Novidades Python 3.11

Live de Python # 220

Roteiro

0. Coisas que não vamos falar!

Vão ganhar lives específicas

1. Geralzão

Inclusões interessantes

2. TOMLlib

Biblioteca para lidar com TOML

3. Asyncio

TaskGroups

4. Exceptions

Notas e except*

5. Typing

TypedDict, Self e TypeVarTuple

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriana Cavalcanti, Alexandre Harano, Alexandre Lima, Alexandre Souza, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Andre Azevedo, André Rafael, Aquiles Coutinho, Arnaldo Turque, Aurelio Costa, Bruno Batista, Bruno Freitas, Bruno Guizi, Bruno Oliveira, Bruno Ramos, Caio Nascimento, Carina Pereira, Christiano Morais, Clara Battesini, Daniel Freitas, Daniel Haas, Danilo Segura, David Couto, David Kwast, Delton Porfiro, Dhyeives Rodovalho, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino Aguilar, Diogo Paschoal, Douglas Bastos, Douglas Braga, Douglas Zickuhr, Dutofanim Dutofanim, Eliel Lima, Elton Silva, Emerson Rafael, Eneas Teles, Erick Ritir, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Everton Silva, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Fabricio Araujo, Felipe Rodrigues, Fernanda Prado, Fernando Rozas, Flávio Meira, Flavkaze Flavkaze, Gabriel Barbosa, Gabriel Mizuno, Gabriel Nascimento, Gabriel Simonetto, Geandreson Costa, Guilherme Cabrera, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Gustavo Dettenborn, Gustavo Pereira, Gustavo Suto, Heitor Fernandes, Henrique Junqueira, Hugo Cosme, Igor Taconi, Israel Gomes, Italo Silva, Jair Andrade, Jairo Jesus, Jairo Lenfers, Janael Pinheiro, João Paulo, João Rodrigues, Joelson Sartori, Johnny Tardin, Jonatas Leon, Jônatas Silva, José Gomes, Joseíto Júnior, Jose Mazolini, José Pedro, Juan Gutierrez, Juliana Machado, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leonardo Mello, Leonardo Nazareth, Luancomputacao Roger, Lucas Adorno, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Oliveira, Lucas Simon, Lucas Teixeira, Lucas Valino, Luciano Silva, Luciano Teixeira, Luiz Junior, Luiz Lima, Luiz Paula, Luiz Perciliano, Maicon Pantoja, Maiquel Leonel, Marcelino Pinheiro, Marcelo Matte, Márcio Martignoni, Marcio Moises, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Marina Passos, Mateus Lisboa, Matheus Cortezi, Matheus Silva, Matheus Vian, Mauricio Nunes, Mrreinadogoes Mrreinadogoes, Murilo Andrade, Murilo Cunha, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Patricia Minamizawa, Patrick Felipe, Paulo Braga, Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, P Muniz, Priscila Santos, Rafael Lopes, Rafael Rodrigues, Rafael Romão, Ramayana Menezes, Regis Tomkiel, Renato Veirich, Ricardo Silva, Riverfount Riverfount, Robson Maciel, Rodrigo Alves, Rodrigo Cardoso, Rodrigo Freire, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Rogério Sousa, Ronaldo Silva, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Thalles Rosa, Thiago Araujo, Thiago Bueno, Thiago Curvelo, Thiago Moraes, Thiago Oliveira, Thiago Salgado, Thiago Souza, Tiago Minuzzi, Tony Dias, Valcilon Silva, Valdir Tegon, Victor Wildner, Vinícius Bastos, Vinicius Stein, Vitor Luz, Vladimir Lemos, Walter Reis, Wellington Abreu, Wesley Mendes, William Alves, Willian Lopes, Wilson Neto, Wilson Rocha, Xico Silvério, Yury Barros

Obrigado você

Coisas que não vamos falar!

