Tuplos e Listas

Carlos Tavares

28 de Outubro de 2024

Tuplos

Tuplos i

Alguns tipos de dados fazem mais sentido estarem "juntos". Por exemplo, um **ponto**:

- Até agora representámos um ponto utilizando duas variáveis:

  » x = 5
  » y = 5

- O python permite-nos representar a mesma informação utilizando uma única variável através de um **tuplo:**

  » ponto = (5,5)

Tuplos ii

Outro exemplo: uma **pessoa**

» # (Nome, Altura, Peso (Kg), Tipo sanguíneo)
» pessoa = ("Jonh Doe", 1.80, 65, "AB+")

Informalmente, corresponde à noção de **registo** (**record**), ou seja, um tipo de dados que agrega várias informações de dimensões diferentes sobre um objecto

Tuplos iii

Sintaticamente, um tuplo é representado como uma construção do tipo:

» (campo_1, campo_2, campo_3, . . ., campo_n)

onde n é qualquer dimensão finita.

É uma forma de construir tipos compostos em Python

$$Tipo \times Tipo \ldots \times Tipo$$

Os tuplos em Python são imutáveis.

Como lidar com tuplos em Python i

Formas de aceder aos elementos de um tuplo:

Um elemento específico: **variável [índice]**

Exemplo: a variável ponto

```
» ponto [0]
5
» ponto [1]
6
```

Como lidar com tuplos em Python ii

Um intervalo de elementos: **variável [i:j]**

» pessoa [1:3]
(23,42)

» pessoa [1:]
(1.8, 65, "AB+")

» pessoa [:3]
('Jonh Doe', 1.8, 65)

Como lidar com tuplos em Python iii

Pattern matching usando tuplos:

» (**nome**, **altura**, **peso**, **tipo_sanguíneo**) = pessoa
» **nome**
Joaquim
» **altura**
1.80

Os tuplos são iteráveis:

```
for i in pessoa:
    print (i)
```

Imutabilidade dos Tuplos i

Tuplos são imutáveis

» pessoa[0] = "Doe John"
TypeError: 'tuple' object does not support item assignment

No entanto, é sempre possível criar novos tuplos!

» pessoa = ("Desconhecido", 1.60, 65, "0-")

Uma mudança pode ser feita substituindo apenas um campo:
» pessoa = pessoa [0] + (1,70, 80, "0-")

Também é possível criar cópias de tuplos
» pessoa *= 2

Listas

Listas i

Uma lista é sintaticamente representada em **Python** como:

[elem_1, elem_2, . . ., ... , elem_3]

Exemplo:

- $[12, 13, 14, 15, 16]$
- $['a', 'b', 'c', 'd']$

Listas ii

Tuplos vs **Listas**

Os tuplos são imutáveis, ou seja, uma vez definidos, os seus elementos não podem ser alterados directamente e não se pode acrescentar ou eliminar elementos. Listas podem ter elementos adicionados e removidos arbitrariamente.

É possível criar listas a partir de tuplos:

» list (tuple)

E tuplos a partir de listas

» tuple ([1,2,3,4])

Listas iii

Mais exemplos

- Lista vazia - []
- Lista de listas - [[1, 2, 3, 4], [4, 3, 2, 1]]
- Lista de listas de listas - [[[1, 2, 3, 4], [4, 3, 2, 1]], [[5, 6, 7, 8], [8, 7, 6, 5]]]
- Listas de diferentes tipos - [12, "Joaquina", [1,2,3,4]]

Acedendo a elementos de listas (Muito similar aos tuplos):

variável_lista [índice]

Listas iv

Por *pattern matching*:

```
» [x, y, z] = [12, "Joaquina", [1,2,3,4]]
» x
12
» y
?
» z
?
» z [3]
?
```

Listas v

É possível **selecionar** intervalos em listas:

» l = [1,2,3,4,5,6,7,8,9, 10]

» **print** (l)
[1,2,3,4,5,6,7,8,9, 10]

» **print** (l[4:8])
[5, 6, 7, 8]

» **print** (l[:7])
[1, 2, 3, 4, 5, 6, 7]

» **print** (l[3:])
[4, 5, 6, 7, 8, 9, 10]

Listas vi

```
» print (l[0:-2])
[1, 2, 3, 4, 5, 6, 7, 8]
```

Listas vii

Listas também são objetos iteráveis

```
for i in l:
        print (i)
```

Alternativamente, pode-se usar a função **len**

```
i = 0
while i < len (l):
        print (l [i])
        i = i + 1
```

Operadores sobre listas i

Operador ==

Retorna true se a lista for exactamente a mesma!

