



Corrotinas

Live de Python #152



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



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

Controle de fluxo

Entendendo de
vez yield (2001)

Geradores



Funções geradoras foram imaginadas na PEP-255 (2001, py 2.2), embora já tenhamos falado de geradores na live #86, sempre é bom recapitular.

```
1 def função_geradora():  
2     yield 1  
3  
4 >>> função_geradora()  
5 # <generator object função_geradora at 0x7f567ef09040>
```



Yield



Ok, def é de definição, return é de retorno. Mas afinal, o que significa **yield**?

yield

verb

US  /ji:ld/ UK  /ji:ld/

yield *verb* (PRODUCE)



C2 [T]



to supply or produce something positive such as a profit, an amount of food or information:

<https://dictionary.cambridge.org/us/dictionary/english/yield>

Yield



Ok, def é de definição, return é de retorno. Mas afinal, o que significa **yield**?

yield
verb
US  /jiːld/ UK  /jiːld/

yield *verb* (PRODUCE)

C2 [T]

fornecer ou produzir algo positivo, como lucro, uma quantidade de comida ou informação

<https://dictionary.cambridge.org/us/dictionary/english/yield>

Yield



Esse é exatamente o contexto que usamos quando nos referimos a funções geradoras.

produzir algo:

```
1 def função_geradora():  
2     yield 1  
3     yield 2  
4     yield 3  
5  
6 gerador = função_geradora()  
7  
8 next(gerador)      # 1  
9 next(gerador)      # 2  
10 next(gerador)     # 3  
11 next(gerador)     # StopIteration
```

Contudo, entretanto.



O verbo 'to yield', pode ter significados diferentes em outros contextos

yield *verb* (STOP)



[1] US

(UK **give way**)

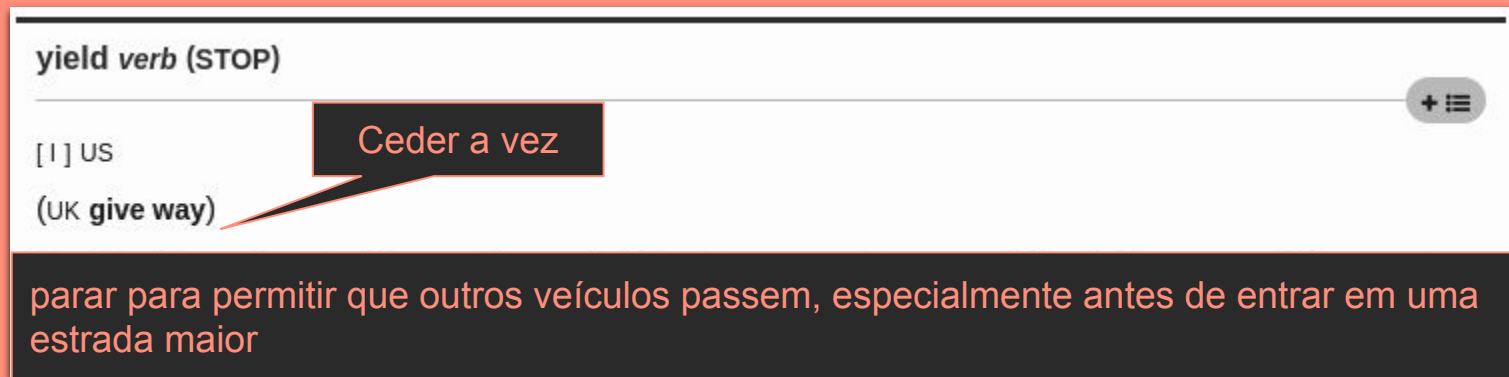
to stop in order to allow other vehicles to go past, especially before you drive onto a bigger road:

<https://dictionary.cambridge.org/us/dictionary/english/yield>

Contudo, entretanto.