0

Não iremos abordar nessa live

- **O ganho de performance da versão 3.10 para 3.11**
- Mudanças internas no bytecode
- Novidades do Unittest
- SafePaths "-P"
- typing.Dataclass Trasforms
- typing.LiteralString
- Adicições no datetime

Teremos uma live especial para avaliar performance

https://docs.python.org/3.11/whatsnew/3.11.html#faster-cpython

Melhorias gerais

O TODO

Desempacotamento de tuplas

Pathlib

O glob da pathlib agora consegue distinguir diretórios:

```
1   from pathlib import Path
2
3   p = Path('.')
4
5   p.glob('*/')  # Acaba em /, volta só os diretórios
```

Functools

Singledispatch agora suporta união de tipos

```python
from functools import singledispatch

@singledispatch
def generic(arg):
    return 'generic'

@generic.register
def _(arg: int | str):
    return f'{arg=} é int ou str'

@generic.register
def _(arg: float | bool):
    return f'{arg=} é float ou bool'
```

contexlib

Gerenciador de contexto para mudar o path de execução

```
1  from contextlib import chdir
2  from os import getcwd
3
4  print(getcwd())
5
6  with chdir('path'):
7      print(getcwd())
8
9  print(getcwd())
```

Melhoria das mensagens de erro

Agora o python consegue dizer exatamente em qual expressão o erro foi levantado

```
1   def b(): assert False
2   def a(): return b()
3   def xpto(): return a()
4
5   xpto()
```

```python
def b(): assert False
def a(): return b()
def xpto(): return a()

xpto()
```

```
py
Traceback (most recent call last):
  File "/home/dunossauro/live_3_11/mensagens_de_erro.py", line 5, in <module>
    xpto()

  File "/home/dunossauro/live_3_11/mensagens_de_erro.py", line 3, in xpto
    def xpto(): return a()
                       ^^^
  File "/home/dunossauro/live_3_11/mensagens_de_erro.py", line 2, in a
    def a(): return b()
                   ^^^
  File "/home/dunossauro/live_3_11/mensagens_de_erro.py", line 1, in b
    def b(): assert False
             ^^^^^^^^^^^^
AssertionError
```

Chegou quem faltava

TOML LIB

TOML

Tom's **O**bvious **M**inimal **L**anguage, se descreve como "Um formato de arquivo de configuração para humanos".

TOML é uma linguagem para arquivos de configuração. Assim como usamos o .ini, .yaml e em alguns casos o .json

Porém, de forma simplificada. A ideia de trazer isso nativamente é por conta no novo arquivo de configuração oficial do python o **pyproject.toml**

Um básico overview

```toml
# exemplo.toml, isso é um comentário!
chave1 = "valor"     # str
chave2 = 1           # int
chave3 = 1.0         # float
chave4 = true        # bool
chave5 = [1, 2, 3]   # array
```

https://toml.io/en/

Tabelas

```toml
# exemplo2.toml
name = { first = "Tom", last = "Preston-Werner" }

[database]
host = 'postgresql://db'
port = 5432

[database.dev]
db_name = 'xxxxxxxxxx'

[database.prod]
db_name = 'xxxxxxxxxx'
```

TOML lib

Foi incluída na biblioteca padrão um parser para toml, a tomllib. A tomllib conta somente com duas funções:

- **loads**: Carreca uma string no formato toml
- **load**: Abre um arquivo em formato toml

Não existe uma maneira de escrever toml ainda. A documentação recomenda a biblioteca tomkit para isso.

