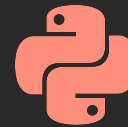




Concorrência estruturada com Trio

Live de Python #242



1. Um tico de história

Mas e o AsyncIO?

2. Trio

Uma introdução!

3. Baterias inclusas

Algumas coisas que vem por padrão

4. Testes

Uma pincelada, por que é muita coisa mágica



picpay.me/dunossauro



apoia.se/livedepython



pix.dunossauro@gmail.com



Ajude o projeto <3



Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alemão, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alisson Souza, Alysson Oliveira, Ana Carneiro, Ana Padovan, Andre Azevedo, Andre Mesquita, Andre Paula, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bernardo At, Bernardo Fontes, Bruno Almeida, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Christiano Moraes, Christian Semke, Damianth, Daniel Freitas, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Goivinho, Davi Souza, Delton Porfiro, Denis Bernardo, Diego Farias, Diego Guimarães, Dilenon Delfino, Diogo Paschoal, Diogo Silva, Edgar, Eduardo Silveira, Eduardo Tolmasquim, Emerson Rafael, Eneas Teles, Ennio Ferreira, Erick Andrade, Érico Andrei, Everton Silva, Fabiano, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Fabricio Patrocinio, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Firehouse, Flávio Meira, Francisco Neto, Francisco Silvério, Gabriel Espindola, Gabriel Mizuno, Gabriel Paiva, Gabriel Simonetto, Geandreson Costa, Geizelder, Gilberto Abrao, Giovanna Teodoro, Giuliano Silva, Guilherme Beira, Guilherme Felitti, Guilherme Gall, Guilherme Silva, Guionardo Furlan, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Italo Silva, Janael Pinheiro, Jean Victor, Joelson Sartori, Jônatas Oliveira, Jônatas Silva, Jon Cardoso, Jorge Silva, José Gomes, Joséito Júnior, Jose Mazolini, Josir Gomes, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kálita Lima, Kaneson Alves, Leandro Miranda, Leandro Silva, Leo Ivan, Leonardo Mello, Leonardo Nazareth, Leon Solon, Looooooooo, Luancomputacao Roger, Lucas Adorno, Lucas Carderelli, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Lucas Valino, Luciano Filho, Luciano Ratamero, Luciano Teixeira, Luis Alves, Luis Eduardo, Luiz Duarte, Luiz Lima, Luiz Paula, Luiz Perciliano, Mackilem Laan, Marcelo Araujo, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Maria Clara, Marina Passos, Mateus Lisboa, Mateus Ribeiro, Mateus Silva, Matheus Silva, Matheus Vian, Mauricio Nunes, Mírian Batista, Mlevi Lsantos, Moisés Ferreira, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Pydocs Pro, Pytonyc, Rafael Lopes, Rafael Romão, Ramayana Menezes, Regis Santos, Renato Oliveira, Rene Bastos, Ricardo Silva, Richard Carvalho, Riverfount, Rjribeiro, Robson Maciel, Rodrigo Barretos, Rodrigo Freire, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Ribeiro, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Samuel Santos, Téó Calvo, Thaynara Pinto, Thiago Araujo, Thiago Borges, Thiago Curvelo, Thiago Souza, Tiago Minuzzi, Tony Dias, Tyrone Damasceno, Valcilon Silva, Valdir Tegon, Vcwild, Vicente Marcal, Vinícius Costa, Vinicius Stein, Walter Reis, Willian Lopes, Wilson Duarte, Wilson Neto, Zeca Figueiredo



Obrigado você



Mas e o AsyncIO?

História

Asyncio e sua evolução constante



Não é de hoje que vocês sabem que sou apaixonado pelas corrotinas do python.

Geradores e uma Introdução histórica à corrotinas com...
Eduardo Mendes
Pública ▾
4 vídeos · 1.305 visualizações · Última atualização em 8 d...

Ordenar

- Live de Python #151 - Desvendando o yield e as funções geradoras**
Eduardo Mendes · 7,4 mil visualizações · Transmitido há 2 anos
- Live de Python #152 - Uma introdução histórica à corrotinas**
Eduardo Mendes · 5,7 mil visualizações · Transmitido há 2 anos
- Live de Python #153 - Uma introdução histórica à corrotinas PARTE 2**
Eduardo Mendes · 2,7 mil visualizações · Transmitido há 2 anos
- Live de Python #154 - Uma introdução histórica à corrotinas PARTE 3 (AsyncIO)**
Eduardo Mendes · 4,4 mil visualizações · Transmitido há 2 anos

<https://www.youtube.com/playlist?list=PLOQgLBuj2-3J4lRxaIwXhRMU6UPoigf9>

Asyncio e sua evolução constante



Nessa live discutimos o conceito de TaskGroups e *excpetions



<https://youtu.be/7BfiLFiqGRU>

Mais recentemente



Conversamos sobre como usar asyncio para requests async



<https://youtu.be/V4hSLZRCCoE>

Em 2015 — Esse era o AsyncIO — 3.5



```
1  import asyncio
2
3
4  @asyncio.coroutine
5  def main():
6      print('antes do sleep')
7      yield from asyncio.sleep(1)
8      print('depois do sleep')
9
10 loop = asyncio.get_event_loop()
11 asyncio.ensure_future(main())
12 loop.run_forever()
13 loop.close()
```

