Tuplas

MC102-2018s1-Aula13a-180412

Arthur J. Catto, PhD
ajcatto@g.unicamp.br

12 de abril de 2018

1 Tuplas

1.1 O modelo

Uma ***tupla*** é uma ***sequência de objetos***, ***ordenada***, ***imutável***, ***iterável*** e ***tipicamente heterogênea***.

Tuplas em Python são úteis para modelar o que outras linguagens de alto nível geralmente chamam de *records*. No entanto, ao invés de ter um nome como um campo de um *record*, cada elemento de uma tupla é identificado por um *índice* que indica sua posição na estrutura. A indexação é a única forma de acesso aos elementos de uma tupla.

Uma *tupla* é sempre uma estrutura ad-hoc, isto é, definida no momento de sua criação. Por isso, é difícil garantir que duas tuplas, mesmo que relacionadas, tenham o mesmo número de campos e as mesmas propriedades associadas a eles.

Assim, é fácil cometer erros difíceis de localizar, como a introdução de campos extras ou a inversão de ordem entre campos. *Namedtuples*, discutidas no final desta aula, representam uma extensão interessante desse modelo e procuram atacar alguns desses problemas.

Uma tupla é representada como uma sequência de valores (usualmente entre parênteses) separados por vírgulas.

```
In [2]: auto = ('Honda', 'City', 'DX 1.5', 2016, 52500.0)
        auto

Out[2]: ('Honda', 'City', 'DX 1.5', 2016, 52500.0)
```

O número de elementos em uma *tupla* é dado pela função `len`.

```
In [3]: len(auto)

Out[3]: 5
```

Podemos obter os elementos de uma *tupla* por indexação direta, iterando em um `for`, um `while` ou por fatiamento.

```
In [5]: for campo in auto:
            print(campo)
```

```
Honda
City
DX 1.5
2016
52500.0
```

```
In [6]: i = 0
        while i < len(auto):
            print(auto[i])
            i += 1
```

```
Honda
City
DX 1.5
2016
52500.0
```

```
In [7]: auto_2 = auto[:2] + ('EX 1.5', 2018, 69650.0)
        print(auto_2)
```

```
('Honda', 'City', 'EX 1.5', 2018, 69650.0)
```

Uma *tupla* nula ou vazia é representada apenas por um par de parênteses.

```
In [9]: nada = ()
        print(type(nada), nada)
```

```
<class 'tuple'> ()
```

A criação de uma *tupla* com um único elemento já requer um cuidado extra. Veja o que acontece...

```
In [11]: marca = ('Honda')  # isto é considerado uso normal de parênteses
         print(type(marca))
```

```
<class 'str'>
```

```
In [12]: marca = ('Honda',)  # uma vírgula antes do parêntese direito faz com que a expressão se
         print(type(marca))
```

```
<class 'tuple'>
```

1.2 Como "desempacotar" uma tupla

Numa atribuição é possível associar os elementos de uma tupla colocada do lado direito a uma tupla de variáveis colocada do lado esquerdo.

```
In [14]: marca, modelo, motor, ano, valor = auto
         print(ano, marca, modelo, motor, valor)
```

```
2016 Honda City DX 1.5 52500.0
```

```
In [15]: marca, modelo, *outros = auto
         print(marca, modelo, outros)
```

```
Honda City ['DX 1.5', 2016, 52500.0]
```

Também é possível usar uma tupla de expressões do lado direito do operador de atribuição. Nesse caso, as expressões são avaliadas da esquerda para a direita e seus resultados atribuídos às variáveis da tupla do lado esquerdo, também da esquerda para a direita.

