Novidades Python 3.12

Live de Python #247

Roteiro

1. Melhorias gerais

Não podemos ver, mas sentir!

2. Bibliotecas

Adições em libs já existentes

3. CLIs

Ferramentas de linha de comando

4. Gramática e Sintaxe

Modificações "na" linguagem

5. Outros pontos

Menções honrosas

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alemao, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alfredo Braga, Alisson Souza, Alysson Oliveira, Andre Azevedo, Andre Mesquita, Andre Paula, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Ayr-ton, Bernardo At, Bruno Almeida, Bruno Barcellos, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Carlos Ramos, Christian Semke, Claudemir Firmino, Cristian Firmino, Damianth, Daniel Freitas, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Goivinho, Davi Souza, Dead Milkman, Delton Porfiro, Denis Bernardo, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino, Diogo Paschoal, Edgar, Edilson Ribeiro, Eduardo Silveira, Eduardo Tolmasquim, Emerson Rafael, Erick Andrade, Érico Andrei, Everton Silva, Fabiano, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Correa, Fábio Thomaz, Fabricio Biazzotto, Fabricio Patrocinio, Felipe Augusto, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Firehouse, Flávio Meira, Francisco Neto, Francisco Silvério, Gabriel Espindola, Gabriel Mizuno, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Geandreson Costa, Geizielder, Giovanna Teodoro, Giuliano Silva, Guilherme Felitti, Guilherme Gall, Guilherme Piccioni, Guilherme Silva, Guionardo Furlan, Gustavo Suto, Haelmo, Haelmo Almeida, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Henrique Andrade, Higor Monteiro, Italo Silva, Janael Pinheiro, Jean Victor, Jefferson Antunes, Joelson Sartori, Jonatas Leon, Jônatas Silva, Jorge Silva, José Gomes, Joseíto Júnior, Jose Mazolini, Josir Gomes, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Júlio Sanchez, Julio Silva, Kaio Peixoto, Kálita Lima, Leandro Silva, Leandro Vieira, Leonardo Mello, Leonardo Nazareth, Lucas Carderelli, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Pavelski, Lucas Schneider, Lucas Simon, Luciano Filho, Luciano Ratamero, Luciano Teixeira, Luis Eduardo, Luiz Duarte, Luiz Lima, Luiz Paula, Mackilem Laan, Marcelo Araujo, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Marina Passos, Mateus Lisboa, Mateus Ribeiro, Mateus Silva, Matheus Silva, Matheus Vian, Mírian Batista, Mlevi Lsantos, Murilo Carvalho, Murilo Viana, Natan Cervinski, Nathan Branco, Ocimar Zolin, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Pedro Henrique, Pedro Pereira, Peterson Santos, Pytonyc, Rafael Faccio, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Regis Santos, Renato José, Renato Oliveira, Renê Barbosa, Rene Bastos, Rene Pessoto, Ricardo Silva, Riverfount, Rjribeiro, Robson Maciel, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Vaccari, Rodrigo Vieira, Rui Jr, Samanta Cicilia, Samuel Santos, Selmison Miranda, Téo Calvo, Thiago Araujo, Thiago Borges, Thiago Curvelo, Thiago Souza, Tony Dias, Tyrone Damasceno, Valdir, Valdir Tegon, Vinicius Stein, Walter Reis, Willian Lopes, Wilson Duarte, Yros Aguiar, Zeca Figueiredo

Obrigado você

Não podemos ver,
só sentir!

Melhorias Gerais

Mensagens de erro!

As melhorias em mensagens de erro vem acontecendo desde a versão **3.10**. Na versão 3.10 ganhamos funcionalidades para identificar a posição do erro na expressão para:

- SintaxError
- IndentationError
- AttributeError
- NameError

Agora na **3.12** além de melhoria nos erros já citados, também ganhamos o **importError** com mensagens melhores e algumas sugestões de nomes na biblioteca padrão.

