Corrotinas II

Live de Python #153

Roteiro

1. Controle de Fluxo
Entendendo de vez o yield

2. Geradores melhorados
Yield como expressão

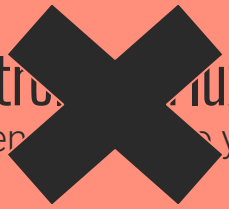
3. Delegando para sub geradores
Cedendo a vez para outro gerador

4. asyncIO
A forma final das corrotinas

Roteiro

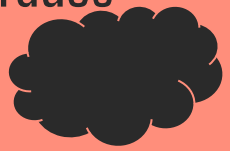
1. Contr[illegible]luxo

Entender[illegible] yield



2. Geradores melhorados

Yield como expressão



3. Delegando para sub geradores

Cedendo a vez para outro gerador

4. asyncIO

A forma final das corrotinas (talvez semana que vem)

picpay.me/dunossauro

apoia.se/livedepython

PIX

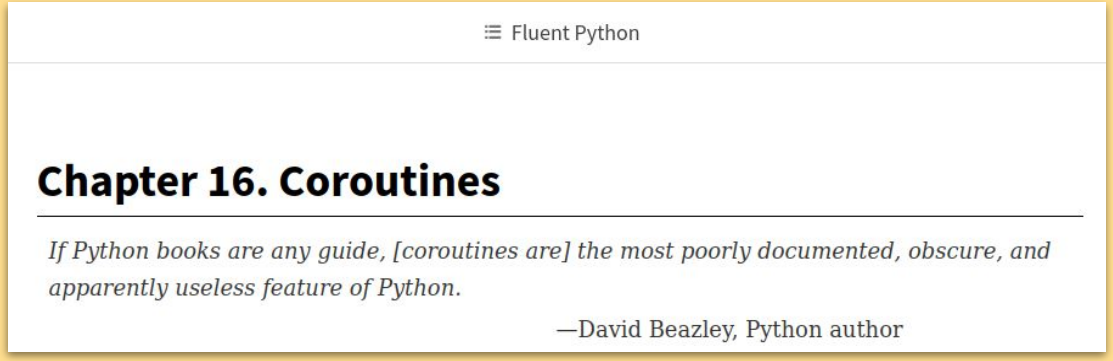
Ajude o projeto

Ademar, Alex Menezes, Alexandre Fernandes, Alexandre Harano, Alexandre Souza, Alexandre Tsuno, Alysson Oliveira, Amaziles Jose, Andre Rodrigues, André Almeida, Antonio Cassiano, Bianca Rosa, Bruno Fernandes, Bruno Rocha, Caio Vinicius, Carlos Alberto, César Moreira, César Túlio, Davi Alves, David Kwast, Diego Moreira, Dilenon Stefan, Douglas Bastos, Edgard Sampaio, Edivaldo Venancio, Edson Braga, Eduardo Marcos, Eduardo Sidney, Elias Da, Eugenio Mazzini, Everton Alves, Fabio Castro, Fabrício Vilela, Faricio Lima, Fernando Lanfranchi, Flavkaze, Franklin Sousa, Fábio Serrão, Gabriel Simonetto, Gabriela Santiago, Geandreson Costa, Gladson Araujo, Guilherme Felitti, Guilherme Marson, Guilherme Ostrock, Henrique Machado, Hélio De, Isaac Ferreira, Israel Azevedo, Italo Bruno, Jeison Sanches, Johnny Tardin, Jonatas Baldin, Jonatas Leon, Jones Ferreira, Jorge Luiz, José Willia, Jovan Costa, João Lugão, João Paulo, Juan Ernesto, Jônatas Silva, Júlia Kastrup, Kaneson Alves, Leonardo Cordeiro, Leonardo Galani, Leonardo Ribeiro, Lorena Carla, Lucas Barreto, Lucas Barros, Lucas Ferreira, Lucas Sartor, Luiz Lima, Maiquel Leonel, Maiquel Leonel, Marcela M, Marcelo Pontes, Maria Clara, Natan Cervinski, Nicolas Teodosio, Otavio Carneiro, Patric Lacouth, Patrick Henrique, Paulo Henrique, Paulo Henrique, Pedro Baesse, Pedro Martins, Peterson W, Rafael De, Reinaldo Chaves, Renan Gomes, Renne Rocha, Rodrigo Ferreira, Rodrigo Vaccari, Ronaldo Fraga, Rubens Gianfaldoni, Sandro Roberto, Silvio Xm, Thiago Dias, Thiago Martins, Tyrone Damasceno, Valdir Junior, Victor Matheus, Vinícius Bastos, Vinícius Borba, Wesley Mendes, Willian Lopes, Willian Lopes, Willian Vieira, Wilson Beirigo

