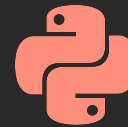




# Novidades 3.13

Live de Python # 275



## 1. Cores

Novo shell interativo, melhoria nos erros e doctests

## 2. Typing

TypedDict, Typels, Default para genéricos

## 3. Internals

Locals, atributos estáticos, Docstrings

## 4. Bibliotecas

Copy, Warnings

## 5. Experimentos

JIT e remoção do GIL

## 6. Outras novidades

Mobile, Documentação, ...



[apoia.se/livedepython](https://apoia.se/livedepython)



[pix.dunossauro@gmail.com](mailto:pix.dunossauro@gmail.com)



[patreon.com/dunossauro](https://patreon.com/dunossauro)



Ajude o projeto <3



Ademar Peixoto, Adriana Cavalcanti, Adriano Casimiro, Alexandre Girardello, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alfredo Braga, Allan Kleitson, Alysson Oliveira, Andre Azevedo, Andre Makoski, Andre Paula, Apc 16, Aslay Clevisson, Aurelio Costa, Belisa Arnhold, Bernarducs, Breno Bruno, Brisa Nascimento, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Lira, Bruno Lopes, Bruno Ramos, Bruno Santos, Caio Nascimento, Carlos Gonçalves, Carlos Ramos, Cecilia Oliveira, Celio Araujo, Christian Fischer, Claudio 360, Cleiton Fonseca, Daniel Aguiar, Daniel Bianchi, Daniel Brito, Daniel Real, Daniel Souza, Daniel Wojcickoski, Danilo Boas, Danilo Silva, David Couto, David Kwast, Denis Bernardo, Dgeison, Diego Guimarães, Dino, Edgar, Elton Guilherme, Emerson Rafael, Érico Andrei, Everton Silva, Fabio Barros, Fábio Barros, Fabio Faria, Fabiokleis, Fabricio Biazotto, Felipe Augusto, Felipe Corrêa, Fernanda Prado, Filipe Oliveira, Francisco Aclima, Frederico Damian, Fulvio Murenu, Gabriel Lira, Gabriel Mizuno, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Geilton Cruz, Geisler Dias, Giovanna Teodoro, Giuliano Silva, Guilherme Felitti, Guilherme Ostrock, Guilherme Piccioni, Guilherme Silva, Gustavo Suto, Haelmo Almeida, Harold Gautschi, Heitor Fernandes, Helton, Helvio Rezende, Henri Alves, Henrique Andrade, Henrique Machado, Henriquesebastiao, Herian Cavalcante, Hiago Couto, Igor Cruz, Igor Taconi, Ivan Santiago, Jairo Lenfers, Janael Pinheiro, Jean Victor, Jeferson Vitalino, Jefferson Antunes, Jhonata Medeiros, Jlx, Joao Rocha, Jonas Araujo, Jonatas Leon, Jônatas Silva, Jorge Silva, Jornada Filho, Jose Barroso, Jose Edmario, Joseíto Júnior, Jose Mazolini, José Predo), Jose Terra, Josir Gomes, Jrborba, Juan Felipe, Juliana Machado, Julio Batista-silva, Julio Franco, Júlio Sanchez, Kael, Kaio Peixoto, Leandro O., Leandro Pina, Leandro Vieira, Leonan Ferreira, Leonardo Alves, Leonardo Mello, Leonardo Nazareth, Letícia, Logan Merazzi, Lucas Carderelli, Lucas Castro, Lucas Lattari, Lucas Mello, Lucas Mendes, Lucas Moura, Lucas Nascimento, Lucas Schneider, Lucas Simon, Luciano Ratamero, Luciano Teixeira, Luiz Carlos, Luiz Duarte, Luizinhoo, Luiz Martins, Luiz Paula, Luiz Perciliano, Mackilem Laan, Marcelo Araujo, Marcelo Fonseca, Marcelo Grimberg, Marcio Freitas, Marcio Junior, Marcos Almeida, Marcos Gomes, Marcos Oliveira, Marcos Rodrigues, Marina Passos, Marlon Rocha, Mateusamorim96, Mateus Lisboa, Matheus Vian, Mírian Batista, Murilo Carvalho, Murilo Viana, Ocimar Zolin, Otávio Carneiro, Patrick Felipe, Pedro Henrique, Peterson Santos, Philipe Vasconcellos, Phmmdev, Prof Santana, Pydocs Pro, Pytonyc, Rafael Araújo, Rafael Costa, Rafael Faccio, Rafael Paranhos, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Renan, Renan Sebastião, Renã Pedroso, Renato José, Rene Pessoto, Renne Rocha, Ricardo Silva, Ricardo Viana, Richard Costa, Rinaldo Magalhaes, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Rui Jr, Samanta Cicilia, Santhiago Cristiano, Selmison Miranda, Sergio Nascimento, Shinodinho, Téó Calvo, Tharles Andrade, Thedarkwithin, Thiago Araujo, Thiago Borges, Thiago Lucca, Thiago Paiva, Tiago, Tiago Emanuel, Tiago Henrique, Tomás Tamantini, Tony Dias, Tyrone Damasceno, Valdir, Vinicius Silva, Vinicius Souza, Vinicius Stein, Vitor Silva, Vladimir Lemos, Wagner Gabriel, Williamslews, Willian Lopes, Zeca Figueiredo



