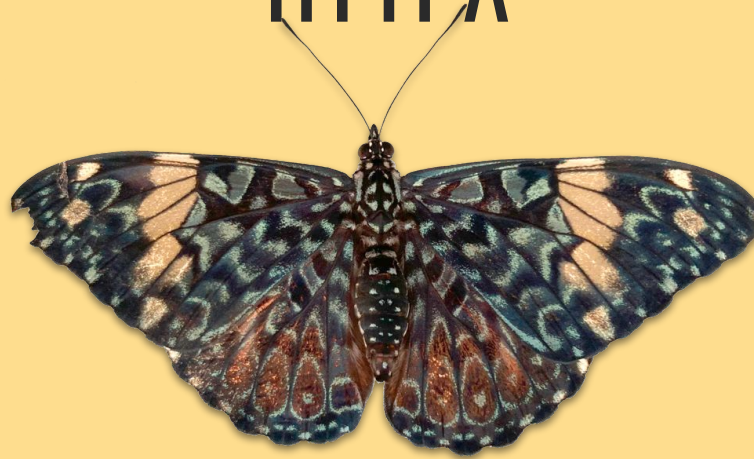
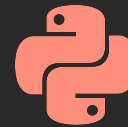


Requests assíncronos com HTTPX



Live de Python #234



1. HTTPX

Conhecendo o básico da biblioteca

2. Um resumo sobre I/O

As questões do eventloop

3. Requests assíncronos

Juntando o problema de I/O com requests

4. iometer

Controlando o event loop



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, Andre Mesquita, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bernardo At, Bruno Almeida, Bruno Barcellos, Bruno Barros, Bruno Batista, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Christiano Moraes, Damianth, Daniel Freitas, Daniel Wojcikowski, 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, Érico Andrei, Everton Silva, Fabiano, 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 Moreira, Gabriel Paiva, Gabriel Simonetto, Gabriel Souza, Geandreson Costa, Gilberto Abrao, Giovanna Teodoro, Giuliano Silva, Guilherme Felitti, Guilherme Gall, Guilherme Silva, Guionardo Furlan, Gustavo Pereira, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Hugo Cosme, Igor Riegel, Italo Silva, Janael Pinheiro, Jhonatan Martins, Joelson Sartori, Jônatas Silva, Jon Cardoso, Jorge Silva, Jose Alves, José Gomes, Joséito Júnior, Jose Mazolini, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leandro Silva, Leo Ivan, 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 Nunes, Mirian Batista, Mlevi Lsantos, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Paulo D., Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Pydocs Pro, Pytonyc, Rafael Lopes, Rafael Romão, Rafael Veloso, 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 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, Vicente Marcal, Vinicius Stein, Vladimir Lemos, Walter Reis, Willian Lopes, Wilson Duarte, Wilson Neto, Xico Silvério, Yuri Fialho, Zeca Figueiredo



Obrigado você



“Meu objetivo hoje não é focar nas questões específicas sobre requisições, como verbos e códigos de resposta!”

(Podemos fazer uma live somente sobre isso!)



Nota 1



“Também não tenho objetivo de me aprofundar em como o asyncio interage com o sistema operacional!”

(Também podemos fazer uma live somente sobre isso no futuro)



Nota 2



De forma pragmática sobre como
estruturar e organizar um código de
forma concorrente para fazer
requisições assíncronas



Sobre o que vamos falar então?



HTTPX

Uma apresentação
resumida



- **HTTPX** é um projeto criado por **Tom Christie**
 - Também criador do uvicorn, mkdocs, starlette, DRF, etc.
- Toma como base a API já consagrada pelo **requests**
- Versão inicial: 0.0.1 - **Março 2015**
- Versão atual: **0.24.0**
- Licença: **BSD**
- Suporta o protocolo HTTP 1.1 e 2
- No que nos importa nessa live. Oferece suporte a:
 - asyncio
 - Trio
 - AnyIO

```
pip install httpx
```



Instalação



Um olá mundo



```
1  import httpx
2
3  response = httpx.get('https://httpbin.org/get')
4  print(response.text)
```

Métodos



```
1  import httpx
2
3  httpx.get('https://httpbin.org/get')
4  httpx.post('https://httpbin.org/post')
5  httpx.patch('https://httpbin.org/patch')
6  httpx.delete('https://httpbin.org/delete')
```