Obrigado você

Disclaimers

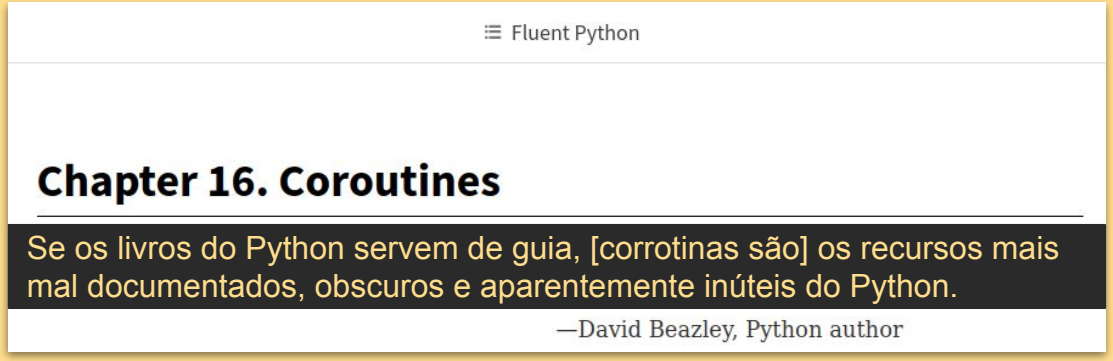
- Vamos no modo lento
- Não temos a ambição de cobrir o tema todo nessa live
- O importante é todas estarem com o básico em mente



