

Python: Dicionários

Douglas Duarte

Dicionários

- São estruturas de dados que implementam *mapeamentos*
- Um mapeamento é uma coleção de associações entre pares de valores
 - O primeiro elemento do par é chamado de *chave* e o outro de *conteúdo*
- De certa forma, um mapeamento é uma generalização da ideia de acessar dados por índices, exceto que num mapeamento os índices (ou chaves) podem ser de qualquer tipo *imutável*

Chaves vs. Índices

- Considere que queiramos representar um caderno de telefones
 - Uma solução é ter uma lista de nomes e outra de telefones
 - Telefone de nome[i] armazenado em telefone[i]
 - Acrescentar “Joao” com telefone “20122232”:
nome+= “Joao” telefone+=“20122232”
 - Para encontrar o telefone de “Joao”:
Tel = telefone[nome.index[“Joao”]]
 - Dicionários tornam isso mais fácil e *eficiente*
telefone[“Joao”] = “20122232”
Tel = telefone[“Joao”]

Criando dicionários

- Uma constante do tipo dicionário é escrita
`{ chave1:conteúdo1, ... chaveN:conteúdoN }`
- Uma variável do tipo dicionário pode ser “indexada” da maneira habitual, isto é, usando colchetes
- O conteúdo associado a uma chave pode ser alterado atribuindo-se àquela posição do dicionário
- Novos valores podem ser acrescentados a um dicionário fazendo atribuição a uma chave ainda não definida
- Não há ordem definida entre os pares chave/conteúdo de um dicionário

Exemplo

```
>>> dic = {"joao":100,"maria":150}
>>> dic["joao"]
100
>>> dic["maria"]
150
>>> dic["pedro"] = 10
>>> dic
{'pedro': 10, 'joao': 100, 'maria': 150}
>>> dic = {'joao': 100, 'maria': 150, 'pedro': 10}
>>> dic
{'pedro': 10, 'joao': 100, 'maria': 150}
```

Dicionários não têm ordem

- As chaves dos dicionários não são armazenadas em qualquer ordem específica
 - Na verdade, dicionários são implementados por tabelas de espalhamento (*Hash Tables*)
 - A falta de ordem é proposital
- Diferentemente de listas, atribuir a um elemento de um dicionário não requer que a posição exista previamente

$X = []$

$X[10] = 5$ # ERRO!

...

$Y = \{ \}$

$Y[10] = 5$ # OK!

A função *dict*

- A função `dict` é usada para construir dicionários e requer como parâmetros:
 - Uma lista de tuplas, cada uma com um par chave/conteúdo, ou
 - Uma sequência de itens no formato *chave=valor*
 - Nesse caso, as chaves têm que ser strings, mas são escritas sem aspas

Exemplo

```
>>> d = dict([(1,2),('chave','conteudo')])
```

```
>>> d[1]
```

```
2
```

```
>>> d['chave']
```

```
'conteudo'
```

```
>>> d = dict(x=1,y=2)
```

```
>>> d['x']
```

```
1
```

```
>>> d = dict(1=2,3=4)
```

```
SyntaxError: keyword can't be an expression
```


Formatando com Dicionários

- O operador de formatação quando aplicado a dicionários requer que os valores das chaves apareçam entre parênteses antes do código de formatação
 - O conteúdo armazenado no dicionário sob aquela chave é substituído na string de formatação
 - Ex:

```
>>> dic = { "Joao":"a", "Maria":"b" }  
>>> s = "%(Joao)s e %(Maria)s"  
>>> s % dic  
'a e b'
```

Método *clear*

- `clear()`

- Remove todos os elementos do dicionário

- Ex.:

```
>>> x = { "Joao":"a", "Maria":"b" }
```

```
>>> y = x
```

```
>>> x.clear()
```

```
>>> print (x,y)
```

```
{ } { }
```