Reponses



```
1  import httpx
2
3  response = httpx.get('https://httpbin.org/get')
4  response.text      # Para texto
5  response.content   # Para binários
6  response.json      # para json
7
8  response.headers
9  response.status_code
10 response.cookies
```

Timeout e redirect



```
1  from httpx import get
2
3  r = httpx.get(
4      'https://httpbin.org/redirect/10',
5      follow_redirects=True,
6      timeout=5.0 # None = Inf
7  )
8
9  r.history
10 r.history[0].text
```



```
1  from httpx import Client
2
3  with httpx.Client() as client:
4      r = client.get('https://httpbin.org/get')
5
6  r.json()
```

Um resumo sobre o
problema!

I/O

Uma API de verdade



Vamos supor que estejamos construindo uma base de conhecimento sobre pokémons. Nisso, precisamos pegar um pokémon e descobrir se ele tem uma evolução.

Tudo que temos é um Nome de um pokemon!

Caso você esteja procurando um exemplo “enterprise”, pense nisso como uma coleta de documentos aninhados.

Com isso em mente



Usemos a PokéAPI.

- [https://pokeapi.co/api/v2/**pokemon**{nome_do_pokemon}](https://pokeapi.co/api/v2/pokemon/{nome_do_pokemon})
 - retorna o ID da espécie
 - .id
- [https://pokeapi.co/api/v2/**pokemon-species**{id-do-pokemon}](https://pokeapi.co/api/v2/pokemon-species/{id-do-pokemon})
 - retorna o id da evolução
 - .evolution_chain.url
- [https://pokeapi.co/api/v2/**evolution-chain**{id-da-evolução}](https://pokeapi.co/api/v2/evolution-chain/{id-da-evolução})
 - retorna para quem o pokémon evolui
 - .chain.evolves_to[0].species.name

```
1  from httpx import Client
2
3  poke = 'charmander'
4
5  with Client(base_url='https://pokeapi.co/api/v2') as client:
6      # request 1
7      response = client.get(f'/pokemon/{poke}')
8      id_ = response.json().get('id')
9
10     # request 2
11     response = client.get(f'/pokemon-species/{id_}')
12     evolution = response.json().get('evolution_chain').get('url')
13
14     # request 3
15     response = client.get(evolution)
16     print( # Sim, tá bem escondido mesmo
17         response
18         .json()
19         .get('chain')
20         .get('evolves_to')[0]
21         .get('species')
22         .get('name')
23     )
24
```

Evoluindo o conceito



Como estamos montando uma base de conhecimento, seria ideal termos uma quantidade interessante de pokémons na nossa base.

Por exemplo:

- Bubasaur
- Charmander
- Squirtle
- Caterpie
- Weedle
- Pikachu

```
1 from httpx import Client
2
3 pokes = ['bulbasaur', 'charmander', 'squirtle', ...]
4
5 def evolution(poke):
6     with Client(base_url='https://pokeapi.co/api/v2') as client:
7         # request 1
8         response = client.get(f'/pokemon/{poke}')
9         id_ = response.json().get('id')
10
11        # request 2
12        response = client.get(f'/pokemon-species/{id_}')
13        evolution = response.json().get('evolution_chain').get('url')
14
15        # request 3
16        response = client.get(evolution)
17        print(
18            response
19            .json()
20            .get('chain')
21            .get('evolves_to')[0]
22            .get('species')
23            .get('name')
24        )
25
26 for poke in pokes:
27     evolution(poke)
```

O problema do I/O



Se adicionarmos uma base grande de pokémons nosso aumento de tempo é exponencial.