Sugestões da biblioteca padrão

```
# 3.11
Traceback (most recent call last):
  File "/home/dunossauro/live_novidades/erros.py", line 1, in <module>
    @dataclasses.dataclass
     ^^^^^^^^^^^
NameError: name 'dataclasses' is not defined

# 3.12
Traceback (most recent call last):
  File "/home/dunossauro/live_novidades/erros.py", line 1, in <module>
    @dataclasses.dataclass
     ^^^^^^^^^^^
NameError: name 'dataclasses' is not defined. Did you forget to import 'dataclasses'?
```

Sugestões de coisas contidas nas bibliotecas

```
# 3.11
Traceback (most recent call last):
  File "/home/dunossauro/live_novidades/erros.py", line 5, in <module>
    from itertools import islici
ImportError: cannot import name 'islici' from 'itertools' (unknown location)

# 3.12
Traceback (most recent call last):
  File "/home/dunossauro/live_novidades/erros.py", line 5, in <module>
    from itertools import islici
ImportError: cannot import name 'islici' from 'itertools' (unknown location).
    Did you mean: 'islice'?
```

Compreensões

Compreensões **ganharam %11 de performance**. Agora elas rodam no mesmo frame que o código que a chamou, anteriormente eram como chamadas de funções. Com isso elas tem acesso ao escopo e as variáveis desse escopo:

```
1   a = 7
2   print([locals() for x in [1]])
```

Execução

```
# 3.11
[{'.0': <tuple_iterator object at 0x7f6548dba470>, 'x': 1}]

# 3.12
[{'__name__': '__main__', '__doc__': None, '__package__': None,
  '__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x7f6f613fc710>,
  '__spec__': None,
  '__annotations__': {},
  '__builtins__': <module 'builtins' (built-in)>,
  '__file__': '/home/dunossauro/live_novidades/erros.py',
  '__cached__': None, 'a': 7, 'x': 1}]
```

Inclusão de funcionalidades nas bibliotecas padrões

Bibliotecas

pdb

Agora podemos definir variáveis globais e temporárias que não interferem no frame nem na stack:

```
>>> import pdb
>>> pdb.set_trace()
>>> $a = 10
>>> $a
10
```

Itertools

Itertools agora conta com uma nova função chamada **batched**, ela corta um iterável em pedaços menores.

```python
from itertools import batched

batched([1, 2, 3, 4, 5], 2)
# (1, 2), (3, 4), (5,)
```

Math

Agora podemos calcular o somatório do produto entre dois iteráveis:

```
from math import sumprod

sumprod([1, 2, 3], [1, 2, 3])  # 14
# (1 * 1) + (2 * 2) + (3 * 3)
```

Tempfile

Melhorias gerais na manutenção de temporários e também nos caminhos temporários:

- **NamedTemporaryFile**: Agora pode manter o arquivo aberto
- **mkdtemp**: Sempre gera caminhos absolutos, mesmo se o passado seja relativo

NamedTemporaryFile

Agora conta com um parâmetro **delete_on_close**. Agora o arquivo pode ser mantido após a finalização do gerenciador de contexto:

```
1   with TemporaryFile(delete_on_close=False) as fp:
2       fp.write(b'Hello world!')
3       fp.close()
```

mkdtemp

Agora sempre provê caminhos absolutos mesmo que **dir** seja relativo:

```
>>> from tempfile import mkdtemp

# 3.11
>>> mkdtemp(dir='.')
'./tmptm_w0uj4'

# 3.12
>>> mkdtemp(dir='.')
'/home/dunossauro/tmpx8zd6fln'
```

Typing

Diversas melhorias foram implementadas na biblioteca:

- **Checagem de protocolos em runtime**: Agora protocolos são dinâmicos
- **Anotação para sobrescrita de método**: Avisar o checador que o método está sendo sobreescrito
- **Anotação de tipos para kwargs**: Agora você pode juntar TypedDict e **kwargs

Checagem de protocolos em runtime

Agora protocolos com **runtime_checkable** podem ser checador dinamicamente por **isinstance**:

```
>>> from typing import Protocol, runtime_checkable

>>> @runtime_checkable
... class HasX(Protocol):
...     x = 1

>>> class Foo: ...
>>> f = Foo()

>>> isinstance(f, HasX) # False
>>> f.x = 1
>>> isinstance(f, HasX) # True
>>> HasX.y = 2
>>> isinstance(f, HasX) # True
```

