

ASIMOV

Dicionários

Nós temos aprendido sobre *sequências* em Python, mas agora vamos mudar de engrenagem e aprender sobre *mapeamentos* em Python. Se você está familiarizado com outras linguagens, pode pensar nestes Dicionários como tabelas de hash.

Esta seção servirá como uma breve introdução aos dicionários e consiste em:

1.) Construindo um Dicionário

2.) Acessando objetos de um dicionário 3.) Dicionários de assentamento 4.) Métodos básicos do dicionário

Então, o que são os mapeamentos? Os mapeamentos são uma coleção de objetos que são armazenados por uma *chave*, ao contrário de uma sequência que armazena objetos por sua posição relativa. Esta é uma distinção importante, uma vez que os mapeamentos não reterão a ordem, pois possuem objetos definidos por uma chave.

Um dicionário de Python consiste em uma chave e depois em um valor associado. Esse valor pode ser quase qualquer objeto Python.

Construindo um Dicionário

Vamos ver como podemos construir dicionários para obter uma melhor compreensão de como eles funcionam!

```
In [1]: # Cria um dicionário com {} e: que significa uma chave e um valor
my_dict = {'key1': 'value1', 'key2': 'value2'}
```

```
In [2]: # Chamando valores pela chave
my_dict['key2']
```

```
Out[2]: 'value2'
```

É importante notar que os dicionários são muito flexíveis com relação aos tipos de dados que eles podem conter. Por exemplo:

```
In [13]: my_dict = {'key1': 123, 'key2': [12, 23, 33], 'key3': ['item0', 'item1', 'item2']}
```

```
In [4]: # Vamos chamar itens do dicionário
my_dict['key3']
```

Out[4]: ['item0', 'item1', 'item2']

```
In [5]: # Podemos chamar itens de uma lista presente na posição referente à chave 'key3'  
my_dict['key3'][0]
```

Out[5]: 'item0'

```
In [7]: # Podemos chamar métodos nos itens também  
my_dict['key3'][0].upper()
```

Out[7]: 'ITEM0'

Podemos também alterar valores através da chave.

```
In [14]: my_dict['key1']
```

Out[14]: 123

```
In [15]: my_dict['key1'] = my_dict['key1'] - 123
```

```
In [16]: my_dict['key1']
```

Out[16]: 0

Uma nota rápida: o Python possui um método interno de fazer uma subtração ou adição automática (ou multiplicação ou divisão). Poderíamos ter usado += ou -= para a atribuição. Por exemplo:

```
In [17]: # Define o objeto como sendo ele mesmo menos 123  
my_dict['key1'] -= 123  
my_dict['key1']
```

Out[17]: -123

Também podemos criar chaves por atribuição. Por exemplo, se começássemos com um dicionário vazio, poderíamos adicionar-lhe continuamente:

```
In [21]: # Cria um novo dicionário  
d = {}
```

```
In [22]: # Cria uma chave por associação  
d['animal'] = 'Dog'
```

```
In [24]: # Pode fazer isso com qualquer objeto  
d['answer'] = 42
```

```
In [25]: # Mostra  
d
```

Out[25]: {'animal': 'Dog', 'answer': 42}

Aninhamento de dicionários

Espero que você esteja começando a ver o quão poderoso Python é com sua flexibilidade de objetos de nidificação e métodos dos mesmos. Vamos ver um dicionário aninhado dentro de um dicionário:

```
In [26]: d = {'key1':{'nestkey':{'subnestkey':'value'}}}
```

```
In [29]: # Continue chamando as chaves...  
d['key1']['nestkey']['subnestkey']
```

```
Out[29]: 'value'
```

Alguns métodos de dicionários

Existem alguns métodos que podemos chamar em um dicionário. Vamos começar uma breve introdução a alguns deles:

```
In [30]: # Cria um dicionário típico  
d = {'key1':1, 'key2':2, 'key3':3}
```

```
In [35]: # Retorna uma lista de todas as chaves  
d.keys()
```

```
Out[35]: ['key3', 'key2', 'key1']
```

```
In [36]: # Pega todos os valores  
d.values()
```

```
Out[36]: [3, 2, 1]
```

```
In [33]: # Método para retornar as tuplas de todos os itens (aprenderemos sobre as tuplas)  
d.items()
```

```
Out[33]: [('key3', 3), ('key2', 2), ('key1', 1)]
```

Espero que você tenha agora um bom entendimento básico para a construção de dicionários. Há muito mais para explorar aqui, mas vamos revisar os dicionários mais tarde. Depois desta seção, tudo o que você precisa saber é como criar um dicionário e como recuperar seus valores.