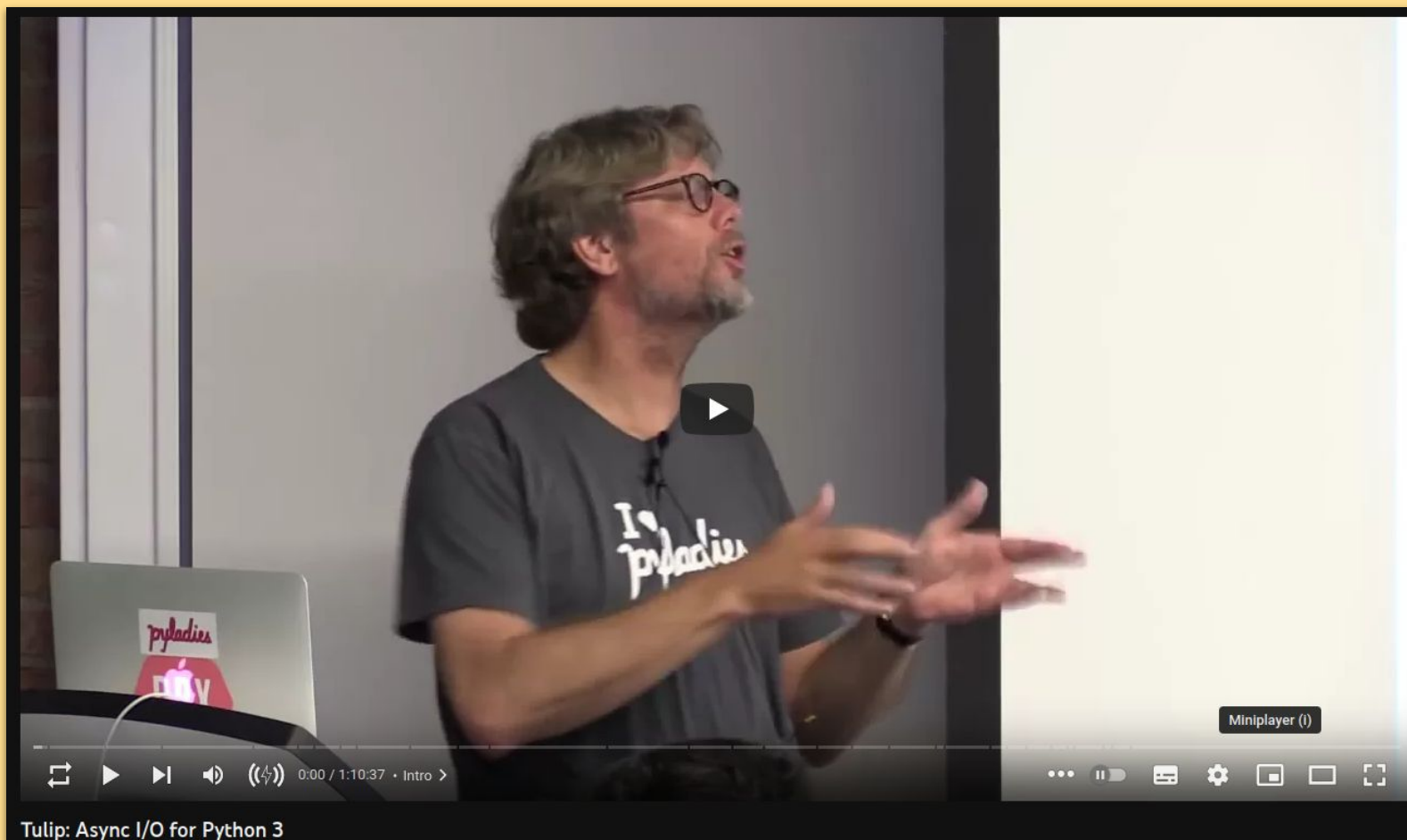
<https://docs.python.org/3.5/library/asyncio.html>

Em 2015 — Esse era o AsyncIO — 3.5



```
1  import asyncio
2
3
4  @asyncio.coroutine
5  def main():
6      print('antes do sleep')
7      yield from asyncio.sleep(1)
8      print('depois do sleep')
9
10 loop = asyncio.get_event_loop()
11 asyncio.ensure_future(main())
12 loop.run_forever()
13 loop.close()
```

Uma palestra legal do Guido em 2012 (3.3)