O verbo 'to yield', pode ter significados diferentes em outros contextos



The screenshot shows the Cambridge Dictionary entry for the verb 'yield'. At the top, it says 'yield verb (STOP)'. Below this, there are two regional definitions: '[1] US' and '(UK give way)'. A red speech bubble points from the text 'Ceder a vez' to the '(UK give way)' definition. At the bottom of the entry, there is a dark blue box containing the text 'parar para permitir que outros veículos passem, especialmente antes de entrar em uma estrada maior'.

yield verb (STOP)

[1] US

(UK give way)

Ceder a vez

parar para permitir que outros veículos passem, especialmente antes de entrar em uma estrada maior

<https://dictionary.cambridge.org/us/dictionary/english/yield>

Yield como controle de fluxo [exemplo_01.py]



Com isso, vamos imaginar yield controlando fluxo. “Cedendo a vez, para outro gerador”

```
def contador(stop):  
    cont = 1  
    while cont <= stop:  
        yield cont  
        cont += 1
```

```
def contador_regressivo(start):  
    while start >= 1:  
        yield start  
        start -= 1
```

Yield como controle de fluxo

Agora vamos imaginar um mecanismo que use a 'cedida de vez' para escalonar [intercalar entre um e outro] os dois geradores.

```
class Scheduler:
    def __init__(self):
        self.queue = deque()

    def add_new(self, coro):
        self.queue.append(coro)

    def run(self):
        while self.queue:
            task = self.queue.popleft()
            try:
                result = next(task)
                print(f'{task=}: {result=}')
                self.queue.append(task)
            except StopIteration:
                ...
```

Lets bora pro código



Exemplo 01



Extrapolando tudo



Exemplo 02



Juntando isso com concorrência [exemplo_02.py]



De volta a 2001 e Threads para entender de vez o ceder a vez

```

s = Scheduler()
s.add_new(contador('A', 60))
s.add_new(contador('B', 60))

from threading import Thread
t = Thread(target=s.run, daemon=True)
```

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)

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

```
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.

Preparação automática de corrotinas



Com a finalidade de simplificar as chamadas, podemos automatizar a criação das corrotinas com um decorador

```
def corrotina(func):  
    def preparação(*args, **kwargs):  
        coro = func(*args, **kwargs)  
        next(coro)  
        return coro  
    return preparação
```

Decorador de primer [exemplo_05.py]

Corrotina iniciada de maneira simplificada com o uso do decorador.

Dessa maneira a corrotina já está pronta pra uso. Sem o next inicial

```
from corrotina import corrotina
```

```
@corrotina
```

```
def média():
```

```
    total = 0.0
```

```
    contador = 0
```

```
    média = None
```

```
    while True:
```

```
        entrada = yield média
```

```
        total += entrada
```

```
        contador += 1
```

```
        média = total/contador
```

```
coro = média()
```

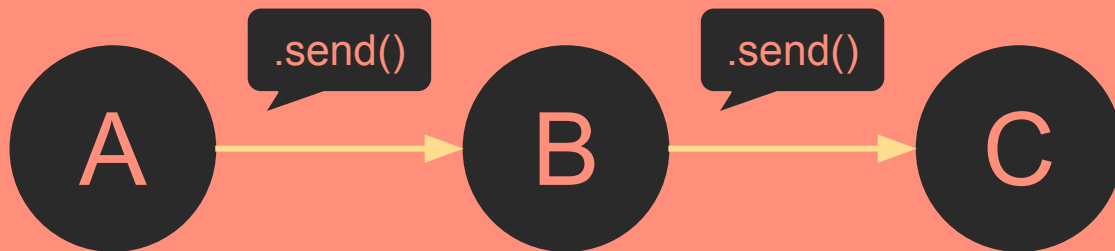
```
coro.next(10) # 10.0
```

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

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 usa-las conectadas. Um exemplo bem