Anotação para sobrescrita de método

```python
from typing import override

class BaseForm:
    def area(self): ...

class Cicle(BaseForm):
    @override
    def area(self): ...

class Error(BaseForm):
    @override
    def teste(self): ...
```

Anotação de tipos para kwargs

Agora você pode anotar tipos para desempacotamento de argumentos nomeados usando **TypedDict** e **Unpack**:

```python
from typing import TypedDict, Unpack

class Movie(TypedDict):
    name: str
    year: int

def foo(**kwargs: Unpack[Movie]): ...
```

CLIs

Ferramentas de linha de comando

SQLite3

Agora o shell do sqlite pode ser chamado via CLI do python:

```
python -m sqlite database.db
```

UUIDs

Agora podemos gerar **UUID** direto pelo **CLI**.

Por padrão é usado uuid4, os uuid 2 e 5 precisam de name e namespace:

```
python -m uuid
df9ff1f0-aa27-4c62-bb70-84bcc4f80aee
```

UUIDs

```
python -m uuid -u uuid4
11cb7757-fb95-4af3-930e-f705d96974f5

python -m uuid -u uuid1
93094c74-63a9-11ee-8325-48ad9a89a599

python -m uuid -u uuid3 -n @dns -N dunossauro.com
28d8df77-5265-3bd1-8a41-6faa311be89b
```

Unittest

Agora o unittest tem como medir a duração individual de testes. Isso pode ser usado com a flag **--durations**.

Durations precisa receber um parâmetro que é a quantidade de testes que ele vai elencar dos mais lentos.

```
python -m unittest exemplo.py --durations=3
```

Unittest

No exemplo ele elencará os 3 testes mais lentos. O resultado:

```
python -m unittest exemplo.py --durations=3
.....
Slowest test durations
----------------------------------------------------------------------
5.000s     test_long_long (exemplo_unittest.TestExample.test_long_long)
3.000s     test_long (exemplo_unittest.TestExample.test_long)
2.000s     test_mid_long (exemplo_unittest.TestExample.test_mid_long)
----------------------------------------------------------------------
Ran 5 tests in 11.101s

OK
```

Unittest

Podemos ver a duração de todos os testes usando **--durations=0** e a flag de verbose **-v**:

```
$ python -m unittest exemplo_unittest.py --durations=0 -v

test_long (exemplo_unittest.TestExample.test_long) ... ok
test_long_long (exemplo_unittest.TestExample.test_long_long) ... ok
test_mid (exemplo_unittest.TestExample.test_mid) ... ok
test_mid_long (exemplo_unittest.TestExample.test_mid_long) ... ok
test_short (exemplo_unittest.TestExample.test_short) ... ok

Slowest test durations
----------------------------------------------------------------------
5.000s     test_long_long (exemplo_unittest.TestExample.test_long_long)
3.000s     test_long (exemplo_unittest.TestExample.test_long)
2.000s     test_mid_long (exemplo_unittest.TestExample.test_mid_long)
1.000s     test_mid (exemplo_unittest.TestExample.test_mid)
0.100s     test_short (exemplo_unittest.TestExample.test_short)
----------------------------------------------------------------------
```

Novas adições na forma de programar!

Sintaxe

A instrução type

Quando as anotações foram inseridas na linguagem, caso tivéssemos um tipo que seria usado mais de uma vez, poderíamos criar um **alias** para ele, como uma variável "tradicional".