- Para cada pokémon 3 requests (0.5s)
- 10 pokémons - 30 requests - 5.0s
- 100 pokémons - 300 requests - 50.0s

Porém, **isso não escala dessa forma**. Pois, é I/O

AsyncIO [exemplo_async.py]

Podemos criar uma fila de eventos e processar eles de forma independente.

```
1  import asyncio
2
3  async def minha_corotina():
4      print("Início da corotina")
5      await asyncio.sleep(1)
6      print("Fim da corotina")
7
8  async def main():
9      tarefa1 = asyncio.create_task(minha_corotina())
10     print("Tarefas pendentes antes de 'await':", asyncio.all_tasks())
11
12     await tarefa1
13     print("Tarefas pendentes após 'await':", asyncio.all_tasks())
14
15  asyncio.run(main())
```

0 loop de eventos



Fila de tarefas



Task



Loop
de
eventos




```
1  import asyncio
2
3  async def minha_corotina():
4      print("Início da corotina")
5      await asyncio.sleep(1)
6      print("Fim da corotina")
7
8  async def main():
9      tarefa1 = asyncio.create_task(minha_corotina())
10     print("Tarefas pendentes antes de 'await':", asyncio.all_tasks())
11
12     await tarefa1
13     print("Tarefas pendentes após 'await':", asyncio.all_tasks())
14
15  asyncio.run(main())
```

0 loop de eventos

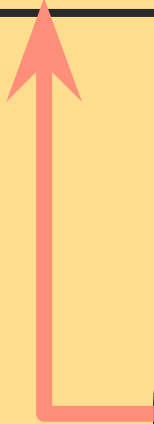


Fila de tarefas

Task

Loop
de
eventos

Executa de forma
sequencial até
encontrar um await

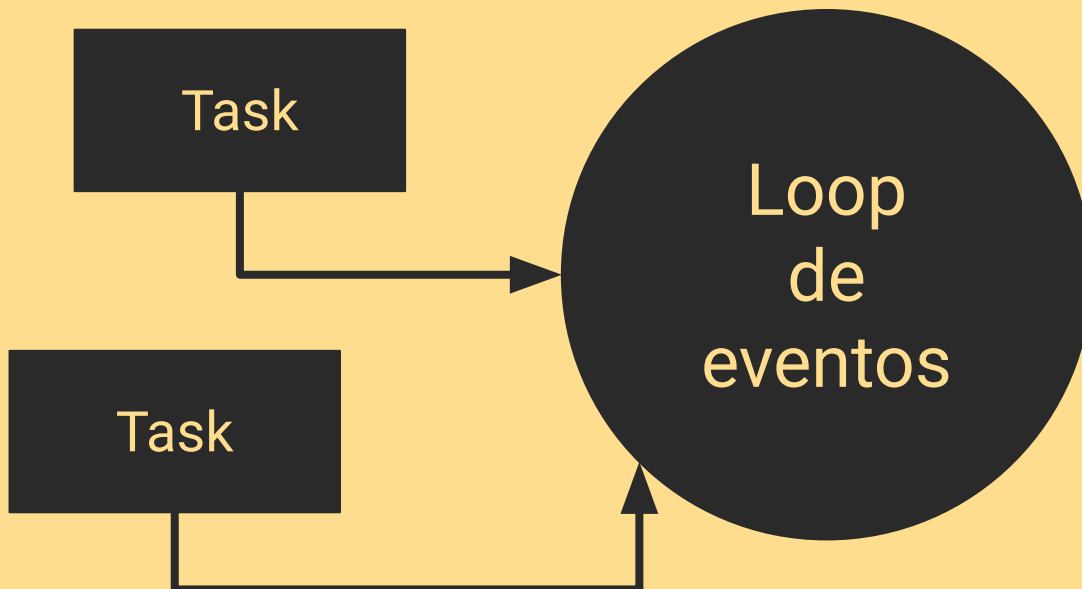


```
1  import asyncio
2
3  async def minha_corotina():
4      print("Início da corotina")
5      await asyncio.sleep(1)
6      print("Fim da corotina")
7
8  async def main():
9      tarefa1 = asyncio.create_task(minha_corotina())
10     tarefa2 = asyncio.create_task(minha_corotina())
11     print("Tarefas pendentes antes de 'await':", asyncio.all_tasks())
12
13     await tarefa1
14     await tarefa2
15     print("Tarefas pendentes após 'await':", asyncio.all_tasks())
16
17  asyncio.run(main())
```

