Live de Python #43

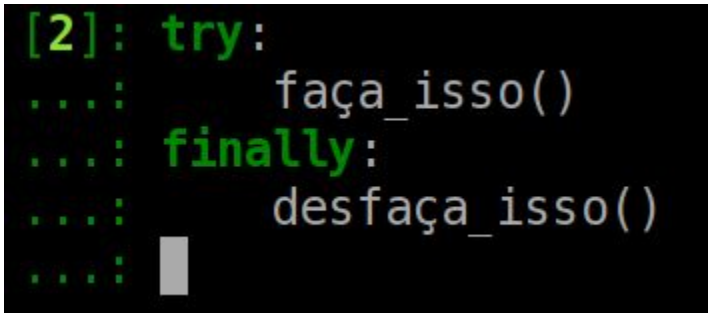
Gerenciadores de contexto

Roteiro

- Entendendo o with
  - Como usar?
  - Que tipos de problema ele quer resolver?
  - Quando devo usar?
- O que há por trás do with?
  - __enter__
  - __exit__
- contextmanager

A era antes do with

With é um statement inserido formalmente na versão 2.6 do Python para um problema idiomático:



A era antes do with

Agora, imagina que você faça algo que não deva ser feito, por exemplo, dar um novo valor a uma palavra reservada

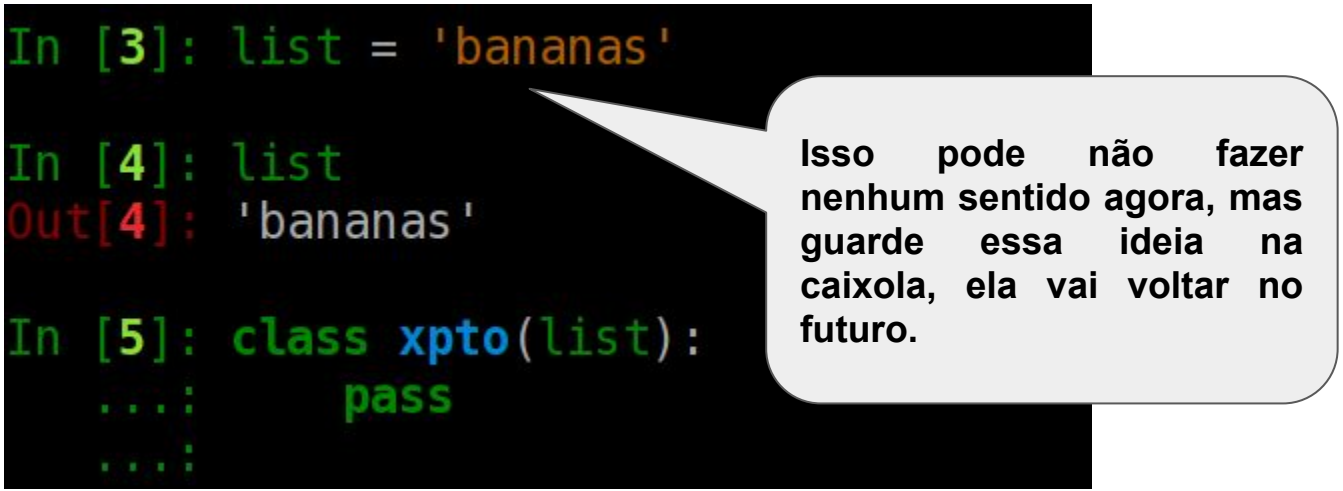
```
In [3]: list = 'bananas'

In [4]: list
Out[4]: 'bananas'

In [5]: class xpto(list):
   ...:     pass
   ...:
```

A era antes do with

Agora, imagina que você faça algo que não deva ser feito, por exemplo, dar um novo valor a uma palavra reservada



A era antes do with

Para quem é da velha guarda do python, coisas assim eram comuns de acontecer. Imagine que você quer abrir um arquivo, mas precisa que ele seja fechado, depois de escrever nele.

```
try:
    file = open('meu_arquivo.txt', 'w') #arquivo em modo de escrita
    file.write('Estamos escrevendo no arquivo')
    file.write('Fazer esse try me parece uma coisa estranha')
finally:
    file.close()
```

