

Dicionários e Tuplas

Dicionários



- Os dicionários são o tipo de dados mais poderoso em Python
- Os dicionários têm nomes diferentes nas várias linguagens
 - Arrays associativos- Perl / PHP
 - Propriedades ou Map ou HashMap - Java
 - Propriedades - C# / .Net

http://en.wikipedia.org/wiki/Associative_array

Dicionários

- As listas **indexam** cada valor com base na sua posição na lista
- Nos dicionários, **indexamos** as coisas com uma “etiqueta”
- **Dicionários** não possuem ordem como as listas

```
>>> mala = dict()
>>> mala['dinheiro'] = 12
>>> mala['doce'] = 3
>>> mala['lenços'] = 75
>>> print(mala)
{'dinheiro': 12, 'lenços': 75, 'doce': 3}
>>> print(mala['doce'])
3
>>> mala['doce'] = mala['doce'] + 2
>>> print(mala)
{'dinheiro': 12, 'lenços': 75, 'doce': 5}
```

Comparação de listas e dicionários

- Dicionários são como listas, no entanto usam chaves em vez de números para localizar valores

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['idade'] = 21
>>> ddd['curso'] = 182
>>> print(ddd)
{'curso': 182, 'idade': 21}
>>> ddd['idade'] = 23
>>> print(ddd)
{'curso': 182, 'idade': 23}
```

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['idade'] = 21
>>> ddd['curso'] = 182
>>> print(ddd)
{'curso': 182, 'idade': 21}
>>> ddd['idade'] = 23
>>> print(ddd)
{'curso': 182, 'idade': 23}
```

Lista

Índice	Valor
--------	-------

[0]

21

[1]

183

lst

Dicionário

Chave	Valor
-------	-------

['curso']

182

['idade']

21

ddd

Declaração de dicionários

- A declaração de dicionários é feita recorrendo a uma lista de pares

chave : valor

- Podemos criar um dicionário vazio usando chaves vazias

```
>>> jjj = { 'luis' : 1 , 'fred' : 42, 'ana' : 100}
>>> print(jjj)
{'ana' : 100, 'luis' : 1, 'fred' : 42}
>>> ooo = { }
>>> print(ooo)
{}
>>>
```

Nome mais comum?

marquard	cwen	cwen
zhen	marquard	zhen
csev	zhen	csev
zhen	csev	marquard
		zhen

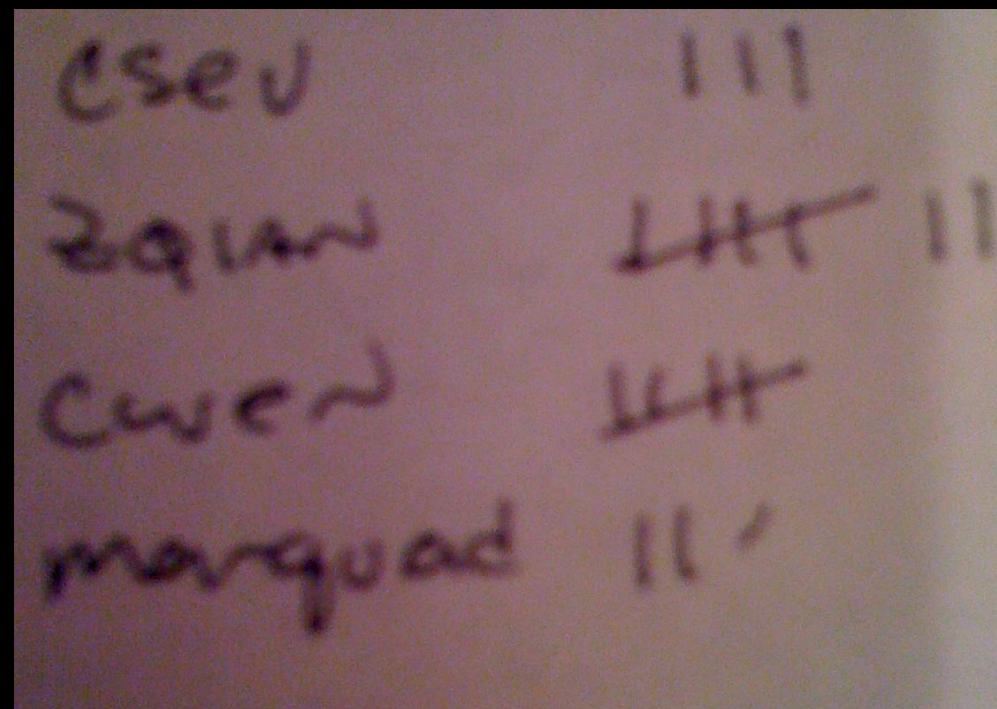
Nome mais comum?