0 loop de eventos



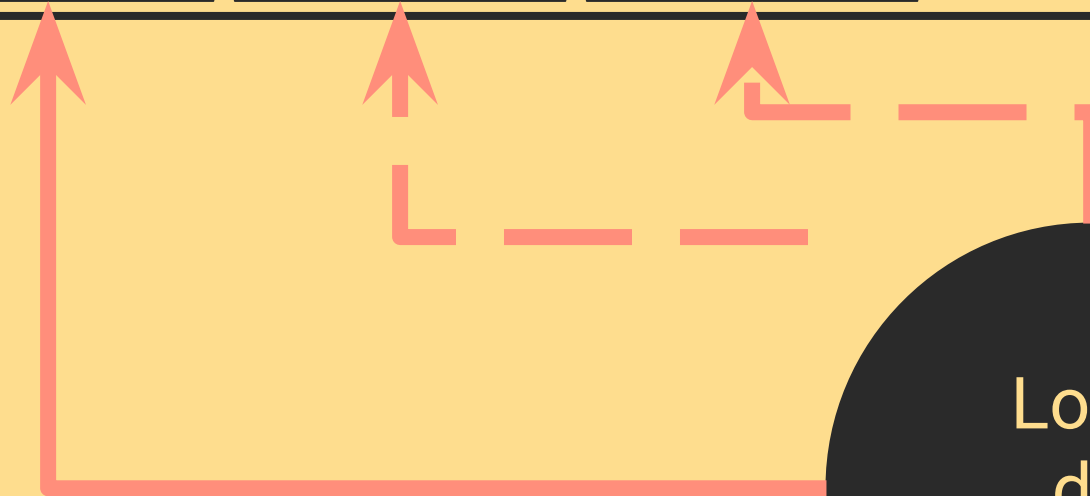
Fila de tarefas



0 loop de eventos

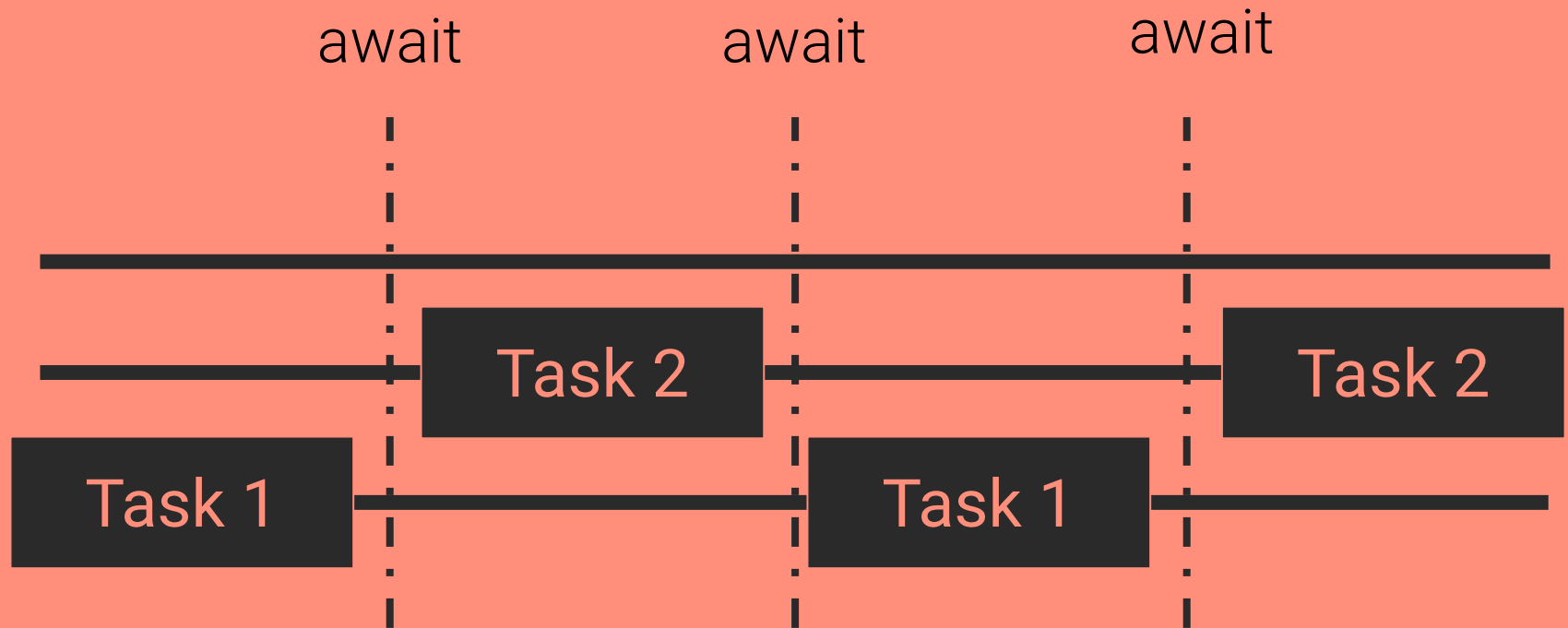


Fila de tarefas



Loop
de
eventos

0 escalonamento



```
1  import asyncio
2
3  async def minha_corotina(nome):
4      print("Início da corotina: ", nome)
5      await asyncio.sleep(1)
6      print("Meio da corotina: ", nome)
7      await asyncio.sleep(1)
8      print("Fim da corotina: ", nome)
9
10 async def main():
11     tarefa1 = asyncio.create_task(minha_corotina(1))
12     tarefa2 = asyncio.create_task(minha_corotina(2))
13     tarefa3 = asyncio.create_task(minha_corotina(3))
14     print("Tarefas pendentes antes de 'await':", asyncio.all_tasks())
15
16     await tarefa1
17     await tarefa2
18     await tarefa3
19     print("Tarefas pendentes após 'await':", asyncio.all_tasks())
20
21 asyncio.run(main())
```

Grupo de tasks



O asyncio tem um mecanismo para agrupar corrotinas e envolvê-las em tasks sem precisar fazer isso de forma imperativa. O **asyncio.gather**

```
1  async def main():
2      tasks = asyncio.gather(
3          *[minha_corotina(n) for n in range(3)]
4      )
5      print("Tarefas pendentes antes de 'await':", asyncio.all_tasks())
6
7      await tasks
8      print("Tarefas pendentes após 'await':", asyncio.all_tasks())
```


Juntando
problemas de I/O +
requests

HTTPX

+

Async

Cliente assíncrono do HTTPX



```
1  import asyncio
2  import httpx
3
4  async def hello_world():
5      async with httpx.AsyncClient() as client:
6          response = await client.get("https://httpbin.org/get")
7          print(response.json())
8
9  asyncio.run(hello_world())
```

Cliente assíncrono do HTTPX



```
1 import asyncio