A única premissa era que o nome deveria iniciar com letra maiúscula (VER: PEP-484, PEP-8)

```python
MyDict = dict[str, str | float | list]

def xpto(d: MyDict): ...
```

A instrução type

Na versão 3.10 do python foi um introduzida uma anotação específica e explícita para alias de tipos PEP-613), o tipo **TypeAlias**, reduzindo a ambiguidade entre uma definição de variável e uma apelido de tipo:

```python
from typing import TypeAlias

MyDict: TypeAlias = dict[str, str | float | list]

def xpto(d: MyDict): ...
```

A instrução type

Agora na versão 3.12 do Python ganhamos a instrução **type** com a (PEP-695), para criação de apelidos de tipo:

```python
type MyDict = dict[str, str | float | list]

def xpto(d: MyDict): ...
```

A nova sintaxe para generics

Agora as variáveis de tipos agora podem ser definidas dentro do próprio namespace de declaração de forma lazy. Simplificando a utilização dos generics para anotações.

Sintaxe antiga:

```
from typing import TypeVar, Generic

T = TypeVar('T')

def generic(x: T, y: T) -> T: ...

S = TypeVar('S', bound=complex)

class C(Generic[S]): ...

C[int]()
```

A nova sintaxe para generics

Sintaxe nova:

```
def generic[T](x: T, y: T) -> T: ...

class C[S: complex]: ...

C[float]()
```

F-strings

Gramatica

F-Strings

As f-strings foram devidamente formalizadas na PEP-701, em contrapartida, a algumas restrições formuladas na PEP-498 em 2015 (a PEP da criação da funcionalidade).

Em 2016 ouve uma tentativa de formalizar as f-strings, mas sem sucesso, foi adiada pelo steering council. Pelo alto custo de manutenção que adicionaria ao analisador sintático.

```
f'Aspas aninhadas: {'aspas'}'  # SyntaxError

f'Barra invertida: {\'barras\'}'  # SyntaxError
```

F-strings

Após a formalização das f-strings, agora podemos:

- Aninhar f-strings (f-string dentro de f-string)
- Usar barra invertida (\)
- Usar as mesmas aspas (f'{"}')
- Fazer comentários em dentro do "meio": {}

F-strings

```
f'Aspas aninhadas: {'aspas'}'

f'Barra invertida: {\'barras\'}'

f'''Comentário: {
   'quebra'  # Comentário
}'''

f'Aninhamento: {f"{'sim'}"}'

f"{f"{f"{f"{f"{f"{1+1}"}"}"}"}"}"
```

Menções honrosas

Outros pontos

Random

Agora temos um valor randômico binomial para distribuições discretas. Retorna o número de sucessos para **n** tentativas independentes com a probabilidade de sucesso sendo **p**:

```
from random import binomialvariate

binomialvariate(n, p)
```

Sys

Monitoração de baixo impacto com **sys.monitoring**.

Mais informações na **PEP-669**

PS* Isso renderia uma live sobre monitoramento e debug!

Statistics

Agora **statistics.correlation**, usado para obter o coeficiente de correlação de Pearson, agora conta com o parâmetro **ranked**. Para obter o coeficiente de correlação de postos de Spearman.

```
>>> from statistics import correlation
>>> qi = [106, 86, 100, 101, 99, 103, 97, 113, 112, 110]
>>> horas_de_tv = [7, 0, 27, 50, 28, 29, 20, 12, 6, 17]

>>> correlation(qi, horas_de_tv, method='ranked')
-0.17575757575757575

# método antigo, linear
>>> correlation(qi, horas_de_tv, method='linear')
-0.037601473846875934
```

Subinterpretadores

A **API C do python** agora permite chamar N interpretadores dentro de outros interpretadores. Uma forma de burlar o **GIL**. Será somente um GIL por interpretador.

Teremos uma live focada só nisso com o **@JSBueno** <3

Referências

- **Notas da release**: https://www.python.org/downloads/release/python-3120/
- **O que há de novo?**: https://docs.python.org/uk/3/whatsnew/3.12.html
- **PEP-692**: https://peps.python.org/pep-0692/
- **PEP-693**: https://peps.python.org/pep-0693/
- **PEP-695**: https://peps.python.org/pep-0695/
- **PEP-698**: https://peps.python.org/pep-0698/
- **PEP-701**: https://peps.python.org/pep-0701/
- **PEP-709**: https://peps.python.org/pep-0709/

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3