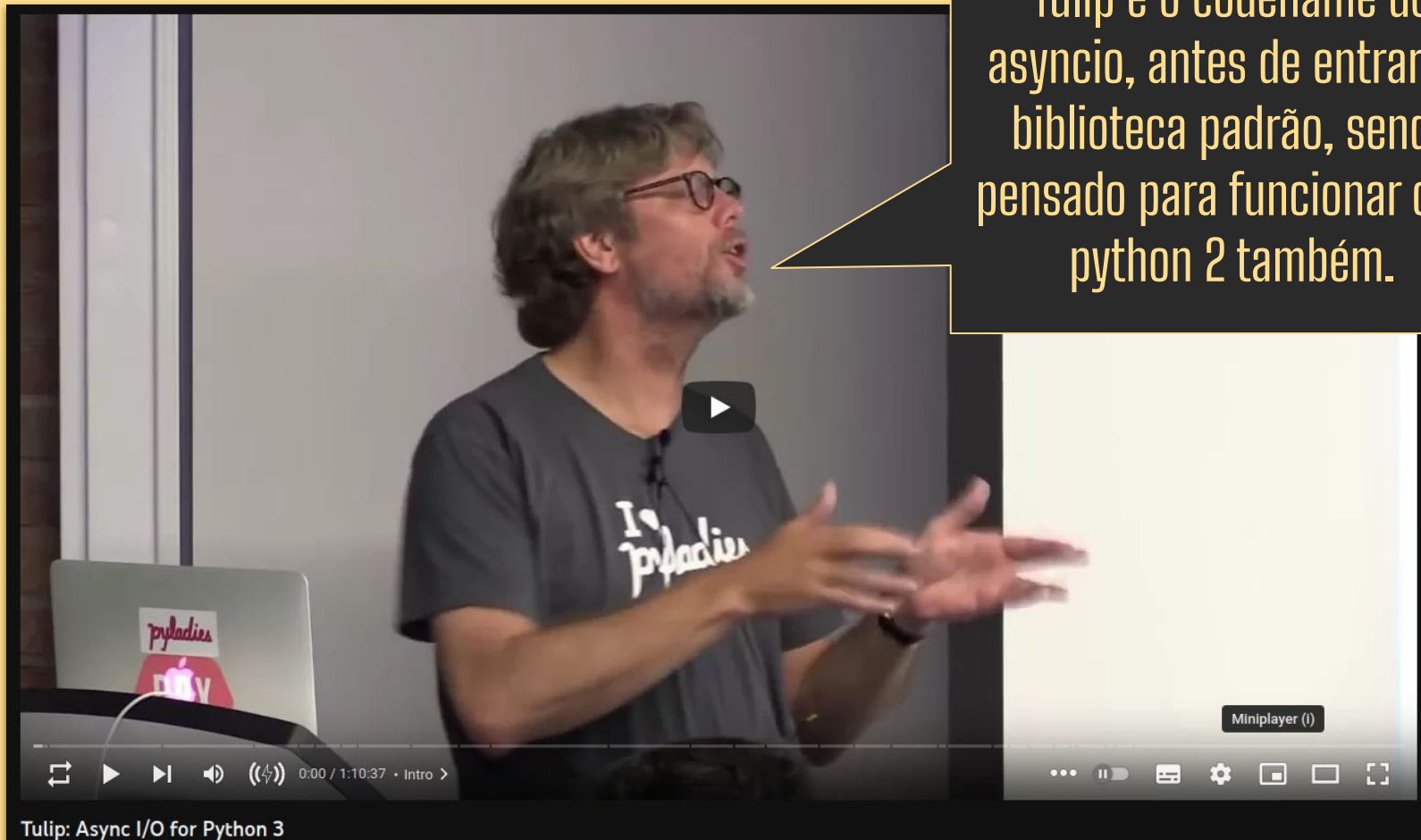


<https://youtu.be/1coLC-MUCJc>

Uma palestra legal do Guido em 2012 (3.3)

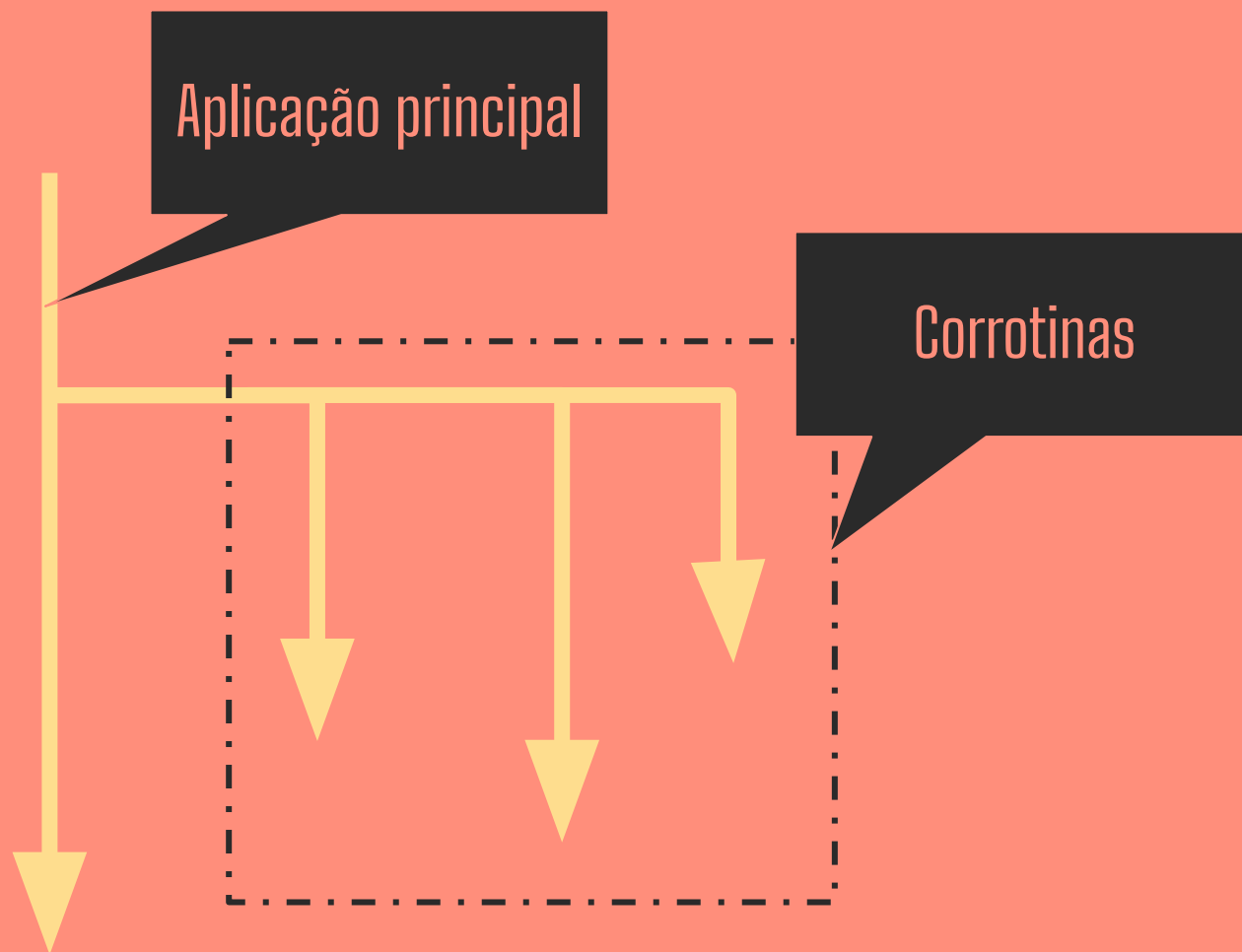


Tulip é o codename do asyncio, antes de entrar na biblioteca padrão, sendo pensado para funcionar com python 2 também.