marquard

cwen

cwen

zhen



zhen

csev

csev

zhen

csev

marquard

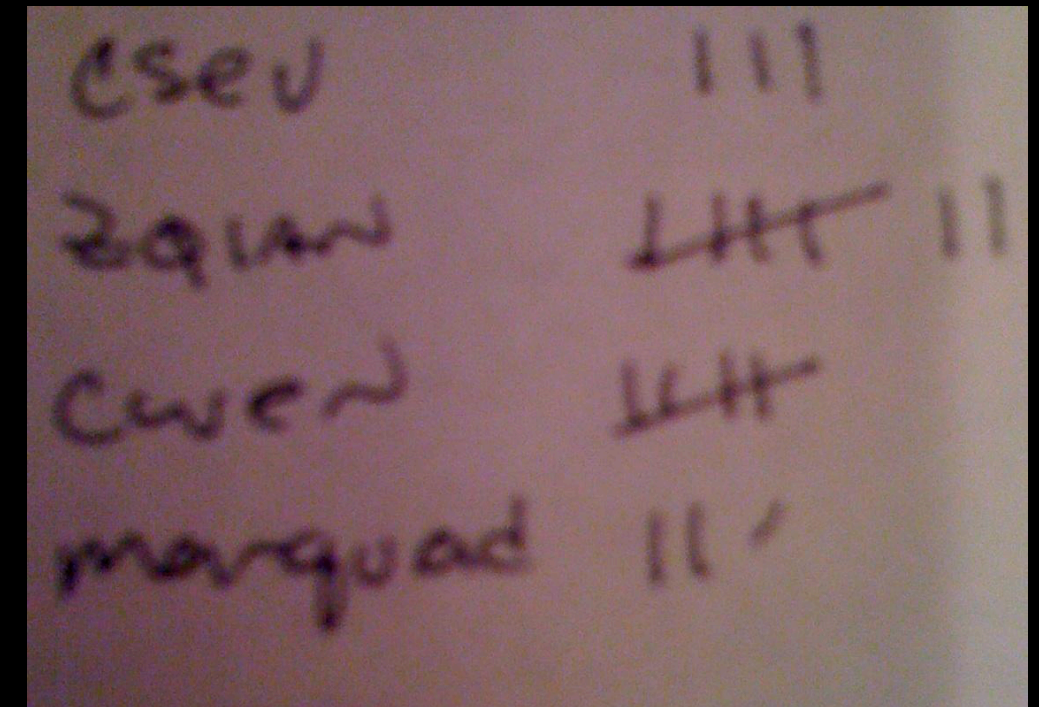
zhen

Múltiplos contadores com um dicionário

- Uma utilização frequente dos dicionários é **contar** o nº de ocorrências de determinada **coisa**

```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print(ccc)
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print(ccc)
{'csev': 1, 'cwen': 2}
```

Key Value



csev	111
cwen	1111
marquard	111

Erros em Dicionários

- É um **erro** tentar acessar o dicionário por referência a uma **chave** não existente no dicionário
- Podemos usar o operador **in** para verificar a existência de uma chave no dicionário

```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> print('csev' in ccc)
False
```

Encontrando um novo nome

- Quando é encontrado um novo nome, é necessário adicioná-lo ao **dicionário**; caso seja a segunda ou terceira ocorrência do **nome**, basta incrementar o contador no **dicionário** sob esse **nome**

```
conta = dict()
nomes = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for nome in nomes:
    if nome not in conta:
        conta[nome] = 1
    else:
        conta[nome] = conta[nome] + 1
print(conta)
```

O método `get` para dicionários

- A verificação da existência de uma `chave` num dicionário, assumindo um valor pré-definido, caso a `chave` não exista é tão comum, que existe o `método get()` que nos facilita este trabalho

```
if nome in conta:  
    x = conta[nome]  
else :  
    x = 0
```

```
x = conta.get(nome, 0)
```

Valor pré-definido caso a chave não exista (e sem erro).

```
{'csev': 2, 'zqian': 1, 'cwen': 2}
```

