a função de `conversao` receber o valor 32 o resultado será 0

Uma primeira abordagem

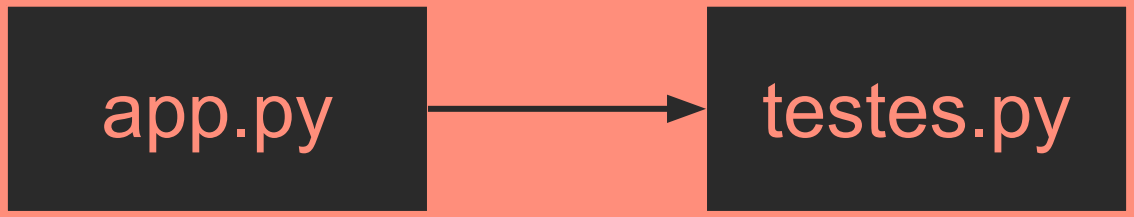
```
"""
>>> conversao(valor)
0.0
"""

def conversao(temperatura):
    return 5 * ((temperatura - 32) / 9)


if __name__ == '__main__':
    # input
    temperatura = float(input('Digite a temperatura em Fahrenheit: '))

```

Uma primeira abordagem

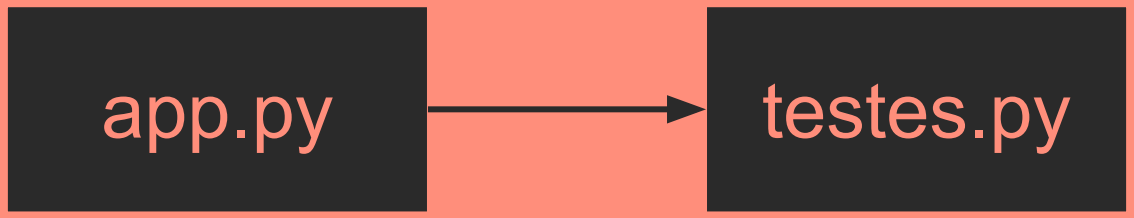


Usando os módulos ao nosso favor

Como isolamos a função da execução. Podemos importar `conversao` em outro módulo. Com isso, podemos iniciar um novo arquivo no projeto. Um arquivo específico para os testes.



Usando os módulos ao nosso favor




```python
# testes.py
from app import conversao

conversao(32) == 0
```

A palavra assert

Assert, é uma palavra do inglês que se destina a "afirmação" de algo.

Algo como:

- Afirme que a conversão de 32 graus fahrenheit é igual a 0 graus celcius;
- Afirme que a conversão de 100 graus fahrenheit é igual a 37.77 graus celcius;

Assert e o seu código

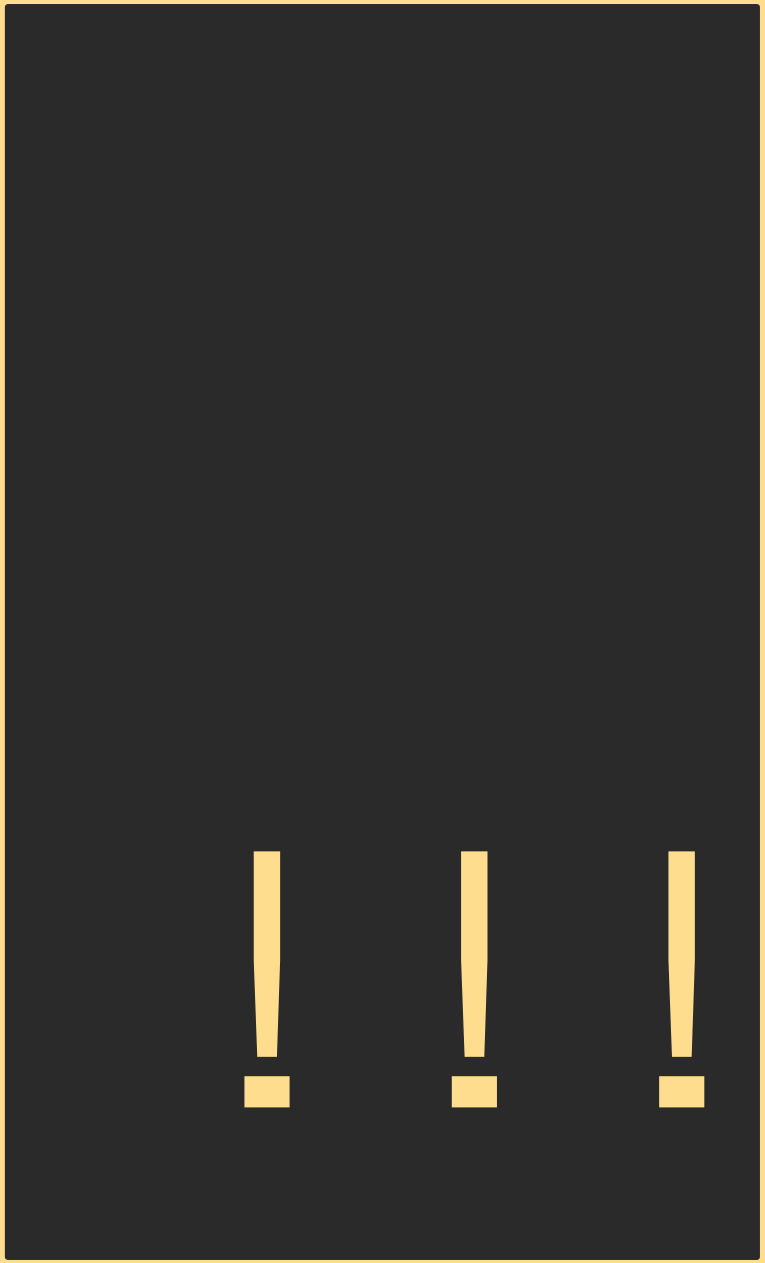
```python
# testes.py
from app import conversao

assert conversao(32) == 0
assert conversao(100) == 37.77
```