Obrigado você



Essa é uma seleção das coisas que eu considero mais interessantes nessa versão.

<https://docs.python.org/pt-br/3/whatsnew/3.13.html>



Não dá pra cobrir tudo



# Cores

0 novo shell, erros e  
doctest

# O novo shell interativo



No novo shell conta com novos recursos interessantes como:

- Suporte a cores (não é highlighting)
- Edição multilinha (setinha pra cima)
- Atalhos
  - F1: Ajuda
  - F2: Histórico
  - F3: Colar multilinha

# O novo shell interativo



No novo shell conta com novos recursos interessantes como:

- Suporte a cores (não é highlighting)
- Edição multilinha (setinha pra cima)
- Atalhos
  - F1: Ajuda
  - F2: Histórico
  - F3: Colar multilinha

**DEMO**



# Melhoria nas mensagens de erro



Como uma constante vindo desde a versão 3.10, na 3.13 3 melhorias nos sistemas de erro:

1. O traceback agora é exibido com cores, facilitando a leitura
2. Os nomes dos parâmetros agora são sugeridos
3. Erros para quando os arquivos tem o mesmo nome de uma biblioteca

# Melhoria nas mensagens de erro



Como uma constante vindo desde a versão 3.10, na 3.13 3 melhorias nos sistemas de erro:

1. **O traceback agora é exibido com cores, facilitando a leitura**
2. **Os nomes dos parâmetros agora são sugeridos**
3. Erros para quando os arquivos tem o mesmo nome de uma biblioteca

```
dunossauro@melville:~/live_313
>>> def xpto(batata):
...     return batata
...
>>> xpto(batat=5)
Traceback (most recent call last):
  File "<python-input-5>", line 1, in <module>
    xpto(batat=5)
    ~~~~^~~~~~
TypeError: xpto() got an unexpected keyword argument 'batat'. Did you mean 'batata'?
>>> 
```

# Melhoria nas mensagens de erro



Como uma constante vindo desde a versão 3.10, na 3.13 3 melhorias nos sistemas de erro:

1. O traceback agora é exibido com cores, facilitando a leitura
2. Os nomes dos parâmetros agora são sugeridos (Demo)
3. **Erros para quando os arquivos tem o mesmo nome de uma biblioteca**

**DEMO**

# Doctest



Agora os doctests também exibem mensagens coloridas!

```
dunossauro@melville:~/live_313

py-3.13.0 melville in ~/live_313
o → python -m doctest doctest_color.py
*****
File "/home/dunossauro/live_313/doctest_color.py", line 2, in doctest_color
Failed example:
    1 + 1
Expected:
    3
Got:
    2
*****
1 item had failures:
  1 of 1 in doctest_color
***Test Failed*** 1 failure.

py-3.13.0 melville in ~/live_313
o →
```

# Como desativar as cores?



Em alguns contextos contar com as cores é ótimo, em debug, no aprendizado. Mas, em alguns ambientes como de CI ou dentro de containers isso acaba se tornando um problema por adicionar ruído.

Isso é contornado por variáveis de ambiente:

**PYTHON\_COLORS=?**: 0 sem cores, 1 com cores

Novidades!

