Dicionários

MC102-2018s1-Aula13b-180412

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

12 de abril de 2018

1 Dicionários

1.1 O modelo

Listas em Python são um instrumento muito poderoso quando desejamos extrair itens com base na sua posição em uma sequência.

Por exemplo, suponha a tabela com os nomes das cidades brasileiras mais populosas publicada pelo IBGE:

Se criarmos uma lista como...

```
In [2]: cidades = ['São Paulo', 'Rio de Janeiro', 'Brasíia', 'Salvador', 'Fortaleza']
```

... é possível responder diretamente a uma pergunta como "Qual a n-ésima maior cidade do Brasil?".

```
In [11]: n = int(input("Qual a posição da cidade desejada? "))
         print(f'A {n}a maior cidade do Brasil é {cidades[n - 1]}.')
```

```
A 2a maior cidade do Brasil é Rio de Janeiro.
```

Numa lista, o valor do item que se acha numa determinada posição é recuperado de forma direta. Essa operação tem "custo" constante, isto é, independente da posição do item e do tamanho da lista.

Suponha que quiséssemos também responder a "Qual a população de x?", onde x é o nome de uma cidade na tabela.

Como fazer isso?

Seria possível, por exemplo, criar uma segunda lista, com a população de cada cidade, na mesma ordem.

```
In [5]: população = [12106920, 6520266, 3039444, 2953986, 2627482]
```

Nesse caso, para obter a população de x, será necessário fazer uma busca, explicitamente ou usando uma função do sistema. Supondo que x esteja na lista...

```
In [12]: cidade = input("Qual a cidade desejada? ")
         i = 0
         while cidades[i] != cidade:
             i += 1
         print(f'A população de {cidade} é {população[i]}.')
```

```
A população de Salvador é 2953986.
```

ou...

```
In [13]: cidade = input("Qual a cidade desejada? ")
         print(f'A população de {cidade} é {população[cidades.index(cidade)]}.')
```

```
A população de Salvador é 2953986.
```

É possível também, por exemplo, criar uma lista de tuplas, reunindo a cidade e a respectiva população.

```
In [15]: cid_pop = [('São Paulo', 12106920),
                    ('Rio de Janeiro', 6520266),
                    ('Brasíia', 3039444),
                    ('Salvador', 2953986),
                    ('Fortaleza', 2627482)
                   ]
```

E nesse caso, a resposta à pergunta pode ser obtida, por exemplo, por...

```
In [17]: x = input("Qual a cidade desejada? ")
         for cidade, população in cid_pop:
             if cidade == x:
                 break
         print(f'A população de {cidade} é {população}.')
```

```
A população de Salvador é 2953986.
```

Em todos os casos, a operação de busca tem complexidade $\mathcal{O}(n)$, isto é o tempo de execução passa a ser proporcional ao tamanho da lista.

Um ***dicionário*** em Python permite resolver esse problema de uma forma muito mais eficiente.

Um ***dicionário*** é uma ***coleção não ordenada de objetos, mutável, iterável*** e ***potencialmente heterogênea***.

Os itens de um dicionários são acessíveis por ***chaves*** e não por índices, como nas estruturas sequenciais (listas, strings e tuplas) que já estudamos.

A implementação de dicionários usa uma técnica chmada *hashing* (que estudaremos mais à frente).

O uso de *hashing* faz com que o tempo de busca de um determinado valor seja potencialmente constante, em vez de crescer linearmente com o tamanho da estrutura, como acontece nas listas.

Um dicionário é representado por um conjunto de pares ***chave: valor***, entre { e } e separados por ***vírgulas***.

A ***chave*** deve pertencer a um tipo imutável (isto é, nada de *listas* ou *dicionários*), enquanto o ***valor*** pode ser de qualquer tipo.

Por exemplo:

```
In [24]: cid2pop = {'São Paulo': 12106920,
                    'Rio de Janeiro': 6520266,
                    'Brasília': 3039400,
                    'Salvador': 2953986,
                    'Fortaleza': 2627482
                   }
```

E agora, para obter a população de x, basta dizer

```
In [25]: x = input("Qual a cidade desejada? ")
         print(f'A população de {cidade} é {cid2pop[x]}.')
```

```
A população de Salvador é 2953986.
```

Em compensação, os itens de um dicionário não são acessíveis por índices...

```
In [26]: cid2pop[1]
```

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

<ipython-input-26-019bd9c87f36> in <module>()
----> 1 cid2pop[1]


KeyError: 1
```

1.2 Como criar um dicionário

O exemplo acima mostrou como criar um dicionário. Se quiséssemos um dicionário *dic* vazio faríamos

```
In [53]: dic = {}
         dic
```

```
Out[53]: {}
```

1.3 Como obter e modificar os itens de um dicionário

Já vimos como obter o valor associado a uma chave.
A população de Brasília está errada. Vamos corrigi-la...