» [1,2,3,4] == [1,2,3]
False
» [1,2,3,4] == [1,2,3, 4]
True
» [1,2,3,4] == [2,3,1, 4]
False

Operadores sobre listas ii

Operador in

Retorna true se um elemento pertence à lista:

» l = [2,3,4, 5]
» 1 in l
False
» 2 in l
True

Operador not in

Comporta-se da maneira contrária ao operador in
» l = [2,3,4, 5]
» 1 not in l
True

Operadores sobre listas iii

Concatenação de listas (+)

É possível juntar listas com o operador +

```
» l = [1,2,3,4]
» m = [5,6,7,8]
» l + m
[1, 2, 3, 4, 5, 6, 7, 8]
```

Multiplicação de listas (*)

É possível multiplicar listas por elas próprias

```
» l = [1,2,3,4]
» l*=2
»l
[1, 2, 3, 4, 1, 2, 3, 4]
```

Adição de elementos a uma lista

A função **append**:

lista.**append** (elemento)

A função **insert**:

lista.**insert** (índice, elemento)

Exemplo:

```
lista = [1, 2, 3, 10]
lista.append (11)
lista.insert (0, 0)
```

Remoção de elementos de uma lista i

A função **remove**:

lista.remove (item)

A função **pop**:

lista.pop (índice)

Exemplo:

l = [1,2,3,4]
l.remove (3)
l.pop(2) » l
[1, 2]

Remoção de elementos de uma lista ii

A função **clear**

A função clear apaga todos os elementos de uma lista.

Exemplo

» l = [1,2,3,4,5]
» l.clear ()
» l
[]

Enumerations i

Uma lista pode ser percurrida com o método enumerate, que também mantém tracking das posições de cada elementos:

```
l = [1,2,3,4,5,6]

for i, j in enumerate (l):
        print (i, "==>", j)
```

Resultado:

```
0 ==> 1
1 ==> 2
2 ==> 4
3 ==> 5
4 ==> 6
```

Enumerations ii

O objecto enumerate é preguiçoso.

Funções úteis para listas i

A função **sort** (Ordena uma lista em ordem crescente)

» l = [5,4,3,2,1]
» l.**sort** ()
» l
[1,2,3,4,5]

A função **reverse**

» l = [4,5,3,1,2]
» l.**reverse** ()
» l
[2,1,3,5,4]

Funções úteis para listas ii

Converter um tuplo em uma lista. O construtor list:

Exemplo:

```
» t = ("John Doe", 87, 1.80)
»l = list (t)
»l
["John Doe", 87, 1.80]
```

Funções úteis para listas iii

Contar as ocorrências de um elemento em uma lista. A função **count**.

Exemplo:

```
» l = [2, 2, 3 ,4 ,4, 4, 4, 5]
» l.count(4)
4
»l.count(3)
1
```

Exercício 1: Dada uma lista, faça uma função para encontrar o seu menor elemento. Como poderia ser isto feito com uma função recursiva?

Exercício 2: Dada uma lista, faça uma função para eliminar elementos repetidos.

Exercício 3: Considere um serviço público, que precisa da implementação de um programa para gerir a sua fila de clientes, com os seguintes requisitos:

- Os clientes registram seu nome, idade e prioridade (True ou False);
- A fila deve conter os clientes com prioridade antes dos que não têm.
- Deve haver uma opção para mostrar quem está à espera na fila;
- Deve haver uma opção para remover um cliente da fila, de uma posição específica.

Perguntas?