Typing

# Defaults para genéricos (PEP-696)



Agora os **genéricos podem ter tipos padrões** para quando não forem definidos

```
1  from dataclasses import dataclass
2
3  @dataclass
4  class D[S = int]:
5      val: S | None = None
6
7
8  d = D()
9  d1 = D[str]()
10
11  d.val = '10' # error!
12  d1.val = 1  # error!
```

S é int

S é str


# Defaults para genéricos (PEP-696)

```
from typing import Generic, TypeVar

T = TypeVar('T', default=int)

class C(Generic[T]):
    def set_val(self, val: T) -> None:
        self.val = val

C().set_val(1)
C().set_val('1') # error
C[str]().set_val('1') # ok
```

A diagram with two red arrows. One arrow starts from the 'val: T' parameter in the 'set\_val' method and points to the 'T' variable in the 'Generic[T]' class definition. The second arrow starts from the 'default=int' argument in the 'TypeVar' definition and points to the 'T' variable in the 'Generic[T]' class definition. This illustrates how the default value 'int' is used to infer the type 'T' for the generic class.



# TypedDict: Read-only (PEP-705)



TypedDicts agora contam com o tipo **ReadOnly**, para bloquear a sobrescrita

```
from typing import TypedDict, ReadOnly, Required, NotRequired
```

```
class D(TypedDict):
```

```
    x: ReadOnly[int]
```

```
    y: int
```

```
    z: Required[float]
```

```
    w: NotRequired[str]
```

```
d: D = {'x': 1, 'y': 2, 'z': 3, 'w': 'w'}
```

```
d['x'] = 10 # error: "x" is a read-only key in "D"
```

# Typels para afinilamento de tipos (PEP-742)



Agora podemos garantir o afinilamento de tipo usando **Typels**:

```
from typing import TypeIs, assert_type

def narrower(x: I) -> TypeIs[R]: ...

def func1(val: A):
    if narrower(val):
        assert_type(val, NP)
    else:
        assert_type(val, NN)
```

# TypeGuard vs Typels



**Typels** é uma Alternativa ao **TypeGuard** para resolver alguns problemas:

1. Quando o **TypeGuard** retorna False para afinilamento, os checadores não podem a afinilar o "else".
2. **TypeGuard** não correlaciona os tipos de entrada e saída

```
from typing import TypeGuard

def is_int(x) -> TypeGuard[int]:
    return isinstance(x, int)
```

```
from typing import TypeIs

def is_int(
    argunto_de_entrada: int | str # I
) -> TypeIs[int]: # R
    return isinstance(argunto_de_entrada, int)

def vai_ser_afunilada(
    argumento: str | int # A
):
    if is_int(argumento): # AP
        reveal_type(argumento)
    else: # AN
        reveal_type(argumento)
```

```
from typing import TypeIs

def is_int(
    argunto_de_entrada: int | str # I
) -> TypeIs[int]: # R
    return isinstance(argunto_de_entrada, int)

def vai_ser_afunilada(
    argumento: str | int # A
):
    if is_int(argumento): # AP
        reveal_type(argumento)
    else: # AN
        reveal_type(argumento)
```

- I: Typels de entrada
- R: Typels tipo de retorno
- A: Argumento afunilável
- AP: Afunilamento positivo
- AN: Afunilamento negativo

```
from typing import TypeIs

def is_int(
    argunto_de_entrada: int | str # I
) -> TypeIs[int]: # R
    return isinstance(argunto_de_entrada, int)

def vai_ser_afunilada(
    argumento: str | int # A
):
    if is_int(argumento): # AP
        reveal_type(argumento)
    else: # AN
        reveal_type(argumento)
```

```
○ → mypy pep_742.py --python-version 3.13
pep_742.py:24: note: Revealed type is "builtins.int"
pep_742.py:26: note: Revealed type is "builtins.str"
Success: no issues found in 1 source file
```



# Final e ClassVar (bug corrigido)

Até a release passada não era possível dizer que um atributo de classe não deveria mudar (Final), agora esse bug foi corrigido

```
1  from dataclasses import dataclass
2  from typing import ClassVar, Final
3
4  @dataclass
5  class D:
6      val: ClassVar[Final[int | None]] = None
7
8  d = D()
9  d.val # acessível
10 d.val = 10 # error!
```

<https://github.com/python/cpython/issues/89547>

Locals, atributos  
estáticos, Docstrings

Intern  
als



# Locals (PEP-667)



A algumas releases atrás foi feita uma alteração no grame para que as variáveis locais fossem lidas do frame. Isso acarretou alguns bugs. Corrigidos nessa versão.

```
def f():  
    x = 1  
    sys._getframe().f_locals['x'] = 2  
    print(x)  
f() # Agora retorna 2!
```