Assert e o seu código

```python
# testes.py
from app import conversao

assert conversao(32) == 0
assert conver
```

```
python testes.py
```

Um framework!



O unittest

O python tem uma biblioteca nativa de testes. Que pode nos auxiliar para saber o que funciona e o que não funciona.

```python
from unittest import TestCase
from app import conversao


class TestConversao(TestCase):
    def test_deve_retonar_0_quando_receber_32(self):
        self.assertEqual(conversao(32), 0)

    def test_deve_retornar_37_77_quando_receber_100(self):
        self.assertAlmostEqual(conversao(100), 37.77, places=1)
```

O unittest

O python tem uma biblioteca nativa de testes. Que pode nos auxiliar para saber o que funciona e o que não funciona.

```python
from unittest import TestCase
from app import conversao


class TestConversao(TestCase):
    def test_deve_retonar_0_quando_receber_32(self):
        self.assertEqual(conversao(32), 0)

    def test_deve_retorr
        self.assertAlmos
```

```
pythom -m unittest teste.py
```

Reformulando nosso problema inicial

“Faça um Programa de conversões de temperatura

Pergunte a temperatura que deseja converter se é em Celsius ou Fahrenheit.

Peça que a temperatura em graus seja fornecida e converta"

C = 5 * ((F - 32) / 9) ; F = C * (9 / 5) + 32

```python
def conversao_para_celcius(temperatura):
  return 5 * ((temperatura - 32) / 9)


def conversao_para_fahrenheit(temperatura):
  return temperatura * (9 / 5) + 32


if __name__ == '__main__':
  print('Programa de conversões de temperatura\n\n')
  opcao = int(
      input(
          'Digite 1 para converter de fahrenheit para celsius\n'
          'Digite 2 para converter de celsius para fahrenheit: '
      )
  )

  temperatura = float(input('Digite a temperatura: '))

  match (opcao):
      case 1:
          temperatura_convertida = conversao_para_celcius(temperatura)
          print(f'A temperatura é: {temperatura_convertida}')
      case 2:
          temperatura_convertida = conversao_para_fahrenheit(temperatura)
          print(f'A temperatura é: {temperatura_convertida}')
      case _:
          print('Esclha incorreta!')

```

Descobrindo a cobertura de testes

Às vezes, testamos, mas não sabemos exatamente se testamos o que precisa ser testado. Ou achamos que estamos testando algo que na realidade não estamos testando de fato. Para isso, temos a cobertura de testes.

```
1   pip install coverage
2   python -m coverage run -m unittest teste.py
3   coverage report
4   coverage html
```