O uso

```python
from tomllib import load

with open('exemplo.toml', 'rb') as f:
    print(load(f))
```

O uso

```python
from tomllib import loads

toml = """
chave = "valor"
"""

print(loads(toml))
```

Asyncio

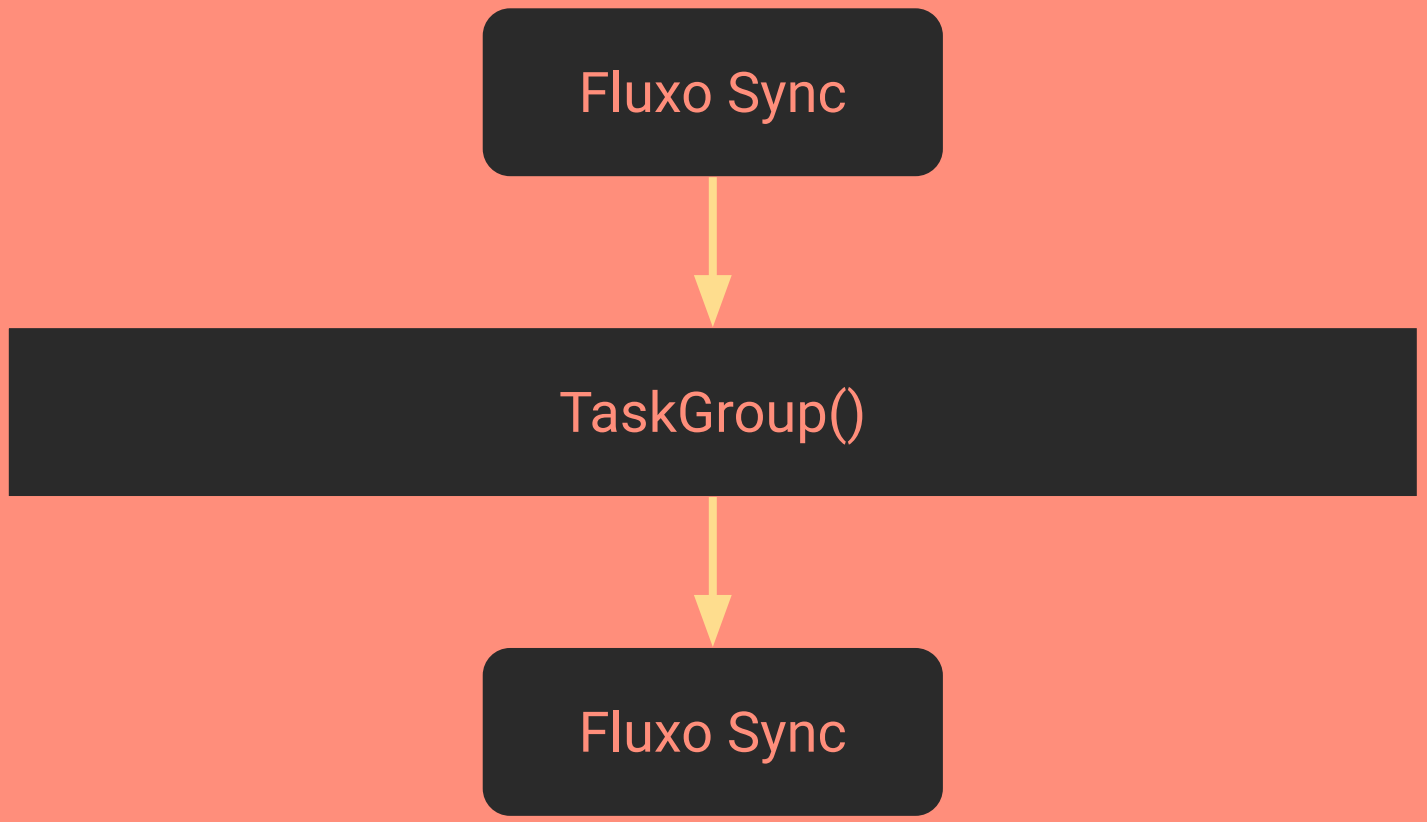
TaskGroups

asyncio.TaskGroup()