# Docstrings



O compilador agora remove espaços em branco comuns de cada linha em uma docstring. Isso reduz o tamanho do cache de bytecode (como arquivos .pyc), com reduções no tamanho do arquivo de cerca de 5%, por exemplo, em sqlalchemy.orm.session do SQLAlchemy 2.0.

```
1  def func():
2      """
3      Olha
4      isso
5      é
6      uma
7      docstring.
8      """
9
10 print(repr(func.__doc__))
```

```
py-3.12.5 melville in ○ → python docstrings_test.py
'\n    Olha\n    isso\n    é\n    uma\n    docstring.\n    '

py-3.13.0 melville ○ → python docstrings_test.py
'\nOlha\nnisso\nné\numa\ndocstring.\n'
```

# Atributos estáticos



Agora todos os atributos criados em métodos arbitrários são adicionados no atributo **`__static_attributes__`**.

Melhoria para checadores e também para autocomplete:

```
class C:
    def xpto(self, val):
        self.val = val

>>> C.__static_attributes__
('a',)
```

Algumas adições  
legais!

Bibliot  
ecas

# Copy.replace()



Agora o módulo de **copy** tem uma nova função chamado **.replace** que cria um novo objeto trocando alguns atributos:

```
1  import copy
2  from dataclasses import dataclass
3
4  @dataclass
5  class C:
6      c: int = 10
7
8  >>> copy.replace(C(), c=20)
9  C(c=20)
```

# O novo dunder `__replace__`



Qualquer objeto com o dunder `__replace__`, pode suportar isso.

Objetos suportados até agora:

- `collections.namedtuple()`
- `dataclasses.dataclass`
- datetimes em geral

```
1 class C:
2     def __init__(self, a, b):
3         self.a = a
4         self.b = b
5
6     def __replace__(self, /, **changes):
7         new_attrs = self.__dict__ | changes
8         return C(**new_attrs)
9
10    def __repr__(self):
11        return f'C({self.__dict__})'
```

# warning.deprecated (PEP-702)



Agora os avisos têm um decorador de **deprecated** que podem avisar o checador de tipos (*ainda sem suporte no mypy e no pyright*) que algo está depreciado. O aviso também é levantado em runtime:

```
from warnings import deprecated

@deprecated('Atualiza essa paradinha!')
def xpto():
    ...

xpto()
---
```

\$ python dep.py  
live\_313/dep.py:9: DeprecationWarning: Atualiza essa paradinha!  
xpto()

# Experi mentos

JIT e Free Threading



# Um compilador JIT! (PEP-744)



Evolução natural das funcionalidades que vêm sendo desenvolvidas no **interpretador adaptativo especializado** (PEP 659).

Após a introdução do quickening e recompilação para super instruções agora temos **microoperações** e **superblocos** nas otimizações de **tier 2**



Interpretador adaptativo especializado - Como o desempenho do Python melhorou? | Live de Python #272

Eduardo Mendes • 3,8 mil visualizações • Transmitido há 3 semanas

Nessa live vamos conversar sobre as melhorias de performance introduzidas no interpretador da versão 3.11, que foram aperfeiçoadas na versão 3.12 e agora serão levadas ao próximo nível...

<https://youtu.be/c8ZxdwTv8N8>

# O sistema de tiers de otimização



Uma nomenclatura adotada nas otimizações na versão 3.13 são os tiers. Eles querem dizer **níveis de otimização**:

- Tier 1: Interpretador adaptativo especializado
- Tier 2: Micro operações otimizadas
  - o JIT é opcional nessa etapa!