```
In [27]: cid2pop['Brasília'] = 3039444
```

O mesmo procedimento permite incluir novos itens no dicionário:

```
In [28]: cid2pop['Belo Horizonte'] = 2523000
         cid2pop['Manaus'] = 2130264

In [29]: cid2pop

Out[29]: {'Belo Horizonte': 2523000,
          'Brasília': 3039444,
          'Fortaleza': 2627482,
          'Manaus': 2130264,
          'Rio de Janeiro': 6520266,
          'Salvador': 2953986,
          'São Paulo': 12106920}
```

É possível também incluir ou atualizar mais do que um novo item de uma vez só, mesclando-se (*merging*) um outro dicionário com o método **update**.

```
In [30]: cid2pop.update({'Curitiba': 1908359, 'Recife': 1633697, 'Porto Alegre': 1484941, 'Belo
         cid2pop

Out[30]: {'Belo Horizonte': 2523794,
          'Brasília': 3039444,
          'Curitiba': 1908359,
          'Fortaleza': 2627482,
          'Manaus': 2130264,
          'Porto Alegre': 1484941,
          'Recife': 1633697,
          'Rio de Janeiro': 6520266,
          'Salvador': 2953986,
          'São Paulo': 12106920}
```

O acesso com uma chave inexistente provoca um erro

```
In [31]: cid2pop['Campinas']


         ---------------------------------------------------------------------------

         KeyError                                  Traceback (most recent call last)

         <ipython-input-31-e39e634706f3> in <module>()
    ----> 1 cid2pop['Campinas']
```

```
KeyError: 'Campinas'
```

Para evitar o erro, usa-se o método **get** que aceita um parâmetro com o valor que deve ser retornado, caso a chave consultada não seja encontrada.

```
In [32]: cid2pop.get('Campinas', 'Não sei...')

Out[32]: 'Não sei...'
```

1.4 Como testar a existência de um item num dicionário

O erro numa consulta com uma chave inexistente também pode ser evitado protegendo-se a consulta por um teste de existência com os operadores **in** e **not in**.

```
In [33]: 'Campinas' in cid2pop

Out[33]: False

In [34]: 'Campinas' not in cid2pop

Out[34]: True
```

... o que nos permite fazer...

```
In [35]: if 'Campinas' in cid2pop:
             resposta = cid2pop['Campinas']
         else:
             resposta = 'não sei'
         resposta

Out[35]: 'não sei'
```

Esse comando também pode ser escrito de forma mais simples como

```
In [36]: resposta = cid2pop['Campinas'] if 'Campinas' in cid2pop else 'não sei'
         resposta

Out[36]: 'não sei'
```

1.5 Como remover itens de um dicionário

Para remover um item de um dicionário usa-se o comando **del**

```
In [37]: del cid2pop['Manaus']
         cid2pop
```

```
Out[37]: {'Belo Horizonte': 2523794,
          'Brasília': 3039444,
          'Curitiba': 1908359,
          'Fortaleza': 2627482,
          'Porto Alegre': 1484941,
          'Recife': 1633697,
          'Rio de Janeiro': 6520266,
          'Salvador': 2953986,
          'São Paulo': 12106920}
```

É possível obter o valor de um item e, ao mesmo tempo, removê-lo usando-se o método **pop**.

```
In [39]: print(cid2pop.pop('Fortaleza'))
```

```
2627482
```

```
In [40]: cid2pop
```

```
Out[40]: {'Belo Horizonte': 2523794,
          'Brasília': 3039444,
          'Curitiba': 1908359,
          'Porto Alegre': 1484941,
          'Recife': 1633697,
          'Rio de Janeiro': 6520266,
          'Salvador': 2953986,
          'São Paulo': 12106920}
```

A tentativa de remoção de uma chave inexistente gera um erro...

```
In [41]: print(cid2pop.pop('Campinas'))
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-41-bd09199dcbe3> in <module>()
    ----> 1 print(cid2pop.pop('Campinas'))


        KeyError: 'Campinas'
```

É possível evitar o erro na remoção de uma chave inexistente dando *None* como um segundo argumento para **pop**.

```
In [42]: print(população.pop('Campinas', None))
```

```
None
```

Para remover todas os itens de um dicionário usa-se o método **clear**.
Clear esvazia o dicionário associado à variável à qual ele é aplicado.

- A variável continua associada ao mesmo objeto (isto é, seu **id** não se altera).
- Apenas o valor do objeto é que foi alterado (neste caso para *vazio*).

```
In [43]: dic = {'a': 123, 'b': 456, 'c': 789}
         id(dic), dic

Out[43]: (4463474728, {'a': 123, 'b': 456, 'c': 789})

In [44]: dic.clear()
         id(dic), dic

Out[44]: (4463474728, {})
```

Note que isto não é o mesmo que atribuir {} à variável. Neste caso a variável deixa de estar associada ao antigo dicionário (que continua existindo) e passa a estar associada a outro dicionário (neste caso um dicionário vazio). Com isso, o **id** da variável muda, como mostram os exemplos a seguir.

