Ficha 2 - Estruturas de dados resolvida

Jorge Gustavo Rocha

May 30, 2020

1 Estruturas de Dados em Python

Vamos começar por olhar para as estruturas de dados em Python que nos ajudam a representar coleções de dados.

Faça estes exercícios no Jupyter.

1.1 Documentação sobre Python 3

1.2 Documentação sobre o Jupyter

Enquanto não estiver familizarizado com o Jupyter, consulta a documentação disponível.

Se estiver mesmo a começar, leia primeiro um pequeno tutorial, como este Jupyter Notebook: An Introduction, entre os muitos que existem na net.

Para acelerar a edição no Jupyter, consulte a lista de atalhos que existem para criar células, mudar o tipo de célula, executar, etc. Além dos atalhos, há mais uns truques que dão jeito.

Use também um guia de Markdown para formatar as células de texto.

Ao resolver os exercícios, adicione caixas de texto com as suas próprias explicações. Quanto melhor forem as suas explicações, mais fácil será reler o código que escreveu daqui a alguns dias.

1.3 Listas

```
concelhos = [ "Amares", "Barcelos", "Braga", "Cabeceiras de Basto", "Celorico de Basto", "Espos
```

1.4 Dicionários

```
populacao = { "Amares": 19853, "Barcelos": 124555, "Braga": 176154, "Cabeceiras de Basto": 176
```

1.5 Arrays

Os arrays são indicados para armazenar uma coleção de elementos do mesmo tipo. As listas, com se viu, podem conter elementos de vários tipos diferentes. Os arrays, para serem mais eficientes, apenas permitem elementos do mesmo tipo (booleanos, inteiros, reais, etc).

Os arrays podem ser criados com o módulo `array` ou com o módulo `numpy`. Ambas as implementações são semelhantes. Nestes exercícios, vamos usar sempre o módulo `numpy` para trabalhar com arrays.

```
import numpy as np
vel = np.array([ 50, 50, 70, 90, 120 ])
```

1.5.1 Arrays com números aleatórios

```
np.random.seed(0)
notas = np.random.randint(100, size=10)
```

1.6 Exercícios de listas

Considere a lista `concelhos` definida anteriormente.

1. Calcule o números de elementos da lista.
2. Calcule o primeiro elemento da lista.
3. A partir da lista inicial, crie um nova lista, com a apenas o primeiro e o último elemento da lista.
4. Para ordenar listas, pode usar duas funções diferentes: `list.sort()` e `sorted()`. Escreva dois exemplos, com cada uma das funções, para mostrar as diferenças. Veja também como se usa o parâmetro `reverse=True` nestas duas funções.

1.7 Resolução dos exercícios com listas

Dada a seguinte lista, guardada na variável `concelhos`:

[2]:
```
concelhos = [ "Amares", "Barcelos", "Braga", "Cabeceiras de Basto", "Celorico␣
↪de Basto", "Esposende", "Fafe", "Guimarães", "Póvoa de Lanhoso", "Terras de␣
↪Bouro", "Vieira do Minho", "Vila Nova de Famalicão", "Vila Verde", "Vizela" ]
```

[6]:
```
# Número de elementos da lista
len(concelhos)
```

[6]: 14

[11]:
```
# Primeiro elemento da lista
concelhos[0]
```

[11]: 'Amares'

O último elemento de uma lista pode ser indexado com `-1`. Ou seja, `concelhos[-1]` dá-nos o último elemento da lista. Neste caso específico, também se podia aceder ao último elemento com `concelhos[13]`.

[12]:
```
[concelhos[0], concelhos[-1]]
```

[12]: ['Amares', 'Vizela']

1.7.1 A função `sorted()`

A função `sorted()` pega numa lista retorna uma nova lista com os elementos ordenados.

[13]:
```
sorted(concelhos)
```

[13]:
```
['Amares',
 'Barcelos',
 'Braga',
 'Cabeceiras de Basto',
 'Celorico de Basto',
 'Esposende',
 'Fafe',
 'Guimarães',
 'Póvoa de Lanhoso',
 'Terras de Bouro',
 'Vieira do Minho',
 'Vila Nova de Famalicão',
 'Vila Verde',
 'Vizela']
```

A função `sorted()` pode levar uma parâmetro para inverter a ordem de ordenação.

[31]:
```
sorted(concelhos, reverse=True)
```

[31]:
```
['Vizela',
 'Vila Verde',
 'Vila Nova de Famalicão',
 'Vieira do Minho',
 'Terras de Bouro',
 'Póvoa de Lanhoso',
 'Guimarães',
 'Fafe',
 'Esposende',
 'Celorico de Basto',
 'Cabeceiras de Basto',
 'Braga',
 'Barcelos',
 'Amares']
```

As listas podem ser ordenadas usando o **resultado** de uma função. No exemplo seguinte ordena-se a lista de concelhos, aplicando a funçao `len()` a cada concelho. A função `len()` calcula o tamanho de uma string.

[35]:
```
sorted(concelhos, key=len)
```

[35]:
```
['Fafe',
 'Braga',
 'Vizela',
```

```
 'Amares',
 'Barcelos',
 'Guimarães',
 'Esposende',
 'Vila Verde',
 'Vieira do Minho',
 'Terras de Bouro',
 'Póvoa de Lanhoso',
 'Celorico de Basto',
 'Cabeceiras de Basto',
 'Vila Nova de Famalicão']
```

1.7.2 O método `list.sort()`

O método `sort()` da classe `list` modifica a própria lista. Não devolve uma nova lista. Ou seja, a lista passa a ficar armazenada com a ordenação imposta por `list.sort()`.