Coverage for **app.py**: 24%

17 statements | 4 run | 13 missing | 0 excluded

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-17 21:49 -0300

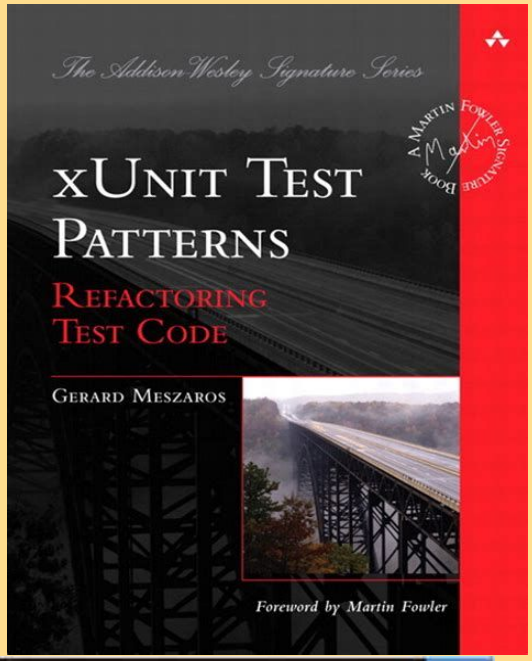
```python
def conversao_para_celsius(temperatura):
    return 5 * ((temperatura - 32) / 9)


def conversao_para_fahrenheit(temperatura):
    return temperatura * (9 / 5) + 32


if __name__ == '__main__':

    print('Programa de conversões de temperatura\n\n')
    opcao = int(
        input(
            'Digite 1 para converter de fahrenheit para celsius\n'
            'Digite 2 para converter de celsius para fahrenheit: '
        )
    )

    temperatura = float(input('Digite a temperatura: '))

    match (opcao):
        case 1:
            temperatura_convertida = conversao_para_celsius(temperatura)
            print(f'A temperatura é: {temperatura_convertida}')
        case 2:
            temperatura_convertida = conversao_para_fahrenheit(temperatura)
            print(f'A temperatura é: {temperatura_convertida}')
        case _:
            print('Esclha incorreta!')
```

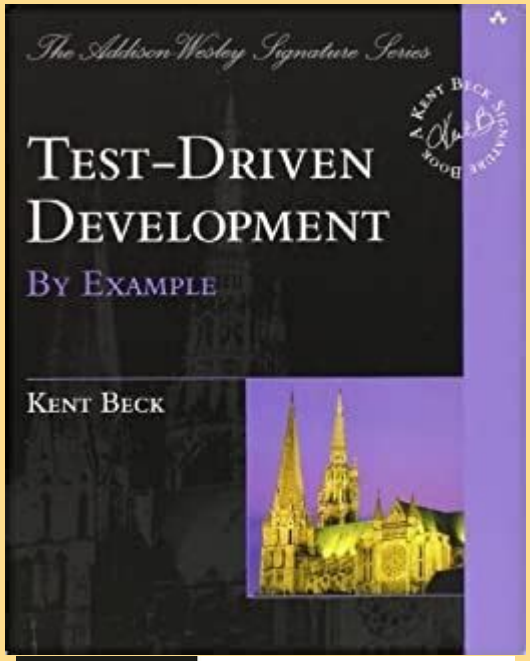
picpay.me/dunossauro

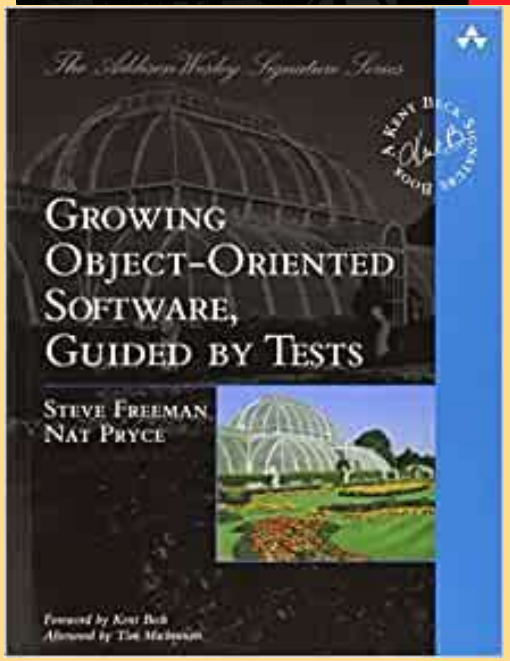
apoia.se/livedepython

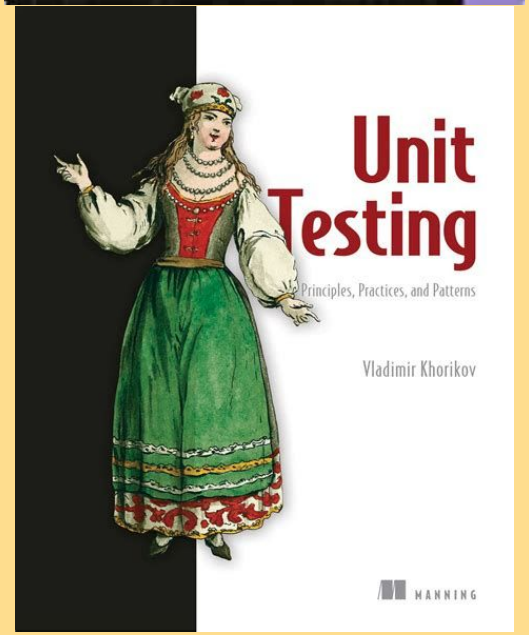
pix.dunossauro@gmail.com

Ajude o projeto <3


The Addison Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler


The Addison Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck


The Addison Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce


Unit Testing
Principles, Practices, and Patterns
Vladimir Khorikov
MANNING