# Tier 2

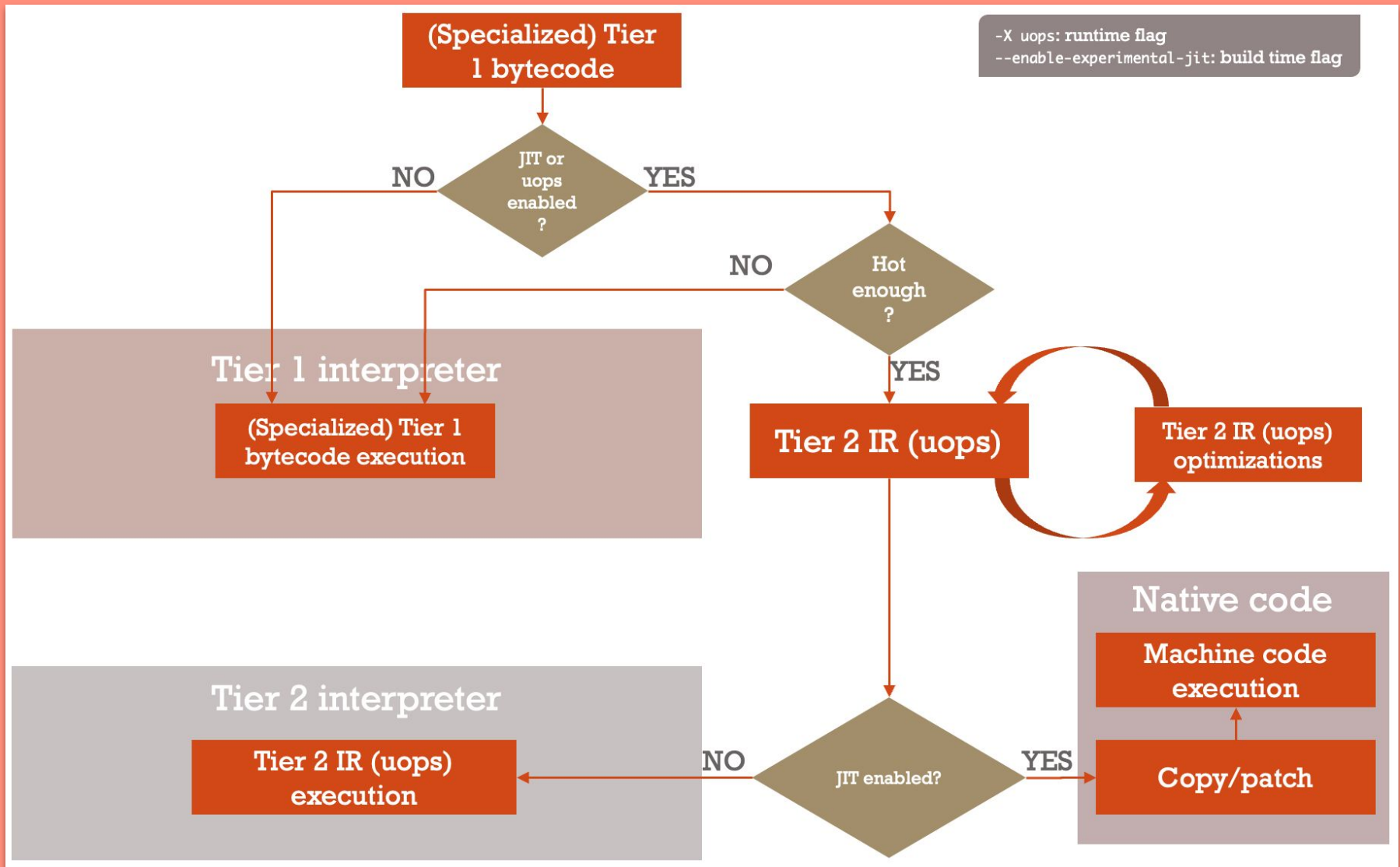


O tier dois atualmente só funciona em loops. Qual a ideia?