2 import httpx
3
4 async def hello_world():
5     async with httpx.AsyncClient() as client:
6         response = await client.get("https://httpbin.org/get")
7         print(response.json())
8
9 asyncio.run(hello_world())
```

Abre espaço para escalar!

CODE!



Bora juntar as duas coisas




Testes



Para testar o httpx e mockar seus requests, temos o RESPX

pip install respx

A detailed illustration of a Papilio butterfly, showing its black wings with white veins and red spots on the hindwings, is positioned on the right side of the code block.

```
1 from httpx import Response
2
3
4 def test_default(respx_mock):
5     respx_mock.get("https://foo.bar/").mock(return_value=Response(204))
6     response = ... # Nossa função
7     assert response.status_code == 204
```

Controllando o
eventloop

aio
meter



- **aiometer** foi criado por Florimond Manca
 - Um dos contribuidores do httpx
- Versão inicial: 0.1.0 - Março de **2020**
- Versão atual: **0.4.0**
- Licença: **MIT**
- Suporta:
 - `asyncio`
 - `trio`

Exemplo de uso



```
1  from functools import partial
2  from aiometer import run_all
3
4  async def main():
5      # result = gather(*[evolution(poke) for poke in pokes])
6      result = run_all(
7          [partial(evolution, poke) for poke in pokes],
8          max_at_once=2,
9          max_per_second=2
10     )
11     await result
```


Referências



- **HTTPX**: <https://www.python-httpx.org/>
- **aiometer**: <https://github.com/florimondmanca/aiometer>
- **PokeAPI**: <https://pokeapi.co/>
- **HTTPbin**: httpbin.org/
- **RESPX**: <https://lundberg.github.io/respx/>



picpay.me/dunossauro



apoia.se/livedepython



pix.dunossauro@gmail.com



Ajude o projeto <3