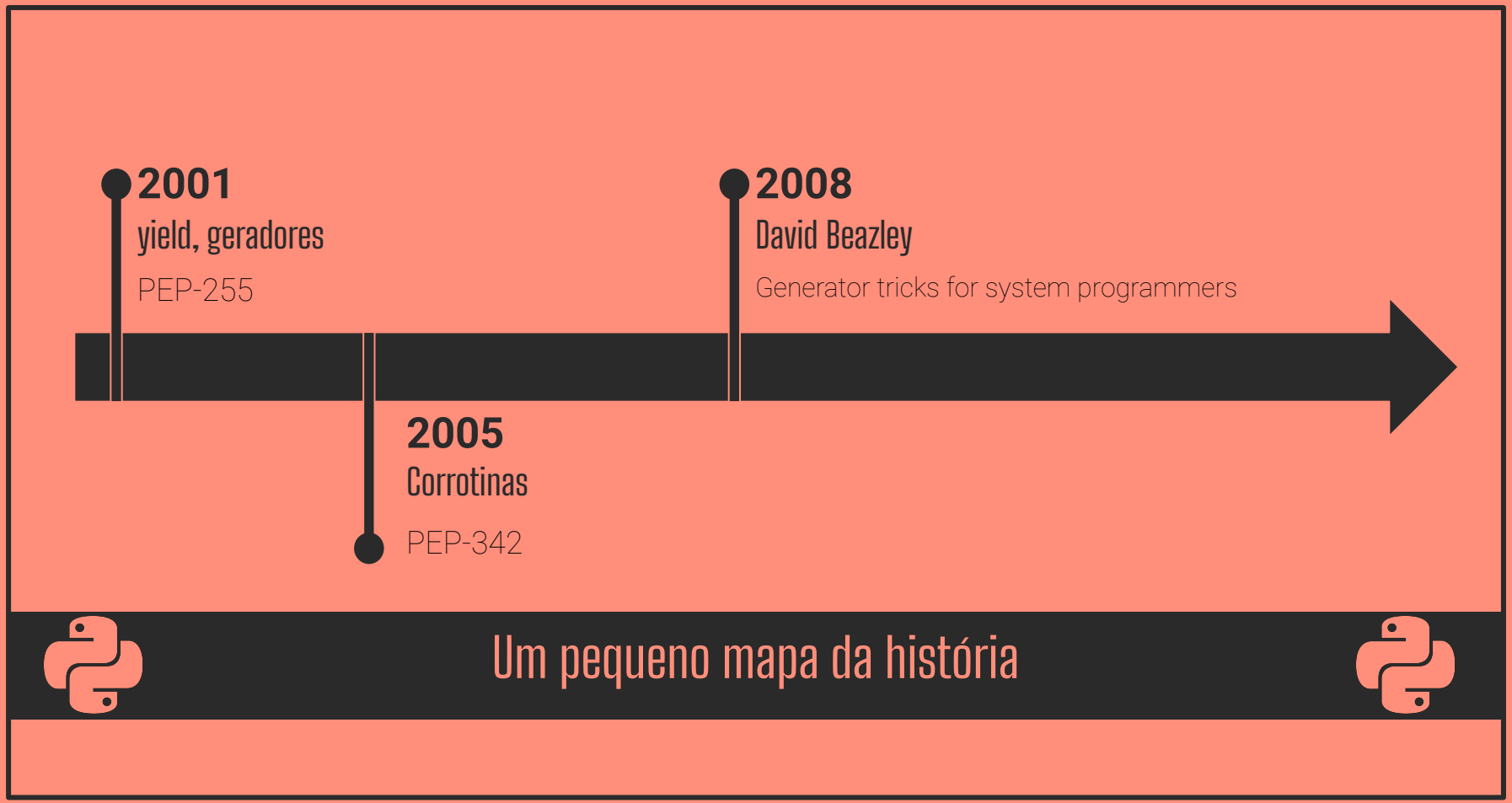
Fluent Python - Luciano Ramalho

Disclaimers

- Vamos no modo lento
- Não temos a ambição de cobrir o tema todo nessa live
- O importante é todas estarem com o básico em mente



Fluent Python - Luciano Ramalho


2001
yield, geradores
PEP-255
2005
Corrotinas
PEP-342
2008
David Beazley
Generator tricks for system programmers
Um pequeno mapa da história

Geradores melhorados

Yield como expressão (2005)

Corrotinas por meio de geradores melhorados

Na PEP-342 foi introduzido o conceito de yield como expressão. Agora yield podem receber valores pelo método `.send()`

```
def corrotina():
    print('Começou')
    valor = yield
    print(f'Recebi: {valor}')
```

Corrotinas por meio de geradores melhorados

Na PEP-342 foi introduzido o conceito de yield como expressão. Agora yield podem receber valores pelo método `.send()`

```
def corrotina():
    print('Começou')
    valor = yield
    print(f'Recebi: {valor}')
```

Corrotinas por meio de geradores melhorados [exemplo_03.py]

Na PEP-342 foi introduzido o conceito de yield como expressão. Agora yield podem receber valores pelo método `.send()`

```
def corrotina():
    print('Começou')
    valor = yield
    print(f'Recebi: {valor}')
```

```
c = corrotina()
next(c)      # Começou
c.send(10)   # Recebi: 10
# StopIteration
```

Exemplificando de maneira palpável [exemplo_04.py]

Exemplo de média cumulativa adaptado do fluent python (16.3)

```python
def média():
  total = 0.0
  contador = 0
  média = None
  while True:
    entrada = yield média
    total += entrada
    contador += 1
    média = total/contador
```

```python
coro = média()
next(coro)  # preparação

coro.next(10)  # 10.0
coro.next(20)  # 15.0
```

Corrotinas

Bom, vimos com exemplos, mas não explicamos até agora do que se tratam as corrotinas.

Coroutine

From Wikipedia, the free encyclopedia

Coroutines are computer program components that generalize subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed. Coroutines are well-suited for implementing familiar program components such as cooperative tasks, exceptions, event loops, iterators, infinite lists and pipes.

Corrotinas

Bom, vimos com exemplos, mas não explicamos até agora do que se tratam as corrotinas.

Coroutine

From Wikipedia, the free encyclopedia

Co-rotinas são componentes de programas de computador que generalizam sub-rotinas para multitarefa não preemptiva, permitindo que a execução seja suspensa e reiniciada. As corrotinas são adequadas para implementar componentes de programa familiares, como tarefas cooperativas, exceções, loops de eventos, iteradores, listas infinitas e pipes.