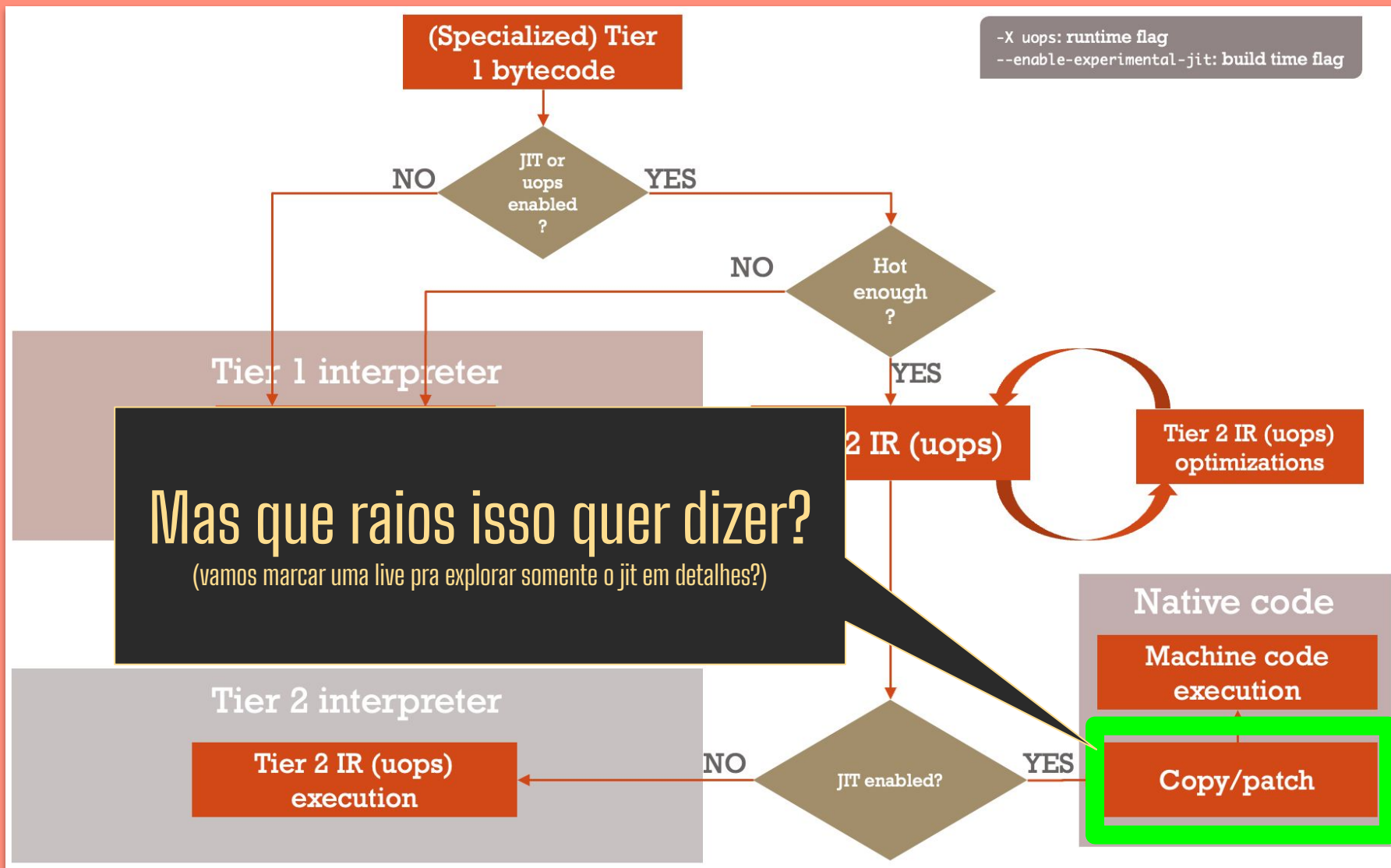
- Após a 17a (JUMP\_BACKWARD) a execução do loop ele coloca um trace!
  - Sim, **atualmente** somente loops!
  - Quando o trace é adicionado, as operações serão analisadas
  - Substituídas por micro operações

```
def fib(n):  
    a, b = 0, 1  
    for _ in range(n):  
        a, b = b, a + b  
    return a
```

```
L1:    FOR_ITER_RANGE      8 (to L2)  
      STORE_FAST          3 (_)  
  
      LOAD_FAST_LOAD_FAST 33 (b, a)  
      LOAD_FAST           2 (b)  
      BINARY_OP_ADD_INT   0 (+)  
      STORE_FAST_STORE_FAST 33 (b, a)  
      JUMP_BACKWARD       10 (to L1)
```



<https://discuss.python.org/t/pep-744-jit-compilation/50756/28>



<https://discuss.python.org/t/pep-744-jit-compilation/50756/28>

# Instalação do JIT



## PEP 744: Compilação JIT

■ PPE



Então, conversei sobre isso com o restante da equipe do Faster CPython e achamos que o esquema a seguir faria sentido para o 3.13 (nomes sujeitos a bikeshedding):