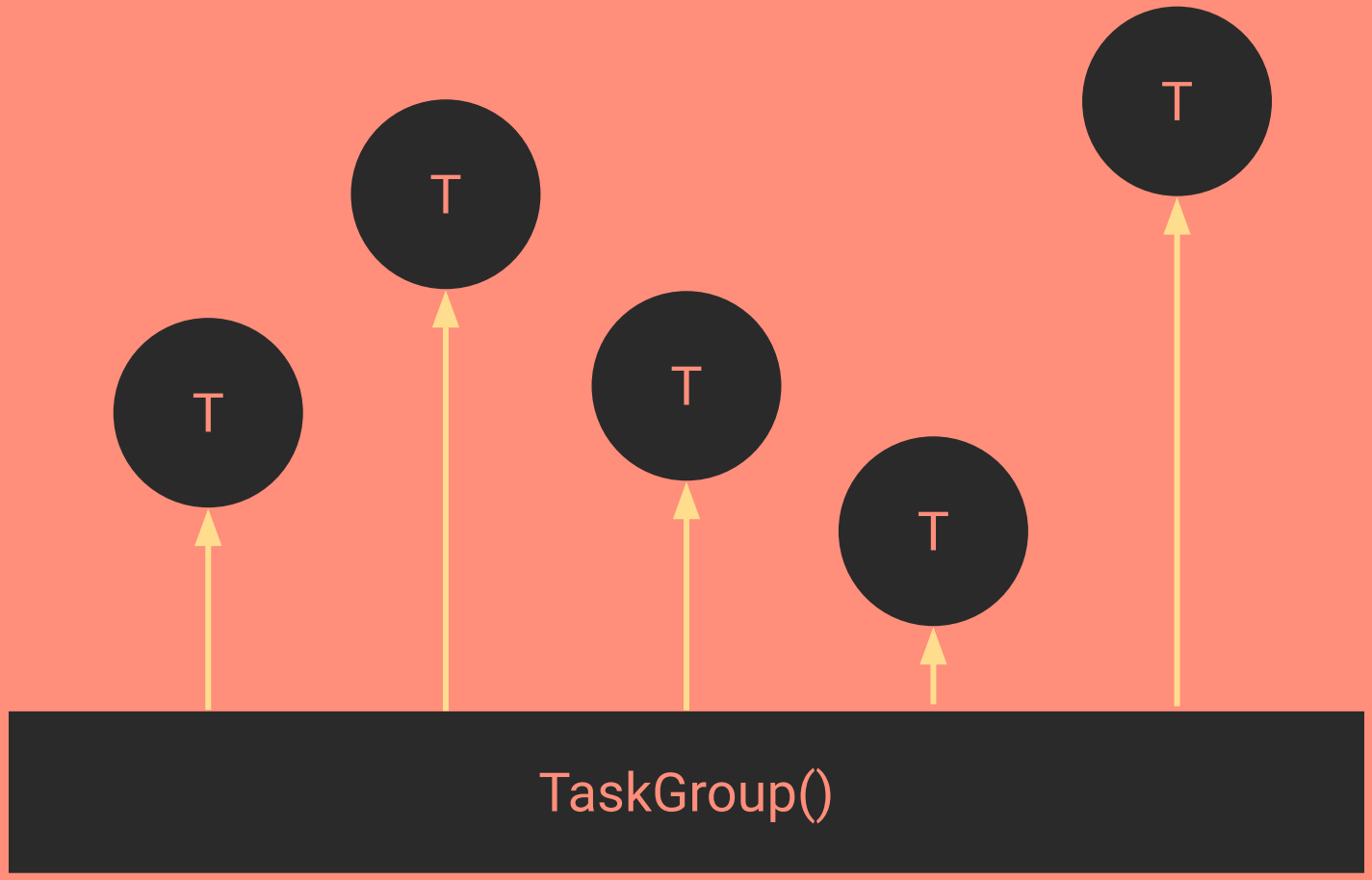
- Forma de executar **corrotinas** em pool.
  - Similar ao ThreadPoolExecutor/ProcessPoolExecutor de futures
  - Similar ao Pool do multiprocess
- Um substituto para o asyncio.gather()

```
1  async def main():
2      async with TaskGroup() as tg:
3          task = tg.create_task(astr('Teste TaskGroup!'))
4          result = await task
5
6      return result
```

Um grupo de tasks



Um grupo de tasks



O fluxo

- O TaskGroups vai sair do gerenciador de contexto somente quando todas as tasks foram finalizadas. Evitando awaits
- Caso uma task falhe ele para todas as tasks
- **A propagação de erros é feita por tasks**

A ideia do taskGroups é trazida do **Trio**, uma alternativa ao asyncio

Exceções

Algumas coisas novas!