- Diferente de atribuir `{}` à variável:

```
>>> x = { "Joao":"a", "Maria":"b" }
```

```
>>> y = x
```

```
>>> x = { }
```

```
>>> print (x,y)
```

```
{ } {'Joao': 'a', 'Maria': 'b'}
```

Método *copy*

- `copy()`
 - Retorna um outro dicionário com os mesmos pares chave/conteúdo
 - Observe que os conteúdos não são cópias, mas apenas referências para os mesmos valores

```
>>> x = {"Joao":[1,2], "Maria":[3,4]}
>>> y = x.copy()
>>> y ["Pedro"]=[5,6]
>>> x ["Joao"] += [3]
>>> print (x)
{'Joao': [1, 2, 3], 'Maria': [3, 4]}
>>> print (y)
{'Pedro': [5, 6], 'Joao': [1, 2, 3], 'Maria': [3, 4]}
```

Método *fromkeys*

- `fromkeys(lista, valor)`
 - Retorna um novo dicionário cujas chaves são os elementos de *lista* e cujos valores são todos iguais a *valor*
 - Se *valor* não for especificado, o default é `None`

```
>>> {}.fromkeys([2,3])  
{2: None, 3: None}
```

```
# Podemos usar o nome da classe ao invés  
# de um objeto:
```

```
>>> dict.fromkeys(["Joao","Maria"],0)  
{'Joao': 0, 'Maria': 0}
```

Método *get*

- `get(chave, valor)`
 - Obtém o conteúdo de *chave*
 - Não causa erro caso chave não exista: retorna *valor*
 - Se *valor* não for especificado chaves inexistentes retornam `None`
 - Ex.:

```
>>> dic = { "Joao":"a", "Maria":"b" }
>>> dic.get("Pedro")
>>> print (dic.get("Pedro"))
None
>>> print (dic.get("Joao"))
a
>>> print (dic.get("Carlos","N/A"))
N/A
```

Método *has_key*

- `has_key(chave)`
 - `dic.has_key(chave)` é o mesmo que `chave in dic`
 - Ex.:

```
>>> dic = { "Joao":"a", "Maria":"b" }  
>>> dic.has_key("Joao")  
True  
>>> dic.has_key("Pedro")  
False
```

Métodos *items*, *keys* e *values*

- `items()` retorna uma lista com todos os pares chave/conteúdo do dicionário
- `keys()` retorna uma lista com todas as chaves do dicionário
- `values()` retorna uma lista com todos os valores do dicionário
- Ex.:

```
>>> dic.items()
[('Joao', 'a'), ('Maria', 'b')]
>>> dic.keys()
['Joao', 'Maria']
>>> dic.values()
['a', 'b']
```

Método *pop*

- `pop (chave)`
 - Obtém o valor correspondente a chave e remove o par chave/valor do dicionário
 - Ex.:

```
>>> d = {'x': 1, 'y': 2}
>>> d.pop('x')
1
>>> d
{'y': 2}
```


Método *popitem*

- `popitem()`
 - Retorna e remove um par chave/valor aleatório do dicionário
 - Pode ser usado para iterar sobre todos os elementos do dicionário
 - Ex:

```
>>> d
{'url': 'http://www.python.org', 'spam': 0, 'title': 'Python Web Site'}
>>> d.popitem()
('url', 'http://www.python.org')
>>> d
{'spam': 0, 'title': 'Python Web Site'}
```

Método *update*

- `update(dic)`
 - Atualiza um dicionário com os elementos de outro
 - Os itens em *dic* são adicionados um a um ao dicionário original
 - É possível usar a mesma sintaxe da função `dict` para especificar *dic*
 - Ex.:

```
>>> x = {"a":1,"b":2,"c":3}
>>> y = {"z":9,"b":7}
>>> x.update(y)
>>> x
{'a': 1, 'c': 3, 'b': 7, 'z': 9}
>>> x.update(a=7,c="xxx")
>>> x
{'a': 7, 'c': 'xxx', 'b': 7, 'z': 9}
```