- `--enable-experimental-jit=no` (o padrão): Não crie o JIT ou o interpretador micro-op. O novo `PYTHON_JIT` variável de ambiente não tem efeito.
- `--enable-experimental-jit=interpreter`: não crie o JIT, mas *crie* e habilite o interpretador micro-op. Isso é útil para aqueles de nós que estão desenvolvendo ou depurando micro-ops (mas não querem lidar com o JIT) e é equivalente a usar o `-X uops` ou `PYTHON_UOPS=1` opções hoje. `PYTHON_JIT=0` pode ser usado para desabilitar o interpretador micro-op em tempo de execução.
- `--enable-experimental-jit=yes-off`: crie o JIT, mas não o habilite por padrão. `PYTHON_JIT=1` pode ser usado para habilitá-lo em tempo de execução.
- `--enable-experimental-jit=yes` (ou apenas `--enable-experimental-jit`): Crie o JIT e ative-o por padrão. `PYTHON_JIT=0` pode ser usado para desativá-lo em tempo de execução.

<https://discuss.python.org/t/pep-744-jit-compilation/50756/42>

# Desempenho?



Alguns dados que levantei com um script proposto pelo Nilo Menezes.  
A PEP diz que a melhoria é em média de 5%

Versão do 3.13 Tempo em segundos	
Tier 1	8.6832
Tier 2 (interpreter)	10.850
JIT (Tier 2)	8.0850
JIT (Tier 2 - No GIL)	10.602
3.12 (tier 1)	9.0079

<https://gist.github.com/dunossauro/afea7800f06d7bd1b23f8bff71bc74e1#file-resultados-md>

# Free Threading



Além da versão experimental, temos também um experimento sem a GIL (Trava global do interpretador para Threads).

Isso permite a execução de multiplas threads sem a proteção dos objetos python. Em teoria, códigos multithreads serão assombrosamente mais eficientes.

Ponto contra? O single thread é mais lento :(



# Isso também é feito ao nível de compilação



Alguns instaladores como pyenv, pdm, uv e rye já tem versões preparadas para essa versão (diferente do JIT). **Elas têm o sufixo t.**

```
pyenv install 3.13.0t
```

Nem todas as bibliotecas escritas em linguagens de mais baixo nível tem suporte a essa versão ainda.

<https://py-free-threading.github.io/tracking/>

# A platform tag `t`



O pip já suporta a abi tag para o free threading.

Os pacotes suportados podem ser vistos dessa forma:

distribution	distribution	python tag	abi tag	platform tag	.whl
pandas	2.2.3	cp313	cp313t	musllinux	.whl
pedalboard	0.9.16	cp313	cp313t	manylinux	.whl

# Desempenho?

Versão do 3.13	GIL	Sem Threads	Com Threads
Tier 1	X	8.6832	8.7207
Tier 1		10.4428	1.2694
Tier 2 (interpreter)	X	10.850	10.8371
Tier 2 (interpreter)		10.3017	1.1947
JIT (Tier 2)	X	8.0850	7.9310
JIT (Tier 2)		10.602	1.2484
Base - 3.12 (tier 1)	-	9.0079	8.6203

<https://gist.github.com/dunossauro/afea7800f06d7bd1b23f8bff71bc74e1#file-resultados-md>

JIT e no GIL  
merecem uma live cada?



Perunta!



Outras  
novidades

Mobile, CLI

# Suporte mobile e wasi (PEP-11)



- **Tier 1 (não bloqueiam lançamento, 1 core dev)**
  - **Android** agora é uma plataforma suportada no build
  - **iOS** agora é uma plataforma suportada no build
- **Tier 2 (bloqueiam lançamento, 2 cores dev)**
  - **wasi** agora tem suporte nível 2

# CLI para o random



```
python -m random
usage: random.py [-h] [-c CHOICE [CHOICE ...] | -i N | -f N]
                  [input ...]
positional arguments:
  input                if no options given, output depends on the input
                        string or multiple: same as --choice
                        integer: same as --integer
                        float: same as --float
options:
  -h, --help          show this help message and exit
  -c, --choice CHOICE [CHOICE ...]
                        print a random choice
  -i, --integer N     print a random integer between 1 and N inclusive
  -f, --float N       print a random floating-point number between 1 and N inclusive
```



[apoia.se/livedepython](https://apoia.se/livedepython)



[pix.dunossauro@gmail.com](mailto:pix.dunossauro@gmail.com)



[patreon.com/dunossauro](https://patreon.com/dunossauro)



Ajude o projeto <3