Essa propriedade permite uma forma simples para permutar os valores de duas variáveis ou para "rodar" os valores de três ou mais variáveis.

```
In [ ]: a = 10
        b = 20
        a, b = b, a
        print('a =', a, '  b =', b)
```

```
In [ ]: a, b, a = 100, 200, 300  # a aparece duas vezes na lista de variáveis
        print('a =', a, '  b =', b)
```

```
In [ ]: a, b, c = 100, 200, 300
        print('a =', a, '  b =', b, '  c =', c)

        a, b, c = b, c, a
        print('a =', a, '  b =', b, '  c =', c)
```

O módulo `operator` oferece o extrator `itemgetter` que pode ser usado para obter um ou mais elementos de uma tupla. Na sua forma mais simples, aplicando-se `itemgetter` a um valor que representa um índice, ele gera uma função capaz de extrair o elemento naquela posição em uma sequência. Por exemplo, no caso de uma tupla...

```
In [16]: from operator import itemgetter

         t = (10, 20, 30)
         itemgetter(2)(t)
```

```
Out[16]: 30
```

1.3 Como criar tuplas a partir de listas "paralelas"

A função *zip* retorna um iterador de tuplas, onde a i-ésima tupla contém o i-ésimo elemento de cada uma das sequências ou iteradores passados como argumentos.
O iterador para quando o argumento "mais curto" se esgota.
Se for passado um único argumento, ela retorna um iterador de 1-tuplas.
Sem argumentos, ela retorna um iterador vazio.
Os argumentos iteráveis são avaliados da esquerda para a direita.

```
In [3]: la = [1, 2, 3]
        lb = [4, 5, 6]
        lc = [7, 8, 9]
        lz = zip(la, lb, lc)
        list(lz)

Out[3]: [(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

Quando combinados, a função *zip*, o comando *for* e a decomposição de tuplas permitem manipular várias sequências em paralelo, tratando os elementos correspondentes sem o uso explícito de índices.
Por exemplo, ...

```
In [5]: ls = []
        for a, b, c in zip(la, lb, lc):
            ls += [a + b + c]
        ls

Out[5]: [12, 15, 18]
```

1.3.1 Exemplo: Cálculo do produto interno de duas listas

Dadas duas listas de elementos numéricos, calcular seu produto interno.

Este problema foi resolvido como parte do exemplo *Cálculo da média ponderada* na *Aula 11*. Aqui nos interessam apenas as linhas 9-11.

O uso de *zip* para criar um iterador combinando as listas *notas* e *pesos* e depois a extração das duplas uma a uma, permitem dispensar a manipulação de índices que era necessária na implementação anterior.

```
In [17]: from random import choice

         pesos = [choice(range(1, 4)) for _ in range(10)]
         print('pesos', pesos)

         notas = [choice(range(11)) for _ in range(10)]
         print('notas', notas)

         total = 0
         for nota, peso in zip(notas, pesos):
             total += nota * peso
```

```
        média = total / sum(pesos)
        print(f'total {total}   média {média:.1f}')

pesos [3, 1, 2, 2, 3, 2, 3, 1, 1, 1]
notas [3, 3, 7, 5, 5, 5, 0, 8, 4, 7]
total 80   média 4.2
```

1.4 *Muito cuidado com aliasing*

Python tuples: immutable but potentially changing
Think labels, not boxes
by Luciano Ramalho

Tweeddledee e Tweeddledum são os gêmeos que Alice encontra em "Através do Espelho". Vamos chamá-los Dee e Dum e representá-los como *tuplas* com sua data de nascimento e habilidades.

```
In [18]: dee = ('1861-10-23', ['poesia', 'briga-simulada'])
         dum = ('1861-10-23', ['poesia', 'briga-simulada'])
```

Como Dee e Dum são gêmeos, suas representações são iguais, embora não sejam uma só.

```
In [19]: dum == dee

Out[19]: True

In [20]: dum is dee

Out[20]: False

In [21]: id(dum), id(dee)

Out[21]: (4521652296, 4521417544)
```

Por outro lado, como ambos estão representados como tuplas, que são objetos imutáveis, não deve ser possível destruir a igualdade entre eles, certo?

Vamos criar um *clone* de Dum.

```
In [22]: doom = dum

In [23]: doom == dum

Out[23]: True

In [24]: doom is dum

Out[24]: True

In [25]: id(doom), id(dum)

Out[25]: (4521652296, 4521652296)
```

Suponha agora que Doom tenha se tornado *rapper*. Vamos acrescentar isso às suas habilidades. Estas estão representadas em `doom[1]`, certo?