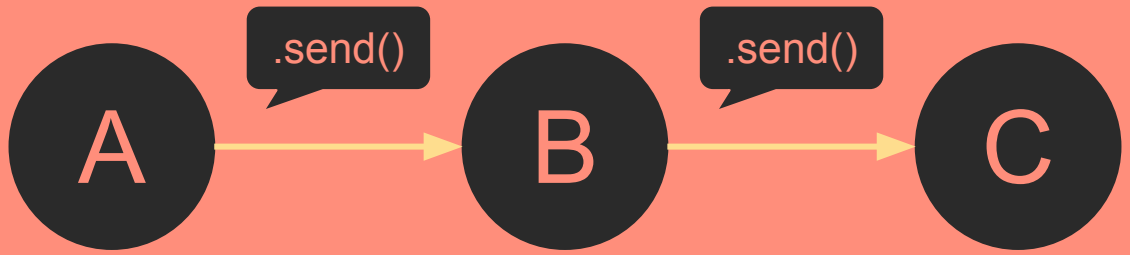
Jogando Jokempo não preemptivo

```
c = jokempo()
next(c)
c.send((0, 'papel'))
c.send((1, 'papel'))

```

```
def jokempo():
    validos = ('pedra', 'papel', 'tesoura')
    ganha = (
        ('pedra', 'tesoura'),
        ('tesoura', 'papel'),
        ('papel', 'pedra')
    )
    result = None

    while True:
        escolhas = [None, None]
        while None in escolhas:
            player, play = yield result
            result = None

            if play in validos:
                escolhas[player] = play
            else:
                result = 'Chave inválida'

        if (escolhas[0],  escolhas[1]) in ganha:
            result = ['win', 0] + escolhas

        elif (escolhas[1], escolhas[0]) in ganha:
            result = ['win', 1] + escolhas

        else:
            result = ['empate', None] + escolhas
```

Pipelines de corrotinas

Pipeline de corrotinas é quando uma corrotina chama a outra



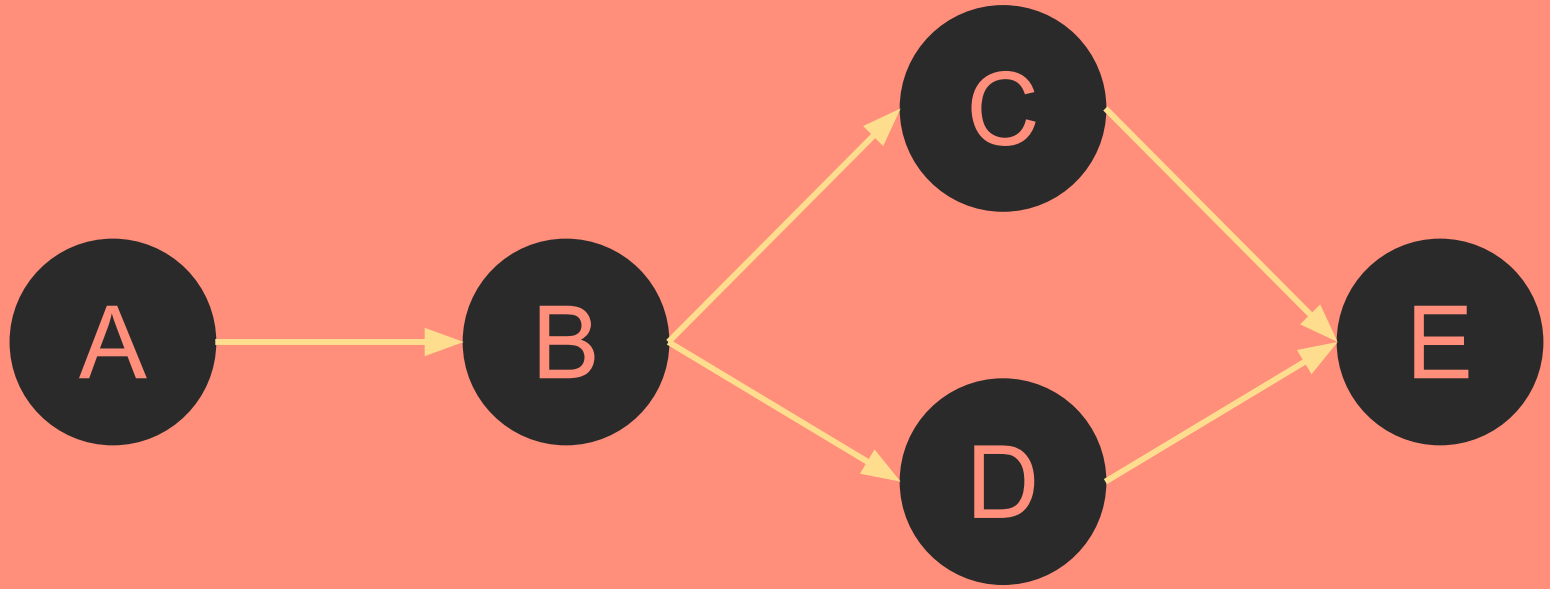
Encadeando corrotinas [exemplo_06.py]