A era antes do with

Você pode estar se perguntando, isso não poderia ser feito sem o **try**? sim, poderia, porém, se por algum motivo o python não abrir/escrever nesse arquivo, sem o **finally** o arquivo permaneceria em estado de erro durante todo tempo de execução

```
try:
    file = open('meu_arquivo.txt', 'w') #arquivo em modo de escrita
    file.write('Estamos escrevendo no arquivo')
    file.write('Fazer esse try me parece uma coisa estranha')
finally:
    file.close()
```

A era antes do with

Agora, em outro contexto, imagine que você tenha múltiplas threads rodando, mas uma delas precisa executar uma ação perigosa. Como você travaria as outras threads?

```
try:
    thread_1.lock()
    thread_1.faça_algo()
finally:
    thread_1.unlock()
```

Acho que podemos usar with agora, não?

WITH

Agora que você já entendeu o problema, tente pensar nesse código que escreve exatamente a mesma coisa no arquivo.

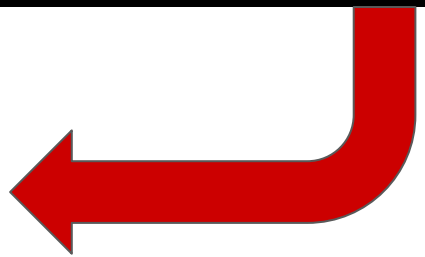
```
with open('meu_arquivo.txt', 'w') as file:
    file.write('Estamos escrevendo no arquivo')
    file.write('Fazer esse try me parece uma coisa estranha')
```

WITH

Agora que você já entendeu o problema, tente pensar nesse código que escreve exatamente a mesma coisa no arquivo.

```
with open('meu_arquivo.txt', 'w') as file:
    file.write('Estamos escrevendo no arquivo')
    file.write('Fazer esse try me parece uma coisa estranha')
```

```
with expression [as variable]:
    with-block
```



Ok, ok. Mas como isso funciona?

(PEP 343)

context manager protocol

Foi definido um protocolo para o uso de gerenciadores de contexto, os objetos que podem gerenciar contexto devem ter dois métodos implementados. __enter__ e __exit__.

Antes de exemplificar, vamos tentar entender o que eles fazem de forma indutiva.

context manager protocol

__enter__, como o próprio nome diz, ele “entra” em determinado contexto. Assim sendo é nesse caso onde as coisas vão ser “feitas”.

Em contra partida, __exit__, pode ser a parte em que ele finaliza o contexto e devolve tudo como estava anterior mente.

context manager protocol

__enter__, como me diz, ele “e rminado contexto. Assim sendo é nesse c

Em contra partida le finaliza o contexto e devolve tudo co nor mente.

context manager protocol

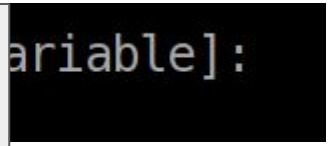
Um pouco diferente disso, quando um objeto implementa este protocolo, quando **__enter__** é chamado, ele vai nos retornar a instância do próprio objeto. O que pode parecer um pouco estranho quando comparado com os blocos **try-finally**. Mas as coisas ficam um pouco mais claras se você pensar no statement **as**.

```
with expression [as variable]:
    with-block
```

context manager protocol

Um pouco diferente disso, quando um objeto implementa este protocolo, quando **__enter__** é chamado, ele vai nos retornar algum dado para ser maniupulado nesse contexto. O que pode parecer um pouco estranho quando comparado com os blocos **try-finally**. Mas as coisas ficam um pouco mais claras se você pensar no statement **as**.

Vale lembrar que você pode ter dois tipos de objeto. Um objeto que implementa o protocolo (como o io.TextBase) e um objeto que faz a manipulação de contexto para outros objetos.



??????????????

Vale lembrar que você pode ter dois tipos de objeto. Um objeto que implementa o protocolo (como o io.TextBase) e um objeto que faz a manipulação de contexto para outros objetos.

Ok, vamos fazer código, vai ser mais fácil de entender

contextmanager

Tá, existe uma maneira mais “simples” que é usar um gerador. Quando chamamos **yield** o contexto de execução vai ser pausado.