```
In [26]: doom[1]

Out[26]: ['poesia', 'briga-simulada']

In [27]: doom[1].append('rap')
         doom

Out[27]: ('1861-10-23', ['poesia', 'briga-simulada', 'rap'])
```

Vamos exibir as *tuplas* que representam Dee, Dum e Doom novamente.

```
In [28]: dee
         dum
         doom

Out[28]: ('1861-10-23', ['poesia', 'briga-simulada'])

Out[28]: ('1861-10-23', ['poesia', 'briga-simulada', 'rap'])

Out[28]: ('1861-10-23', ['poesia', 'briga-simulada', 'rap'])

In [29]: dum == dee

Out[29]: False
```

Ooops! `dum` e `dee` deixaram de ser iguais!
Mas tuplas não são imutáveis?

- Sim, isto é, **nem sempre**...

Moral da história

> Nunca use um objeto mutável como componente de um objeto "teoricamente" imutável.
> Você pode se surpreender!

1.5 Named tuples

Named tuples representam uma extensão ao modelo de *tuplas* buscando atacar alguns dos problemas acima.
Elas são sequências ordenadas, "imutáveis" e iteráveis como *tuplas* normais, mas seus elementos podem também ser referenciados por nome, como os campos dos *records* de outras linguagens de alto nível.

```
In [30]: from collections import namedtuple

         Auto = namedtuple('Auto', 'marca modelo motor ano valor')

         auto = Auto('Honda', 'City', 'DX 1.5', 2016, 52500.0)
         print(auto)
```

```
Auto(marca='Honda', modelo='City', motor='DX 1.5', ano=2016, valor=52500.0)
```

```
In [32]: auto.ano = 2018
```

```
        ---------------------------------------------------------------------------
        AttributeError                            Traceback (most recent call last)

        <ipython-input-32-d0999d039f0b> in <module>()
    ----> 1 auto.ano = 2018


        AttributeError: can't set attribute
```

```
In [33]: print(auto.valor, auto[4])
```

```
52500.0 52500.0
```

1.5.1 No entanto, o problema com aliasing permanece

Vamos criar uma *namedtuple* para conter os dados pessoais dos gêmeos e depois criar as respectivas entradas.

```
In [44]: Bio = namedtuple('Bio', 'aniversário habilidades')
```

```
In [45]: dee = Bio('1861-10-23', ['poesia', 'briga-simulada'])
         dee
```

```
Out[45]: Bio(aniversário='1861-10-23', habilidades=['poesia', 'briga-simulada'])
```

```
In [46]: dum = Bio('1861-10-23', ['poesia', 'briga-simulada'])
         dum
```

```
Out[46]: Bio(aniversário='1861-10-23', habilidades=['poesia', 'briga-simulada'])
```

Vamos verificar a igualdade e a distinção das *namedtuples*.

```
In [47]: dee == dum
```

```
Out[47]: True
```

```
In [48]: dee is dum
```

```
Out[48]: False
```

```
In [49]: id(dum), id(dee)
```

```
Out[49]: (4523418032, 4523417744)
```

Vamos criar Doom, o clone de Dum.

```
In [50]: doom = dum
         doom

Out[50]: Bio(aniversário='1861-10-23', habilidades=['poesia', 'briga-simulada'])

In [51]: doom == dum

Out[51]: True

In [52]: doom is dum

Out[52]: True
```

Vamos acrescentar `rap` às habilidades de Doom.

```
In [53]: doom[1].append('rap')
         doom

Out[53]: Bio(aniversário='1861-10-23', habilidades=['poesia', 'briga-simulada', 'rap'])
```

Como `doom` e `dum` são "rótulos" que estão "colados" no mesmo objeto, isto também afeta as habilidades de Dum que, por estarem representadas numa *namedtuple*, **deveriam ser imutáveis**.

```
In [54]: dum

Out[54]: Bio(aniversário='1861-10-23', habilidades=['poesia', 'briga-simulada', 'rap'])
```

Como `dee` continua o mesmo, `dum` e `dee` deixaram de ser iguais, o que, teoricamente, não poderia acontecer.

```
In [55]: dee

Out[55]: Bio(aniversário='1861-10-23', habilidades=['poesia', 'briga-simulada'])

In [56]: dee == dum

Out[56]: False
```