Uma das grandes vantagens de usar corrotinas é pode escrever sub rotinas e usa-las conectadas. Um exemplo bem simples
(mastering python)

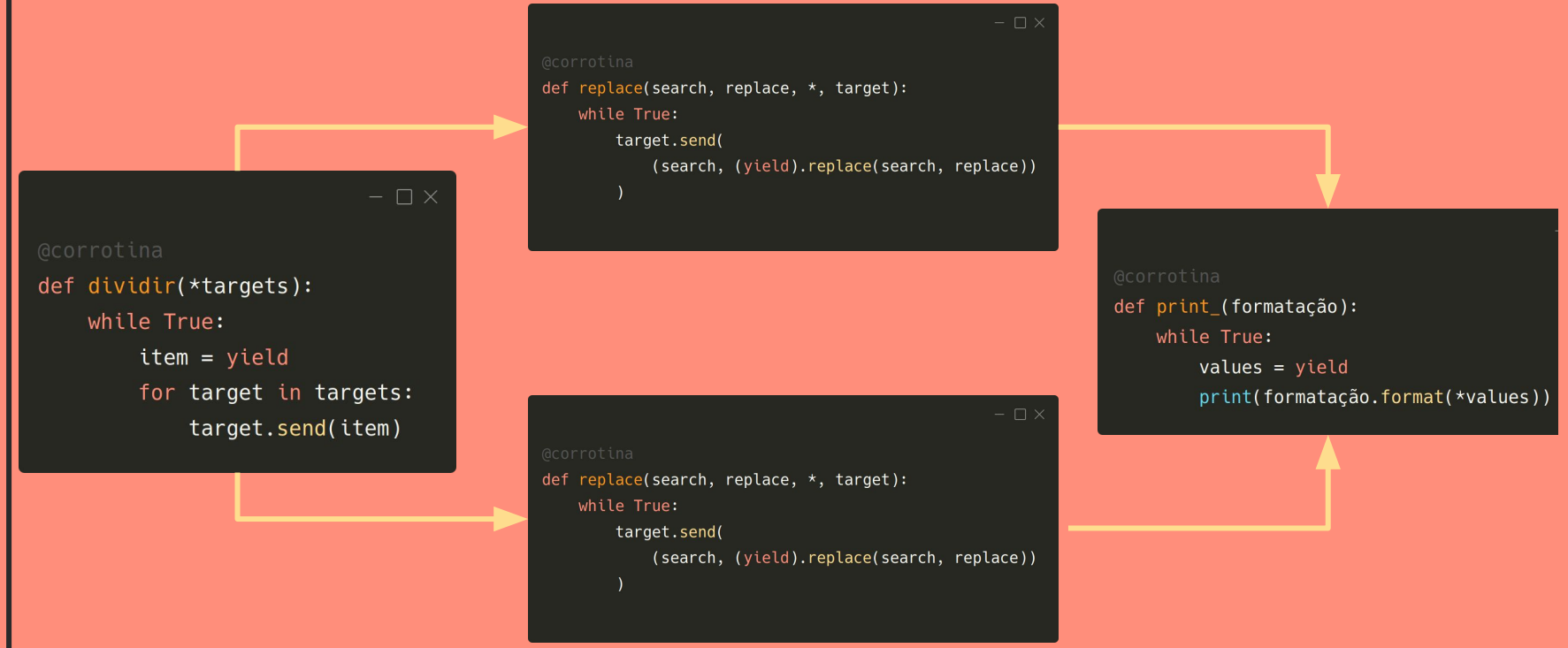
```
formatação = print_(
  'Contador: {} - Total: {} - Resultado: {}'
)

coro = média(formatação)
coro.send(10)
# Contador: 1 - Total: 10.0 - Resultado: 10.0
coro.send(20)
# Contador: 2 - Total: 30.0 - Resultado: 15.0
```

```
from corrotina import corrotina

@corrotina
def print_(formatação):
  while True:
      values = yield
      print(formatação.format(*values))

@corrotina
def média(target):
  total = 0.0
  contador = 0
  while True:
    contador += 1
    total += yield
    target.send((contador, total, total/contador))
```