Contagem simplificada com `get()`

- Podemos usar o método `get()` e atribuir um **valor pré-definido de zero** quando a **chave** ainda não existe no dicionário - e de incrementar um a cada novo ocorrência

```
conta = dict()
nomes = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for nome in nomes:
    conta[nome] = conta.get(nome, 0) + 1
print(conta)
```

pré-definido

`{'csev': 2, 'zqian': 1, 'cwen': 2}`

Padrão de contagem

```
conta = dict()
print('Insira uma linha de texto:')
linha = input(' ')

palavras = linha.split()

print('Palavras:', palavras)

print('A contar...')
for palavra in palavras:
    conta[palavra] = conta.get(palavra, 0) + 1
print('Conta', conta)
```

O padrão geral para contar palavras numa linha de texto consiste em separar (**split**) a linha em palavras e em seguida percorrê-las usando um dicionário (**dictionary**) para contar o número de repetições de cada palavra.

```
conta = dict()
print('Insira uma linha de texto:')
linha = input('')
palavras = linha.split()

print('Palavras:', palavras)
print('A contar...')

for palavra in palavras:
    conta[palavra] = conta.get(palavra,0) + 1
print('Conta', conta)
```

python contapalavras.py

Insira uma linha de texto:
the clown ran after the car
and the car ran into the tent
and the tent fell down on the
clown and the car

```
Palavras: ['the', 'clown', 'ran',
'after', 'the', 'car', 'and', 'the',
'car', 'ran', 'into', 'the', 'tent',
'and', 'the', 'tent', 'fell', 'down',
'on', 'the', 'clown', 'and', 'the',
'car']
```

A contar...

```
Conta {'and': 3, 'on': 1, 'ran': 2,
'car': 3, 'into': 1, 'after': 1,
'clown': 2, 'down': 1, 'fell': 1,
'the': 7, 'tent': 2}
```

Laços e Dicionários

- Mesmo não sendo armazenados em ordem, podemos criar um laço **for** que percorra todas as **entradas** do **dicionário** - na verdade, percorre todas as **chaves** no **dicionário** e **apresenta** os seus valores

```
>>> conta = { 'chuck' : 1 , 'fred' : 42, 'jan' : 100}
>>> for chave in conta:
...     print(chave, conta[chave])
...
jan 100
chuck 1
fred 42
>>>
```


Obter listas de Chaves e Valores

- Podemos obter uma lista de **chaves**, **valores**, ou **items** (ambos) existentes num dicionário

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

O que é uma 'tupla'? - em breve veremos...



Bônus: duas variáveis de iteração!

- Percorrer os pares de **chave-valor** num dicionário usando ***duas*** variáveis de iteração
- A cada iteração, a primeira variável é a **chave** e a segunda é o **valor** correspondente a chave

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42,
            'jan': 100}
>>> for aaa, bbb in jjj.items() :
...     print(aaa, bbb)
...
jan 100
chuck 1
fred 42
>>>
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

```
nome = input('Indique arquivo:')  
handle = open(nome)  
texto = handle.read()  
palavras = texto.split()
```

```
conta = dict()  
for palavra in palavras:  
    conta[palavra] = conta.get(palavra, 0) + 1
```

```
maiorconta = None  
maiorpalavra = None  
for palavra, conta in conta.items():  
    if maiorconta is None or conta > maiorconta:  
        maiorpalavra = palavra  
        maiorconta = conta
```

```
print(maiorpalavra, maiorconta)
```

```
python palavras.py  
Indique arquivo: words.txt  
to 16
```

```
python palavras.py  
Indique arquivo: clown.txt  
the 7
```

Tuplas são como listas

- Tuplas são um tipo de variável cuja funcionalidade se assemelha às listas - eles contêm elementos que são indexados com início em 0.

```
>>> x = ('Guida', 'Susana', 'Jose')
>>> print(x[2])
Jose
>>> y = (1, 9, 2)
>>> print(y)
(1, 9, 2)
>>> print(max(y))
9
```

```
>>> for iter in y:
...     print(iter)
...
1
9
2
>>>
```

mas... Tuplas são “imutáveis”

- Ao contrário de uma lista, após criar um **tupla**, **não podemos alterar** o seu conteúdo - como numa string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>> [9, 8, 6]
>>>
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback: 'str'
object does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback: 'tuple'
object does
not support item
Assignment
>>>
```

O que não fazer com Tuplas