```
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
```

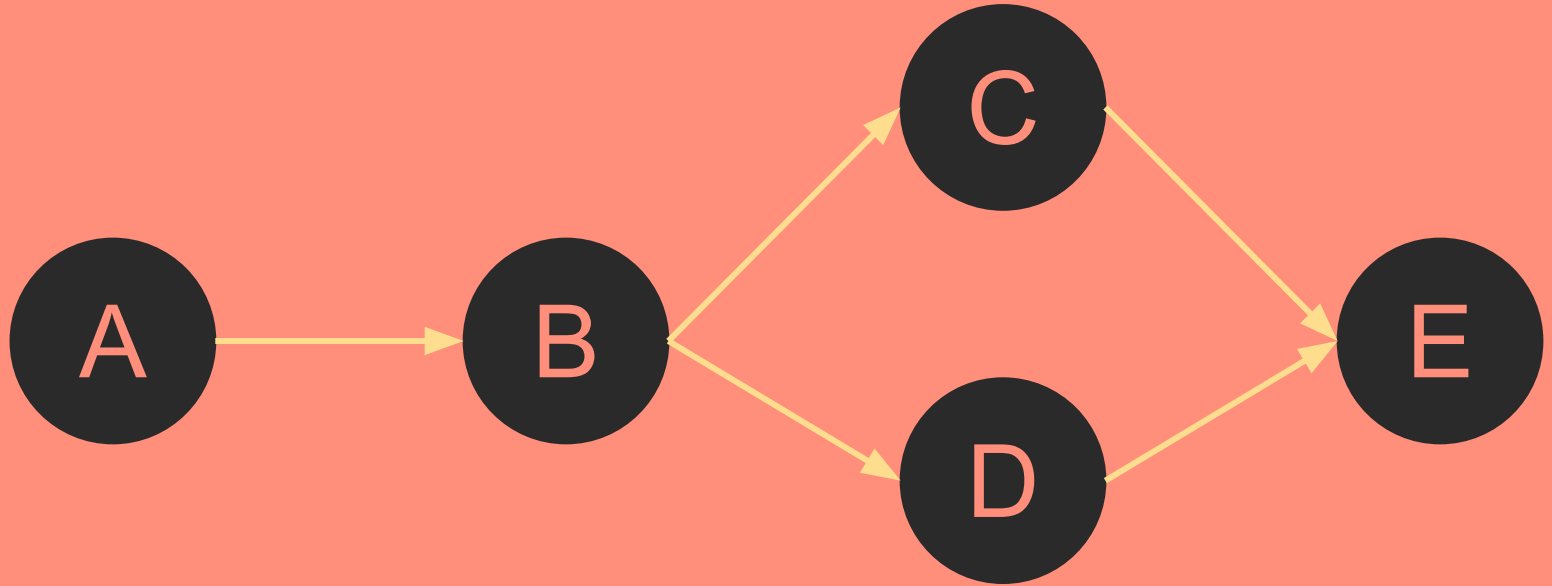
```
    while True:
```

```
        contador += 1
```

```
        total += yield
```

```
        target.send((contador, total, total/contador))
```

Pipes bidirecionais



Pipes bidirecionais [exemplo_07.py]



```
@corrotina
def dividir(*targets):
    while True:
        item = yield
        for target in targets:
            target.send(item)
```

```
@corrotina
def replace(search, replace, *, target):
    while True:
        target.send(
            (search, (yield).replace(search, replace))
        )
```

```
@corrotina
def replace(search, replace, *, target):
    while True:
        target.send(
            (search, (yield).replace(search, replace))
        )
```

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

Referências



- Geradores (PEP-255)
- Yield como fluxo (Cookbook - 12.12)
- Geradores melhorados (PEP-342)
 - Counter gi (Langa aula 4)
 - Média cumulativa (Fluent python)
 - Exceptions (PEP-288)
 - getargspec recipe (Hettinger)
- Sub geradores (PEP-380)
 - Delegate (Ramalho - 16.17)
 - Pipelines (Curso Beazley)
- Asyncio Coros (PEP-492)
 - Langa aula 4
 - HTTPX