Pipes bidirecionais

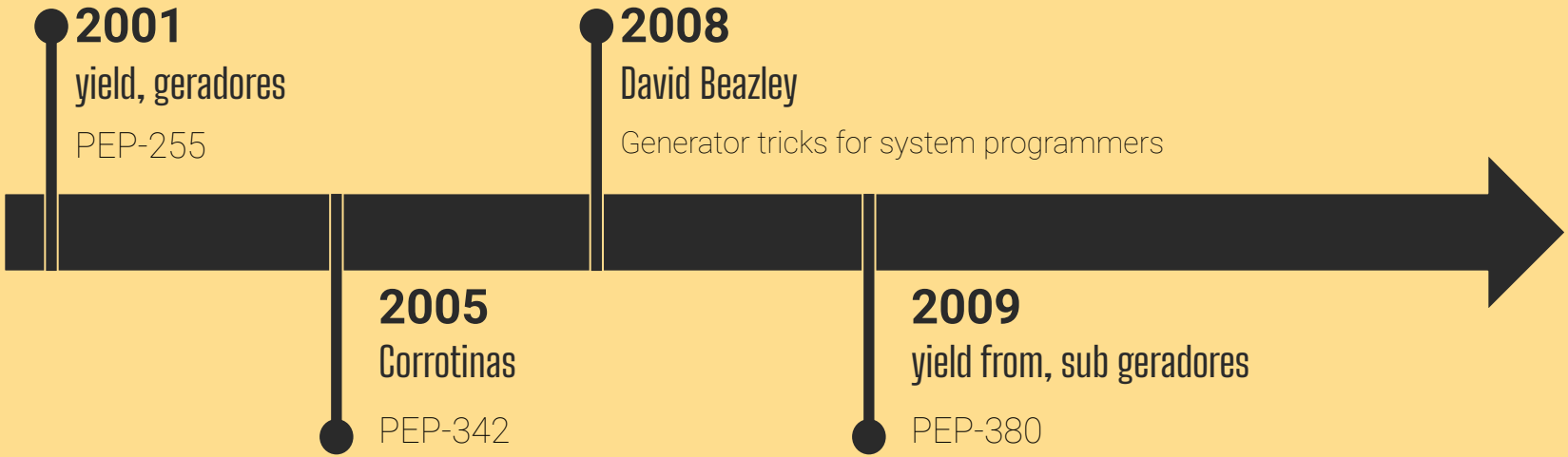


Pipes bidirecionais [exemplo_07.py]



Um exemplo prático, mas não usual

lendo logs



Avançando na linha do tempo

Subgeradores

Uma introdução a PEP-380 e ao yield from (2009)

Subgeradores

Na PEP-380 (2009) foi introduzida a ideia de sub geradores. Com isso as funções geradoras/corrotinas ganham mais 2 escopos possíveis:

- return: Agora o final de uma corrotina pode ser retornado
- yield from: Agora uma corrotina pode delegar outra corrotina

Voltando a média

```
def média():
  total = 0.0
  contador = 0
  média = None
  while True:
    entrada = yield média
    total += entrada
    contador += 1
    média = total/contador
```

A nossa corrotina de média não tinha um retorno, tudo dependia do método ‘send’.

Para finalizar essa corrotina, somente usando o método ‘close’.

Voltando a média [exemplo_08.py]

```python
def média():
    total = 0.0
    contador = 0
    média = None
    while True:
        entrada = yield média
        total += entrada
        contador += 1
        média = total/contador
```

```python
from collections import namedtuple

Result = namedtuple('Result', 'total media contador')


def média():
    total = 0.0
    contador = 0
    média = None
    while True:
        entrada = yield média
        if entrada is None:
            break
        total += entrada
        contador += 1
        média = total/contador
    return Result(total, média, contador)
```

Exemplo adaptado do fluent python 16.13

Voltando a média [exemplo_08.py]

```
def média():
  total = 0.0
  contador = 0
  média = None
  while True:
    entrada = yield média
    total += entrada
    contador += 1
    média = total/contador
```

```
from collections import namedtuple

Result = namedtuple('Result', 'total media contador')


def média():
    total = 0.0
    contador = 0
    média = None
    while True:
        entrada = yield média
        if entrada is None:
            break
        total += entrada
        contador += 1
        média = total/contador
    return Result(total, média, contador)
```