[16]:
```
# Vamos alterar a ordem pela qual são armazenados os elementos da lista
list.sort(concelhos, reverse=True)
# Vamos mostrar que os elementos passaram a estar armazenados por outra ordem
print(concelhos)
```

```
['Vizela', 'Vila Verde', 'Vila Nova de Famalicão', 'Vieira do Minho', 'Terras de
Bouro', 'Póvoa de Lanhoso', 'Guimarães', 'Fafe', 'Esposende', 'Celorico de
Basto', 'Cabeceiras de Basto', 'Braga', 'Barcelos', 'Amares']
```

1.8 Exercícios com dicionários

Considere o dicionário `populacao` atrás definido.

1. Escreva a expressão que nos dá a polulação de `Vizela`
2. Diga qual é o concelho mais populoso
3. Diga qual é a soma da população de todos os concelhos
4. Imprima os concelhos e respetiva população por ordem descrescente de população

1.9 Resolução dos exercícios com dicionários

[36]:
```
populacao = { "Amares": 19853, "Barcelos": 124555, "Braga": 176154, "Cabeceiras␣
↪de Basto": 17635, "Celorico de Basto": 19767, "Esposende": 35552, "Fafe":␣
↪53600, "Guimarães": 162636, "Póvoa de Lanhoso": 24230, "Terras de Bouro":␣
↪7506, "Vieira do Minho": 14077, "Vila Nova de Famalicão": 134969, "Vila␣
↪Verde": 49171, "Vizela": 24477 }
```

Os dicionários têm um conjunto de chaves e de valores. Dá para separar cada um destes concjuntos com:

[38]:
```
populacao.keys()
```

[38]:
```
dict_keys(['Amares', 'Barcelos', 'Braga', 'Cabeceiras de Basto', 'Celorico de
Basto', 'Esposende', 'Fafe', 'Guimarães', 'Póvoa de Lanhoso', 'Terras de Bouro',
'Vieira do Minho', 'Vila Nova de Famalicão', 'Vila Verde', 'Vizela'])
```

[40]:
```
populacao.values()
```

[40]:
```
dict_values([19853, 124555, 176154, 17635, 19767, 35552, 53600, 162636, 24230,
7506, 14077, 134969, 49171, 24477])
```

Para sabermos a população de **"Vizela"**, precisamos de ir buscar o valor associado à chave `"Vizela"`. Podemos fazê-lo de duas formas:

[28]:
```
# população de Vizela
populacao["Vizela"]
```

[28]:
```
24477
```

[30]:
```
# população de Vizela
# forma alternativa
populacao.get("Vizela")
```

[30]:
```
24477
```

A expressão `max(populacao)` daria-nos o máximo das chaves (que seria "Vizela"). Como queremos o máximo dos valores, temos que usar o parâmetro `key` com o método `populacao.get()` que nos permite ir bucar os valores associados às chaves. Assim o máximo é calculado em funçã dos valores e não das chaves.

[41]:
```
# concelho mais populoso
max(populacao, key=populacao.get)
```

[41]:
```
'Braga'
```

[44]:
```
# População de todos os concelhos
sum(populacao.values())
```

[44]:
```
864182
```

Imprimir os concelhos e respetiva população por ordem descrescente de população

[91]:
```
# Só ordena e devolve as chaves
# sorted(populacao, reverse=True, key=populacao.get)
[ (chave, populacao[chave]) for chave in sorted(populacao, reverse=True,␣
 ↪key=populacao.get) ]
```

[91]:
```
[('Braga', 176154),
 ('Guimarães', 162636),
 ('Vila Nova de Famalicão', 134969),
 ('Barcelos', 124555),
```

```
 ('Fafe', 53600),
 ('Vila Verde', 49171),
 ('Esposende', 35552),
 ('Vizela', 24477),
 ('Póvoa de Lanhoso', 24230),
 ('Amares', 19853),
 ('Celorico de Basto', 19767),
 ('Cabeceiras de Basto', 17635),
 ('Vieira do Minho', 14077),
 ('Terras de Bouro', 7506)]
```

[82]:
```
# forma alternativa, usando uma função lambda (função sem nome) que ordena em␣
 ↪função do valor do dicionário
sorted(populacao.items(), reverse=True, key=lambda item: item[1])
```

[82]:
```
[('Braga', 176154),
 ('Guimarães', 162636),
 ('Vila Nova de Famalicão', 134969),
 ('Barcelos', 124555),
 ('Fafe', 53600),
 ('Vila Verde', 49171),
 ('Esposende', 35552),
 ('Vizela', 24477),
 ('Póvoa de Lanhoso', 24230),
 ('Amares', 19853),
 ('Celorico de Basto', 19767),
 ('Cabeceiras de Basto', 17635),
 ('Vieira do Minho', 14077),
 ('Terras de Bouro', 7506)]
```

1.10 Exercícios com arrays

Considere os arrays atrás definidos.

1. Calcule a médias das velocidades registadas no vetor `vel`
2. Calcule a velocidade mínima no vetor `vel`
3. Crie um vetor com 10 elementos, com números aleatórios entre -10 e 10.

[93]:
```
import numpy as np
vel = np.array([ 50, 50, 70, 90, 120 ])
np.random.seed(0)
notas = np.random.randint(100, size=10)
```

[97]:
```
np.mean(vel)
```

[97]:
```
76.0
```

[99]:
```
np.min(vel)
```

[99]: 50

[103]:
```
np.random.randint(-10, 11, 10)
```

[103]:
```
array([ -1,  10,   9,   6,   9,  -5,   5,   5, -10,   8])
```