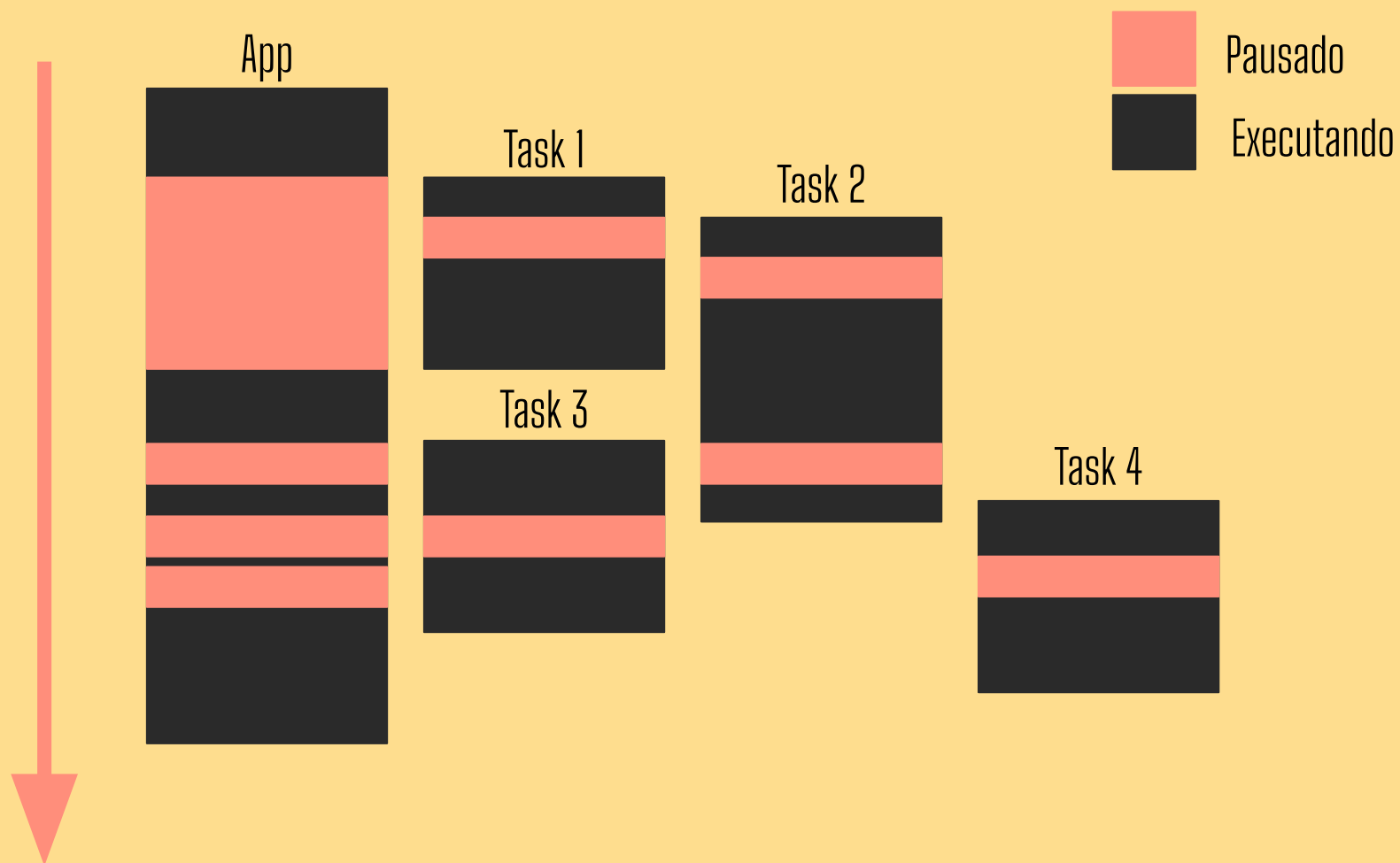


<https://youtu.be/1coLC-MUCJc>

O fluxo da concorrência



O AsyncIO é cooperativo



Comportamentos estranhos [exemplo_00.py]



```
1 import asyncio
2
3 async def corrotina(i):
4     print(f'Iniciando corrotina {i}')
5     await asyncio.sleep(i)
6     print(f'Terminando corrotina {i}')
7
8 async def main():
9     asyncio.create_task(corrotina(1))
10    asyncio.create_task(corrotina(2))
11    asyncio.create_task(corrotina(3))
12
13    for x in range(10):
14        print(f'Ihuu {x}')
15
16 asyncio.run(main())
```

Comportamentos estranhos [exemplo_00.py]



```
1 import asyncio
2
3 async def corrotina(i):
4     print(f'Iniciando corrotina {i}')
5     await asyncio.sleep(i)
6     print(f'Terminando corrotina {i}')
7
8 async def main():
9     asyncio.create_task(corrotina(1))
10    asyncio.create_task(corrotina(2))
11    asyncio.create_task(corrotina(3))
12
13    for x in range(10):
14        print(f'Ihuu {x}')
15
16    asyncio.run(main())
```

```
python exemplo_slides/exemplo_00.py
```

```
Ihuu 0
```

```
Ihuu 1
```

```
Ihuu 2
```

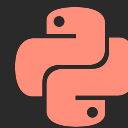
```
Ihuu ...
```

```
Iniciando corrotina 1
```

```
Iniciando corrotina 2
```

```
Iniciando corrotina 3
```