Exemplo adaptado do fluent python 16.13

Entendendo as possibilidades [exemplo_08.py]

```python
def média():
    total = 0.0
    contador = 0
    média = None
    while True:
        entrada = yield média
        if entrada is None:
            break
        total += entrada
        contador += 1
        média = total/contador
    return Result(total, média, contador)
```

Exemplo adaptado do fluent python 16.13

Agora além de poder acompanhar a média por demanda, podemos ter um “resultado final”. Porém, o retorno gera um ‘StopIteration’

```python
coro = média()
next(coro)  # preparação

coro.send(10)  # 10.0
coro.send(20)  # 15.0

try:
    coro.send(None)
except StopIteration as ex:
    print(ex.value)
```

Entendendo as possibilidades [exemplo_08.py]

```python
def média():
    total = 0.0
    contador = 0
    média = None
    while True:
        entrada = yield média
        if entrada is None:
            break
        total += entrada
        contador += 1
        média = total/contador
    return Result(total, média, contador)
```

Agora além de poder acompanhar a média por demanda, podemos ter um “resultado final”. Porém, o retorno gera um ‘StopIteration’

```python
coro = média()
next(coro)  # preparação

coro.send(10)  # 10.0
coro.send(20)  # 15.0

try:
    coro.send(None)
except StopIteration as ex:
    print(ex.value)
```

Exemplo adaptado do fluent python 16.13

Isso não parece muito prático, não é mesmo?

WTF????

```python
@corrotina
def quem_delega():
    result = yield from média()
    yield result


coro = quem_delega()

coro.send(10)    # 10.0
coro.send(20)    # 15.0
coro.send(None)  # Result(total=30.0, media=15.0, contador=2)
```

CALMA

```python
@corrotina
def quem_delega():
    result = yield from média()
    yield result


coro = quem_delega()

coro.send(10)    # 10.0
coro.send(20)    # 15.0
coro.send(None)  # Result(total=30.0, media=15.0, contador=2)
```

CALMA

yield from

O conceito do yield from é delegar a operação a uma outra corrotina.

Aqui temos que nos atentar a nomenclatura inserida na pep-380:

- Sub gerador (quem será consumido pelo yield from)
- Gerador delegante

Exemplo usado na documentação para apresentar a feature

What's new python 3.3?

Estendendo as possibilidades [exemplo_10.py]

```python
def acumular():
    contador = 0
    while True:
        valor = yield

        if valor is None:
            return contador

        contador += valor
```

O sub gerador

```python
c = acumular()
next(c)       # primer
c.send(10)
c.send(10)
c.send(None)  # StopIteration 30
```

Estendendo as possibilidades [exemplo_10.py]

```
def agregador_de_contadores(contadores):
  while True:
    contador = yield from acumular()
    contadores.append(contador)
```

O gerador delegante

```
contadores = []
agregador = agregador_de_contadores(contadores)
next(agregador)

for i in range(4):
    agregador.send(i)

agregador.send(None)
print(contadores)  # [6]
```

Estendendo as possibilidades [exemplo_10.py]

```python
contadores = []
agregador = agregador_de_contadores(contadores)
next(agregador)

for i in range(4):
    agregador.send(i)

agregador.send(None)
print(contadores)  # [6]
```

Consumindo o delegante

```python
for i in range(5):
    agregador.send(i)

agregador.send(None)

print(contadores)  # [6, 10]
```

De volta as definições

O conceito do yield from é delegar a operação a uma outra corrotina.

Aqui temos que nos atentar a nomenclatura inserida na pep-380:

- Sub gerador (quem será consumido pelo yield from)
- Gerador delegante
- Cliente / chamador/ called

O diagrama de Paul sokolovsky

De volta a média com tudo no lugar [exemplo_11.py]

Exemplo adaptado do fluent python 16.17

```python
def cliente(data):
    results = {}
    for key, values in data.items():
        group = agrupador(results, key)
        next(group)
        for value in values:
            group.send(value)
        group.send(None)

    pprint(results)
```

Cliente

```python
def agrupador(resultados, chave):
    while True:
        resultados[chave] = yield from média()
```