```
In [45]: dic = {'a': 123, 'b': 456, 'c': 789}
         id(dic), dic

Out[45]: (4435158072, {'a': 123, 'b': 456, 'c': 789})

In [46]: dic = {}
         id(dic), dic

Out[46]: (4435383136, {})
```

1.6 Iterações sobre dicionários

É possível iterar facilmente sobre as chaves ou valores de um dicionário

```
In [63]: cid2pop = {'Salvador': 2954000,
                    'Fortaleza': 2627482,
                    'Belo Horizonte': 2523794,
                    'São Paulo': 12106920,
                    'Brasília': 3039444,
                    'Rio de Janeiro': 6520266,
                    'Manaus': 2130264}

In [51]: for x in cid2pop:
             print(f'{x:16}  {cid2pop[x]:8}')

São Paulo         12106920
Rio de Janeiro     6520266
Brasília           3039444
Salvador           2953986
```

```
Belo Horizonte     2523794
Curitiba           1908359
Recife             1633697
Porto Alegre       1484941
```

O mesmo resultado pode ser conseguido de um jeito mais *pythoniano* por

```
In [50]: for x, popx in cid2pop.items():
             print(f'{x:16}  {popx:8}')
```

```
São Paulo         12106920
Rio de Janeiro     6520266
Brasília           3039444
Salvador           2953986
Belo Horizonte     2523794
Curitiba           1908359
Recife             1633697
Porto Alegre       1484941
```

1.7 Exemplos

1.7.1 Quais as três cidades brasileiras mais populosas?

```
In [55]: cid2pop = {'Salvador': 2954000,
                    'Fortaleza': 2627482,
                    'Belo Horizonte': 2523794,
                    'São Paulo': 12106920,
                    'Brasília': 3039444,
                    'Rio de Janeiro': 6520266,
                    'Manaus': 2130264,
                    'Curitiba': 1908359,
                    'Recife': 1633697,
                    'Porto Alegre': 1484941
                   }
```

```
In [56]: from operator import itemgetter

         scid2pop = sorted(cid2pop.items(), key=itemgetter(1), reverse=True)
         scid2pop[:3]
```

```
Out[56]: [('São Paulo', 12106920), ('Rio de Janeiro', 6520266), ('Brasília', 3039444)]
```

1.7.2 Quais cidades brasileiras têm mais do que 2 milhões de habitantes?

```
In [57]: cid2pop = {'Salvador': 2954000,
                    'Fortaleza': 2627482,
                    'Belo Horizonte': 2523794,
```

```
                    'São Paulo': 12106920,
                    'Brasília': 3039444,
                    'Rio de Janeiro': 6520266,
                    'Manaus': 2130264,
                    'Curitiba': 1908359,
                    'Recife': 1633697,
                    'Porto Alegre': 1484941
                   }
```

```
In [60]: from operator import itemgetter

         grandes = [(rm, pop) for rm, pop in cid2pop.items() if pop > 2000000]
         grandes = sorted(grandes, key=itemgetter(1), reverse=True)
         grandes
```

```
Out[60]: [('São Paulo', 12106920),
          ('Rio de Janeiro', 6520266),
          ('Brasília', 3039444),
          ('Salvador', 2954000),
          ('Fortaleza', 2627482),
          ('Belo Horizonte', 2523794),
          ('Manaus', 2130264)]
```

1.7.3 Dentre as cidades brasileiras com mais de 1,5 milhões de habitantes, quais as duas com nome mais curto?

```
In [61]: cid2pop = {'São Paulo': 12106920, 'Rio de Janeiro': 6520266,
                    'Brasília': 3039444,   'Salvador': 2953986,
                    'Fortaleza': 2627482,  'Belo Horizonte': 2523794,
                    'Manaus': 2130264,     'Curitiba': 1908359,
                    'Recife': 1633697,     'Porto Alegre': 1484941,
                    'Goiânia': 1466105,    'Belém': 1452275,
                    'Guarulhos': 1349113,  'Campinas': 1182429,
                    'São Luís': 1091868,   'São Gonçalo': 1049826,
                    'Maceió': 1029129
                   }
```

```
In [62]: from operator import itemgetter

         tam_nomes = [(nome, pop, len(nome)) for nome, pop in cid2pop.items() if pop > 1500000]
         tam_nomes = sorted(tam_nomes, key=itemgetter(2))
         tam_nomes
         tam_nomes[:2]
```

```
Out[62]: [('Manaus', 2130264, 6),
          ('Recife', 1633697, 6),
          ('Brasília', 3039444, 8),
          ('Salvador', 2953986, 8),
          ('Curitiba', 1908359, 8),
```

```
          ('São Paulo', 12106920, 9),
          ('Fortaleza', 2627482, 9),
          ('Rio de Janeiro', 6520266, 14),
          ('Belo Horizonte', 2523794, 14)]
```

```
Out[62]: [('Manaus', 2130264, 6), ('Recife', 1633697, 6)]
```