Em 2016

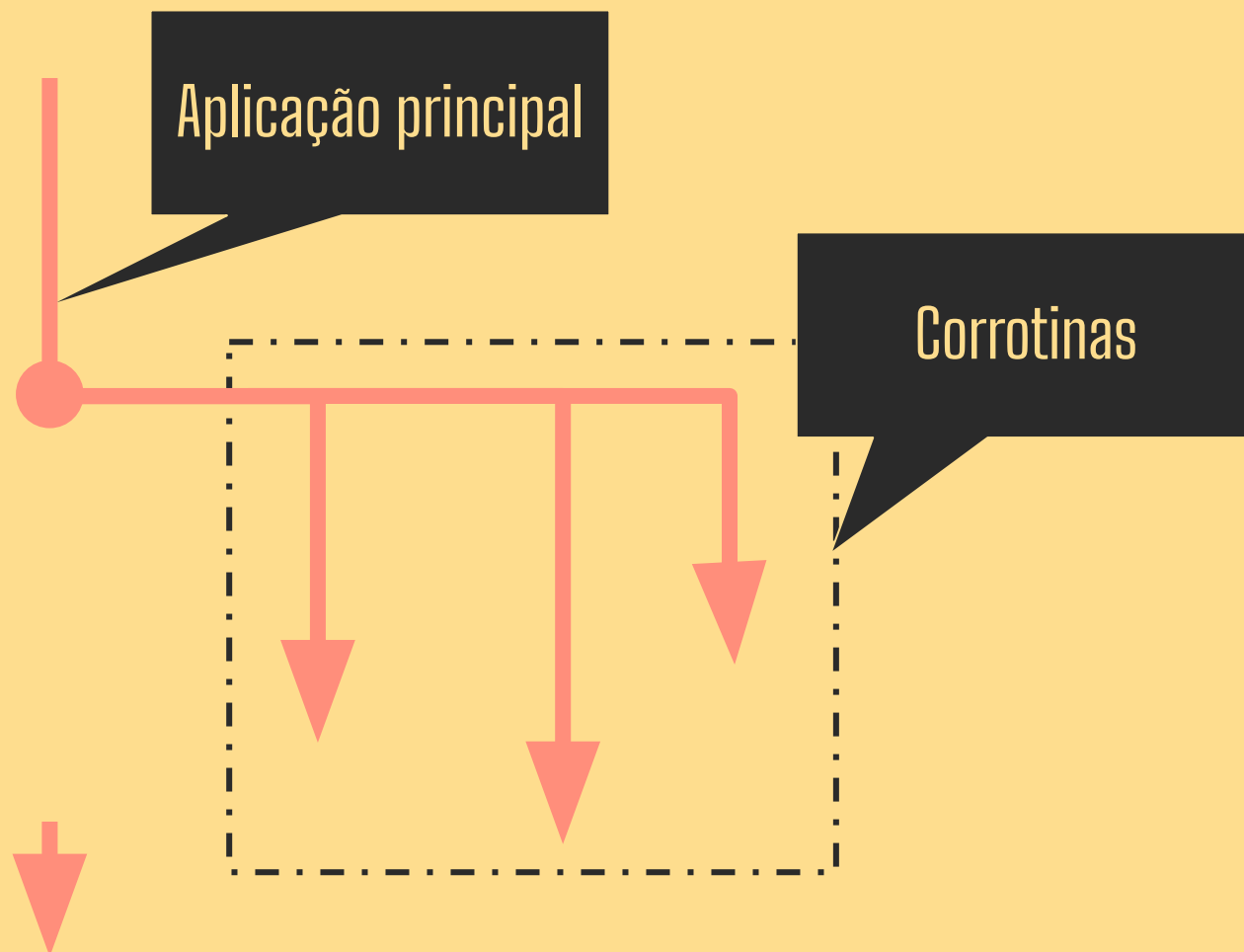


Martin Sústrik (criador do ZeroMQ)
fundamentou um termo chamado
"**Structured concurrency**" ou concorrência
estruturada. Com seu trabalho na biblioteca
[ibmill](#) (c++), baseado nas gorrotines da
linguagem Go.