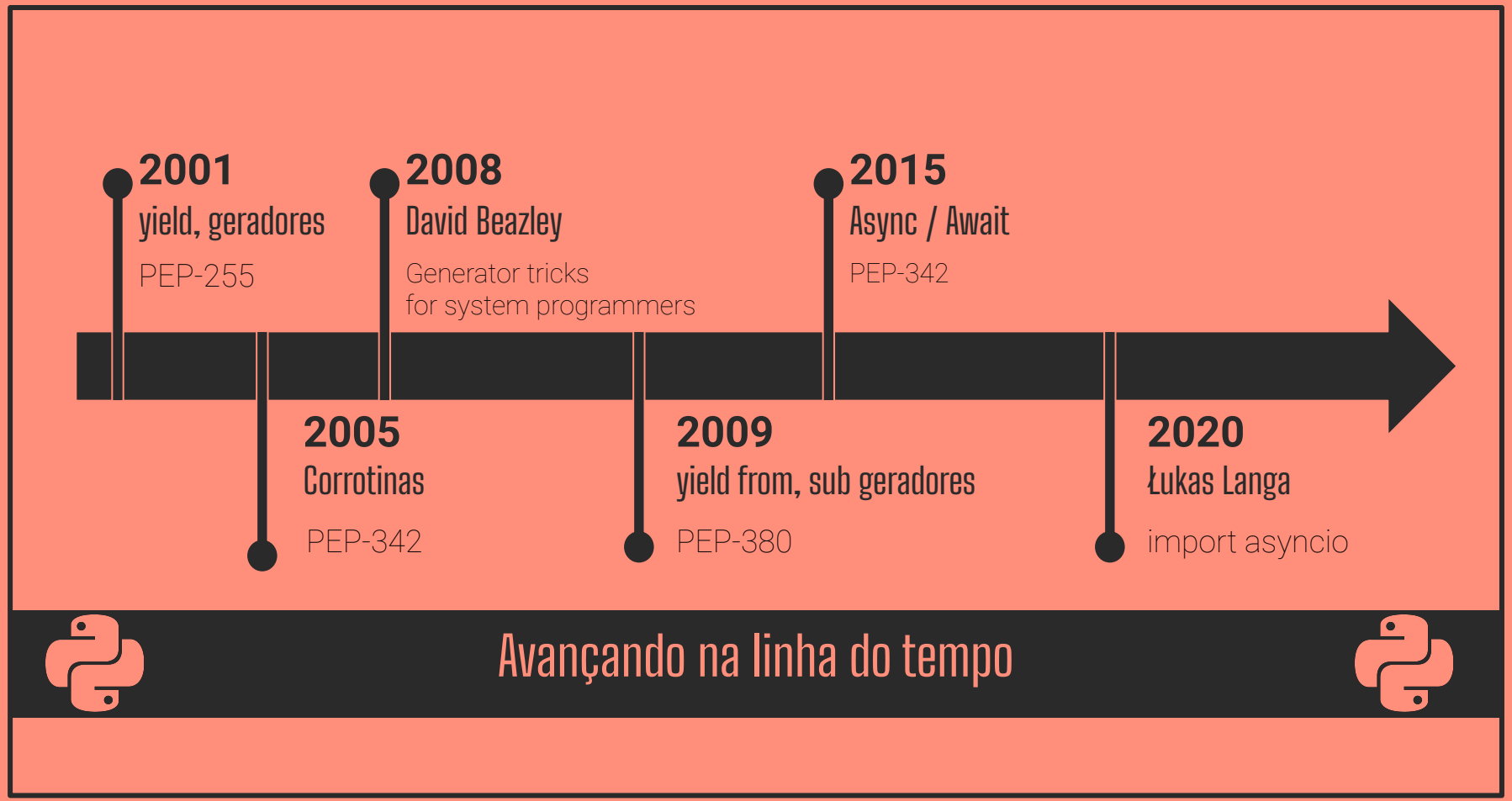
Delegante

Sub Gerador

```python
def média():
    total = 0.0
    contador = 0
    média = None
    while True:
        entrada = yield média
        if entrada is None:
            break
        total += entrada
        contador += 1
        média = total/contador
    return Result(total, média, contador)
```

PEP-492,
corrotinas com
async / await
(2015)

Async IO


2001
yield, geradores
PEP-255
2005
Corrotinas
PEP-342
2008
David Beazley
Generator tricks
for system programmers
2009
yield from, sub geradores
PEP-380
2015
Async / Await
PEP-342
2020
Łukas Langa
import asyncio
Avançando na linha do tempo