ExceptionGroup

A partir da versão 3.11, agora exceções podem ser agrupadas e tratadas de forma individual. Para isso existe uma nova adição sintática na linguagem. o **except***

```
1   eg = ExceptionGroup(
2       'EG', [
3           TypeError(),
4           KeyError(),
5           ExceptionGroup('NEG', [
6               ZeroDivisionError()
7           ])
8       ]
9   )
```

A mensagem de erro de grupo

```
>>> raise eg
  + Exception Group Traceback (most recent call last):
  |   File "<stdin>", line 1, in <module>
  |   File "exception_group.py", line 132, in <module>
  |     raise eg
  | ExceptionGroup: EG (3 sub-exceptions)
  +-+---------------- 1 ----------------
    | TypeError
    +---------------- 2 ----------------
    | KeyError
    +---------------- 3 ----------------
    | ExceptionGroup: NEG (1 sub-exception)
    +-+---------------- 1 ----------------
      | ZeroDivisionError
      +------------------------------------
```

except*

Todas as exceções do grupo podem ser tratadas de maneira individual. E de forma sequencial. Para isso, a keywork except ganhou um *****

```
1  try:
2      raise eg
3  except* TypeError as te:
4      print('TypeError', te)
5  except* KeyError as ke:
6      print('KeyError', ke)
7  except* ZeroDivisionError as ze:
8      print('ZeroDivisionError', ze)
```

Sobre os grupos

A ideia é que ExceptionsGroups sejam lançadas pela linguagem. A primeira funcionalidade do python a contar com essa novidade é o **TaskGroup**. Assim, todas as excpetions levantadas por N tasks podem ser tradas de uma vez.

Sobre os grupos

A ideia é que ExceptionsGroups sejam lançadas pela linguagem. A primeira funcionalidade do python a contar com essa novidade é o **TaskGroup**. Assim, todas as excpetions levantadas por N tasks podem ser tradas de uma vez.

CODAR UM POUCO AQUI, CASO EU ESQUEÇA

Notas para exceções

Agora existe uma nova interface entre pessoas que desenvolvem bibliotecas e pessoas que tem que ler erros. Notas para exceções.

```
1  try:
2      1/0
3  except Exception as e:
4      e.add_note('Nota1')
5      e.add_note('Nota2')
6      e.add_note('Nota3')
7      raise
```

Só algumas coisas,
não todas!

Typing

Self

```python
from typing import Self

class Iterable:
    def __iter__(self) -> Self:
        return self
```

TypedDict

```python
from typing import typedDict, Required, NotRequired

class Filme(typedDict):
    titulo: str
    ano: NotRequired[int]


class Filme(typedDict, total=False):
    titulo: Required[str]
    ano: int
```

Variadic Generics

Na PEP-484 foi inserida a sintaxe para criação de variáveis de tipos genéricas. Porém, não havia uma forma de criar variações nas estruturas.

Um bom caso de exemplo é o empacotamento de argumentos não nomeados

```
1   def func(*args: int):
2       return 1, *args
```

TypeVarTuple

```python
from typing import TypeVarTuple

Ts = TypeVarTuple('Ts')


def func(
    *args: *tuple[*Ts, bool]
) -> tuple[int, *Ts, bool]:
    return 1, *args


reveal_type(func(1, 2, 3, 4, True))
```

Perguntas?

Por hoje é só!

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3