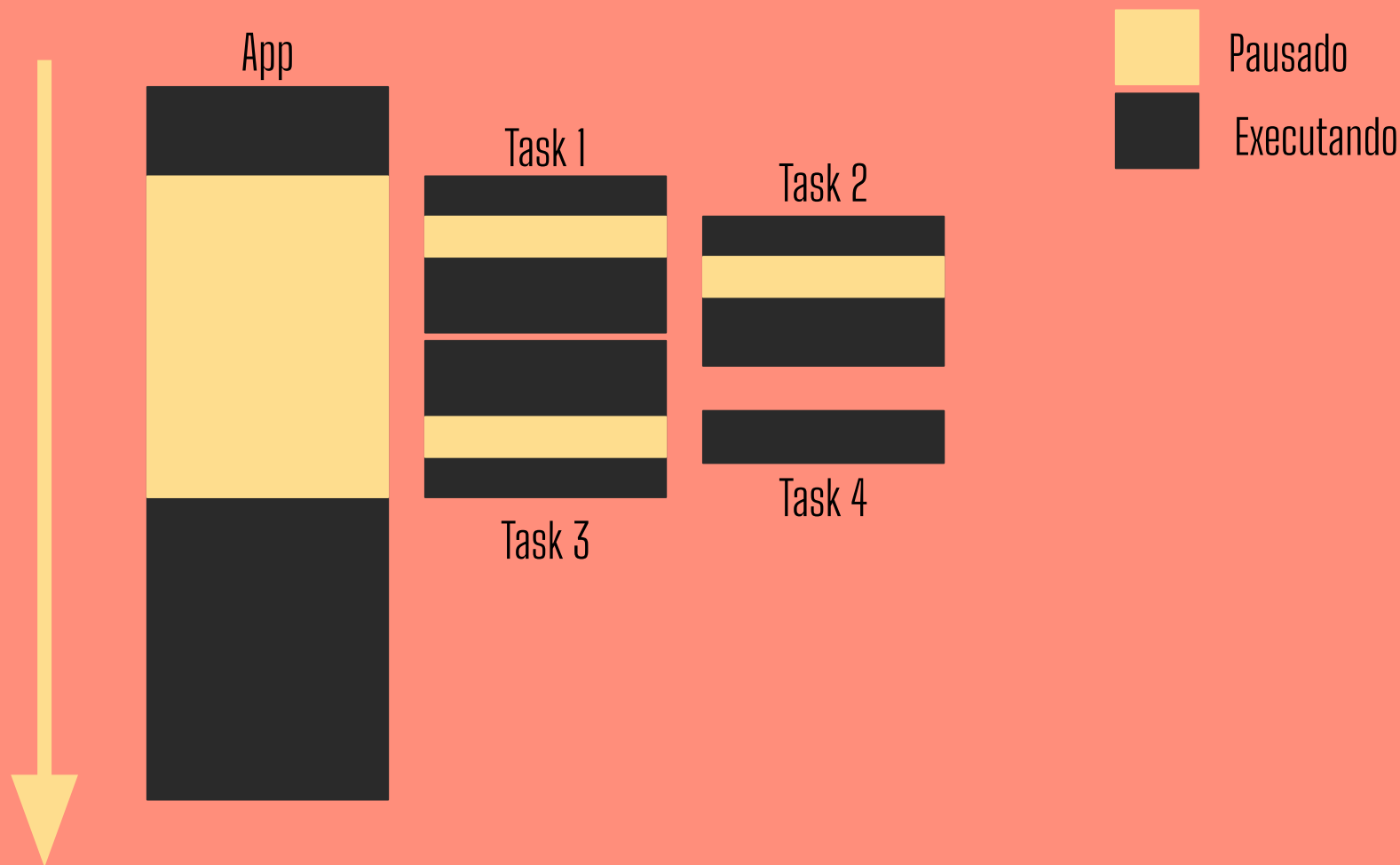


Martin Sústrik - @sustrik

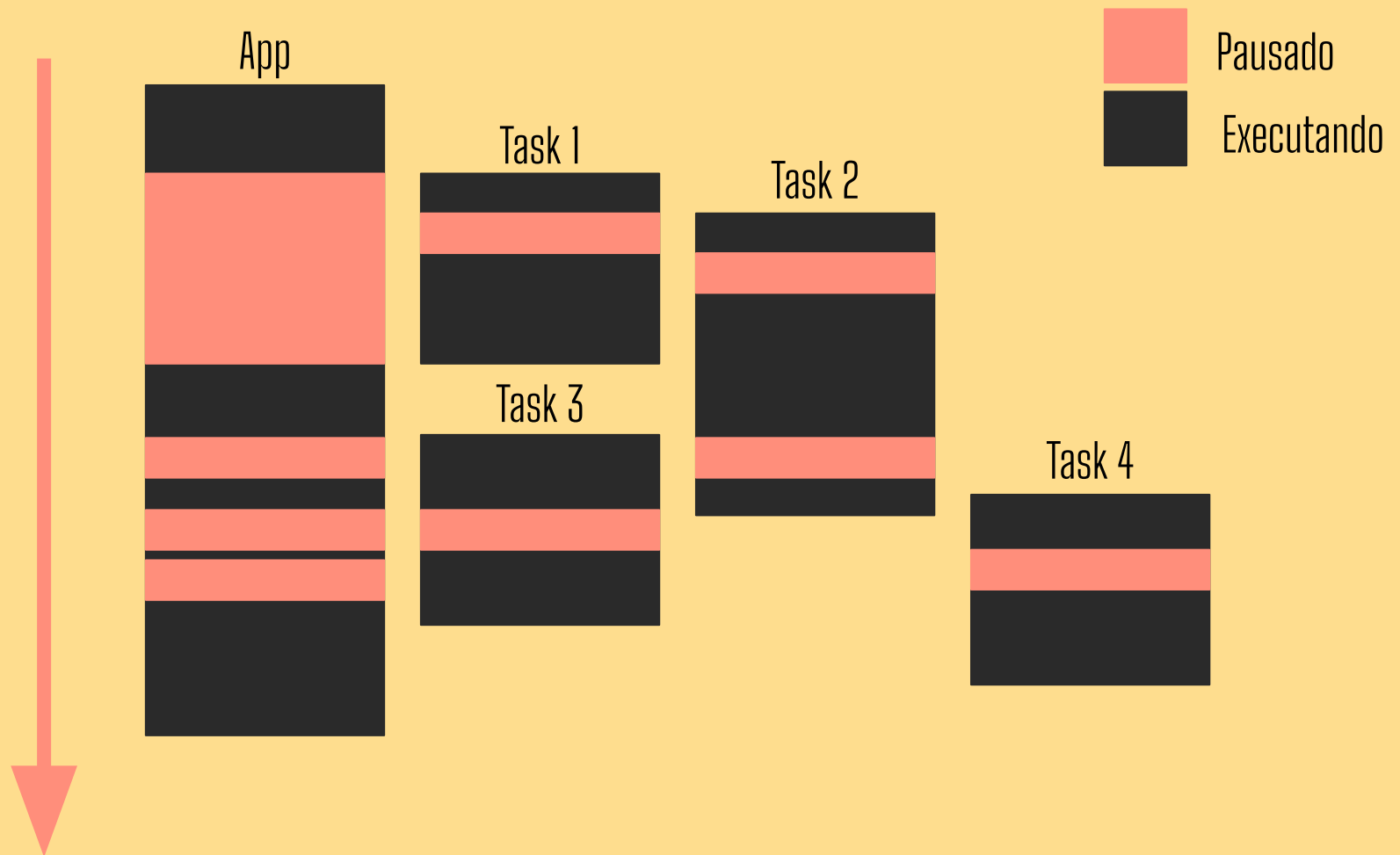
Estruturação das corrotinas



Uma pausa pra assincronia!



Uma volta ao caos



A ideia de estruturado



Estruturado vem de programação estruturada. Onde uma coisa é sequencial a outra.

Nesse caso fazemos toda a loucura async em um lugar só e voltamos para colher os resultados.

Isso tem diversos benefícios:

- Poder **cancelar** todas as tarefas sem medo
- Todo o fluxo será sempre **aguardado**
- Podemos capturar **exceções** em um lugar só
- Garantia de que nada ficará **travado em background**

Em 2017



Nathaniel Smith, após trabalhar na biblioteca **curio**, introduz a concorrência estruturada no python com o conceito cunhado por ele de **Nursey**, ou berçário, em português.

Com isso, nasce a biblioteca **trio** em 2017. Ele se junta ao time de core developers do python em 2018.



Nathaniel J. Smith

Anuncio oficial em março de 2017

Trio

Uma introdução!

Trio



Uma biblioteca para lidar com E/S assíncrona em python.

- Foi inicialmente criado por Nathaniel J. Smith em 2017
- Atualmente está na versão 0.22.1: lançada ontem
- Está sobre licença MIT ou Apache 2
- Lema:
 - *A friendly Python library for async concurrency and I/O*
 - Uma biblioteca Python amigável para concurrency async e I/O





O trio tem diversas premissas interessantes:

- Facilidade de uso (Amigável)
 - Tudo é resolvido com corrotinas
 - Não existe uso de callbacks
 - Não existem futures
- Concorrência estruturada
 - Tudo deve acontecer dentro das Nurseries (já vamos chegar lá)
 - Garante o cancelamento correto
- Baterias inclusas
 - Bons recursos para testes
 - Gerenciamento de recursos:
 - arquivos, canais de memória, streams, tcp, ssh e socket

pip install trio



Instalação



Nosso "olá mundo!" [exemplo_01.py]



```
1  import trio
2
3  async def main():
4      print('Olá Trio!')
5
6  trio.run(main)
```



3.5+

Comparação boba com asyncio [0]



```
1  import asyncio
2
3  async def main():
4      print('Olá Trio!')
5
6  asyncio.run(main())
```



3.7+

Comparação boba com asyncio [1] — Ou então...