```
>>> x = (3, 2, 1)
```

```
>>> x.sort()
```

```
Traceback:
```

```
AttributeError: 'tuple' object has no attribute 'sort'
```

```
>>> x.append(5)
```

```
Traceback:
```

```
AttributeError: 'tuple' object has no attribute 'append'
```

```
>>> x.reverse()
```

```
Traceback:
```

```
AttributeError: 'tuple' object has no attribute 'reverse'
```

```
>>>
```

Listas e Tuplas - diferenças

```
>>> l = list()
>>> dir(l)
['append', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

```
>>> t = tuple()
>>> dir(t)
['count', 'index']
```

Tuplas são mais eficientes

- Uma vez que a estrutura das Tuplas não permite modificações (imutáveis), são estruturas mais simples e eficientes em termos de desempenho e gestão de memória do que as Listas
- Por este motivo, quando precisamos de “variáveis temporárias” preferimos Tuplas em vez de Listas

Atribuição a Tuplas

- Também podemos colocar uma tupla do lado esquerdo de uma instrução de atribuição
- Podemos inclusive omitir os parênteses

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> a, b = 99, 98
>>> print(a)
99
```

Tuplas e Dicionários

- O método `items()` em dicionários retorna uma lista de **tuplas** (chave, valor)

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for k,v in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
[('csev', 2), ('cwen', 4)]
```

Tuplas são Comparáveis

- Os operadores de comparação trabalham com tuplas e outras seqüências. Se o primeiro item for igual, Python passa para o próximo elemento, e assim por diante, até encontrar elementos que diferem.

```
>>> (0, 1, 2) < (5, 1, 2)
```

```
True
```

```
>>> (0, 1, 2000000) < (0, 3, 4)
```

```
True
```

```
>>> ( 'Jones', 'Sally' ) < ( 'Jones', 'Sam' )
```

```
True
```

```
>>> ( 'Jones', 'Sally' ) > ( 'Adams', 'Sam' )
```

```
True
```

Ordenando Listas de Tuplas

- Podemos aproveitar a capacidade de classificar uma lista de **tuplas** para obter uma versão ordenada dos elementos em um dicionário
- Primeiro ordenamos o conteúdo do dicionário pela chave usando o método **items()**

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = d.items()
>>> t
[('a', 10), ('c', 22), ('b', 1)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

Utilizando sorted()

Podemos fazer isso ainda mais diretamente usando a função integrada `sorted`, que recebe uma sequência como um parâmetro e retorna uma sequência ordenada

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> d.items()
[('a', 10), ('c', 22), ('b', 1)]
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

Ordenando pelos valores

- Se pudermos construir uma lista de **tuplas** da forma **(valor, chave)**, podemos ordenar por valor
- Fazemos isso com um loop **for** que cria uma lista de tuplas

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items():
...     tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp.sort(reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

```
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1

lst = list()
for key, val in counts.items():
    lst.append( (val, key) )

lst.sort(reverse=True)

for val, key in lst[:10] :
    print(key, val)
```

As 10 palavras
mais comuns

Retorno de Funções

- É possível retornar mais de um valor em uma função?
- É possível atribuir os valores de retorno a duas variáveis distintas?
- Sim! Usando Tuplas.

Extra: Ordenando pelos valores

- Podemos usar também a função `itemgetter` do módulo `operator`

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = sorted(c.items(),
key=operator.itemgetter(1),
reverse=True)
```

- Mantém a estrutura `(chave, valor)` do dicionário.

```
>>> print(tmp)
[('c', 22), ('a', 10), ('b', 1)]
```

Extra: Versão ainda mais curta

```
>>> c = {'a':10, 'b':1, 'c':22}
```

```
>>> print(sorted([(v,k) for k,v in c.items()]))
```

```
[(1, 'b'), (10, 'a'), (22, 'c')]
```

List comprehension cria uma lista dinâmica. Nesse caso, criamos uma lista de tuplas invertidas (**valor, chave**) e depois ordenamos.

<http://wiki.python.org/moin/HowTo/Sorting>

Os direitos de autor (2010) destes slides pertencem a Charles R. Severance (www.dr-chuck.com) da Escola de Informação da Universidade do Michigan, open.umich.edu e são disponibilizados sob Licença Creative Commons Attribution 4.0. Por favor mantenha este último slide em todas as cópias do documento de forma a cumprir com os requisitos de atribuição da licença. Caso faça alguma alteração, pode adicionar o seu nome e organização à lista de contribuições nesta página quando disponibilizar os conteúdos.

Desenvolvimento original: Charles Severance, Escola de Informação da Universidade do Michigan

Contribuições:

Luís Barreto (tradução para Português de Portugal)