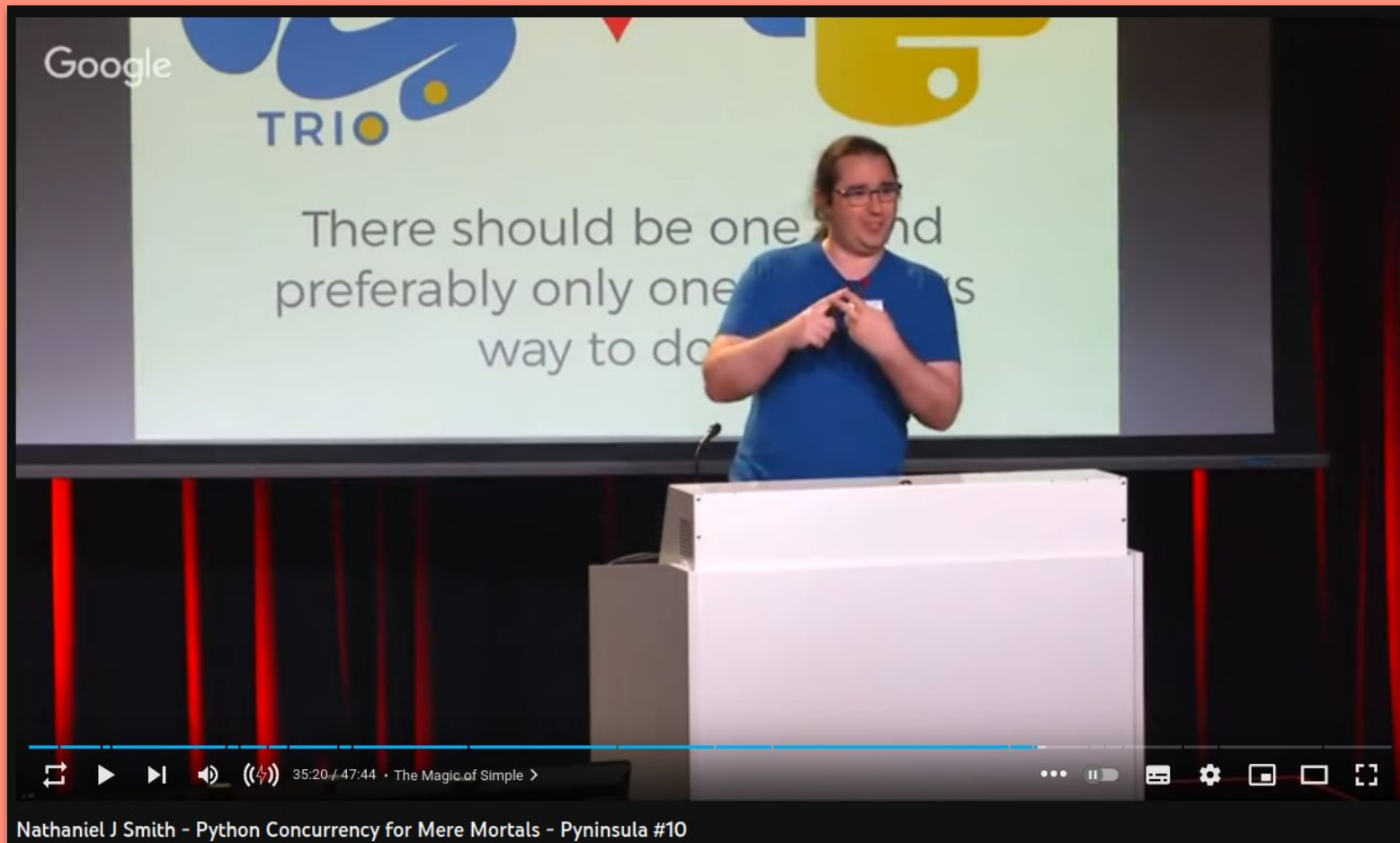
```
1  import asyncio
2
3  async def main():
4      print('Olá Future!')
5
6
7  loop = asyncio.get_event_loop()
8  future = asyncio.ensure_future(main())
9  future.add_done_callback(
10     lambda future: print(f'Future: {future.get_name()} completo')
11 )
12
13 try:
14     loop.run_until_complete(future)
15 finally:
16     loop.close()
```

Comparação boba com asyncio [2] — Ou ainda...



```
1  import asyncio
2
3  async def main():
4      print('Olá Task!')
5
6
7  loop = asyncio.get_event_loop()
8  task = asyncio.create_task(main())
9  task.add_done_callback(
10     lambda task: print(f'Future: {task.get_name()} completo')
11 )
12
13 loop.run_forever() # Isso vai dar erro...
```

Deve haver uma e apenas uma maneira óbvia...



Nathaniel J Smith - Python Concurrency for Mere Mortals (2018): <https://youtu.be/i-R704I8ySE>

O nosso olá mundo! POKEMONS!!!!

[exemplo_02.py]



```
1  from trio import run, open_nursery
2  from httpx import AsyncClient
3
4  async def fetch(url):
5      async with AsyncClient() as client:
6          response = await client.get(url, timeout=None)
7          data = response.json()
8          print(data['id'], data['name'])
9
10 async def main():
11     base_url = 'https://pokeapi.co/api/v2/pokemon/{}'
12     async with open_nursery() as nursery:
13         for n in range(1, 11):
14             poke_url = base_url.format(n)
15             nursery.start_soon(fetch, poke_url)
16
17 run(main)
```


0 nosso olá mundo! POKEMONS!!!! [exemplo_02.py]



```
1  from trio import run, open_nursery
2  from httpx import AsyncClient
3
4  async def fetch(url):
5      async with AsyncClient() as client:
6          response = await client.get(url, timeout=None)
7          data = response.json()
8          print(data['id'], data['name'])
9
10 async def main():
11     base_url = 'https://pokeapi.co/api/v2/pokemon/{}'
12     async with open_nursery() as nursery:
13         for n in range(1, 11):
14             poke_url = base_url.format(n)
15             nursery.start_soon(fetch, poke_url)
16
17 run(main)
```

Live 234

Nursery



O contexto de **Nursery** (*berçário*) é um *lugar onde você deixa as crianças e faz suas próprias tarefas*. A tal da concorrência estruturada!

- Ele pausa a execução do programa para executar todas as corrotinas
- Caso uma corrotina falhe, ele levantará a exceção correta
- Só termina quando todas as corrotinas forem finalizadas
- **start_soon()**: Inicia uma corrotina
- **child_tasks**: Mostra as tarefas em execução

```
10  async def main():
11      base_url = 'https://pokeapi.co/api/v2/pokemon/{'
12      async with open_nursery() as nursery:
13          for n in range(1, 11):
14              poke_url = base_url.format(n)
15              nursery.start_soon(fetch, poke_url)
```

Se algo der errado [exemplo_03.py]



Caso uma das tarefas falhe, a Nursery levantará um grupo de exceções. Fazendo com que o debug fique mais simples, além de claro, cancelar todas as outras operações.

```
10  async def main():
11      url = 'https://pokeapi.co/api/v2/pokemon/{'
12      try:
13          async with open_nursery() as nursery:
14              for n in range(1, 11):
15                  poke_url = url.format(n)
16                  nursery.start_soon(fetch, poke_url)
17
18      # Exceptions de todo o nursery
19      except *ConnectTimeout as ex:
20          print(ex)
21          for error in ex.exceptions:
22              print(f'ERROR: {error.request.url}')
```

Caso as coisas não façam sentido [exemplo_04.py]



A execução das tasks podem ser instrumentadas, mostrando o que aconteceu em cada etapa.

```
1  from trio import run
2  from trio.abc import Instrument
3
4  class Tracer(Instrument):
5      ...
6
7  async def main():
8      ...
9
10 run(main, instruments=[Tracer()])
```

<https://trio.readthedocs.io/en/latest/tutorial.html#task-switching-illustrated>
<https://trio.readthedocs.io/en/latest/reference-lowlevel.html#instrumentation>

Inclusas

Bateri
as

Baterias inclusas



O trio conta com diversas coisas embutidas que facilitam a programação assíncrona. Como:

- **abc.Instrument**: Que vimos no slide passado
- **open_file()**: Leitura e escrita de arquivos async
- **Event()**: Avisa outras corrotinas quando iniciar!
- **Timeouts**: Algumas funcionalidades para timeout (moves)
- **memory_channel()**: Uma forma de corrotinas produzirem e consumirem recursos compartilhados

Manipulação de arquivos [exemplo_05.py]



Suponhamos que nossa corrotina de fetch agora necessite armazenar os resultados dos pokemons e um csv. Podemos usar **await open_file()**

(sem aiofiles :)

```
1  from trio import open_file, open_nursery, run
2
3  async def fetch_and_write(url):
4      async with AsyncClient() as client:
5          response = await client.get(url, timeout=.1)
6          data = response.json()
7
8      async with await open_file('pokes.csv', 'a') as f:
9          await f.write(f"{data['id']},{data['name']}\n")
10
```

Comunicação entre corrotinas! [exemplo_06.py]



```
1  from trio import open_memory_channel, open_nursery, run
2
3  async def producer(send_channel):
4      with send_channel:
5          for i in range(10):
6              await send_channel.send(f'message {i}')
7
8  async def consumer(receive_channel):
9      with receive_channel:
10         async for value in receive_channel:
11             print(f'got value {value}')
12
13  async def main():
14      async with open_nursery() as nursery:
15          send_channel, receive_channel = open_memory_channel(0)
16          nursery.start_soon(producer, send_channel)
17          nursery.start_soon(consumer, receive_channel)
```


Event [exemplo_07.py]



```
1  import trio
2
3  async def task_a(event):
4      await trio.sleep(10)
5      event.set() # Diz que o evento começou
6
7  async def task_b(event):
8      print('Esperando a liberação do evento')
9      await event.wait() # trava até o evento acontecer!
10     print('Evento liberado')
11
12  async def main():
13      async with trio.open_nursery() as nursery:
14          event = trio.Event()
15          nursery.start_soon(task_a, event) # Controla o evento
16          nursery.start_soon(task_b, event) # Aguarda o evento
17          nursery.start_soon(task_b, event) # Aguarda o evento
```

Timeouts!



```
1  from trio import move_on_after, open_file, open_nursery
2  from httpx import AsyncClient
3
4
5  async def fetch_and_write(url):
6      async with AsyncClient() as client:
7
8          with move_on_after(3):
9
10             response = await client.get(url)
11             data = response.json()
12             async with await open_file('pokes.csv', 'a') as f:
13                 await f.write(f"{data['id']},{data['name']}\n")
```

Por que podemos

Testes

Clock



O trio conta com algumas funcionalidades mágicas para testes:


- `magic_clock`: Manipula o tempo dos awaits
- `autojump_clock`: Pula todos os sleeps
- `trio.ToSlowError`: Erro para quando existe movimentação após x tempo

Para simplificar as coisas, vamos usar o `pytest-trio`, mantido pelo próprio time do trio

```
pip install pytest-trio
```

Clock [teste_00.py]

```
1  import trio
2  import pytest
3
4  async def sleep_test():
5      await trio.sleep(5_000)
6      return 'PEI!'
7
8  @pytest.mark.trio
9  async def test_sleep_test(autojump_clock):
10     """Teste sleep_test()."""
11     result = await sleep_test()
12     assert result == 'PEI!'
13
```



move, fail



```
1  import trio
2  import pytest
3
4  async def fetch():
5      with trio.fail_after(10):
6          await trio.sleep(11)
7          # ...
8          return 'xpto'
9
10 @pytest.mark.trio
11 async def test_fetch(autojump_clock):
12     with pytest.raises(trio.TooSlowError):
13         await fetch()
```

```
1  import trio
2  import pytest
3
4  async def meromy_recever(receiver):
5      async with receiver:
6          async for value in receiver:
7              print(f'receiver: {value}')
8
9
10 @pytest.mark.trio
11 async def test_memory(capsys, nursery):
12     sender, receiver = trio.open_memory_channel(0)
13
14     nursery.start_soon(meromy_recever, receiver)
15
16     await sender.send('Mensagem')
17     sender.close()
18
19     out, err = capsys.readouterr()
20     assert out == 'receiver: Mensagem\n'
```

```
pytest test_<>.py
```



Para rodar os testes!





picpay.me/dunossauro



apoia.se/livedepython



pix.dunossauro@gmail.com



Ajude o projeto <3

