Introdução aos dicionários do Python

4 minutos

As variáveis do Python podem armazenar vários tipos de dados. Anteriormente, você aprendeu que pode armazenar cadeias de caracteres e números:

Python

```
name = 'Earth'
moons = 1
```

Embora esse método funcione para quantidades menores de dados, ele pode ficar cada vez mais complexo em trabalhos que usam dados relacionados. Imagine que você quer armazenar informações sobre as luas da Terra e as luas de Júpiter.

Python

```
earth_name = 'Earth'
earth_moons = 1

jupiter_name = 'Jupiter'
jupiter_moons = 79
```

Observe como duplicamos variáveis com diferentes prefixos. Essa duplicação pode ficar complexa. Por ser comum trabalhar com conjuntos de dados relacionados, como o volume médio de chuva em diferentes meses em diferentes cidades, o armazenamento dessas variáveis como valores individuais não é uma opção viável. Nesse caso, você pode usar dicionários Python.

Os dicionários do Python permitem que você trabalhe com conjuntos de dados relacionados. Um *dicionário* é uma coleção de pares chave/valor. Pense nele como um grupo de variáveis dentro de um contêiner, em que a chave é o nome da variável e o valor é o valor armazenado dentro dela.

Criar um dicionário

O Python usa chaves (`{ }`) e dois-pontos (`:`) para denotar um dicionário. Você pode criar um dicionário vazio e adicionar valores posteriormente ou populá-lo no momento da criação. Cada chave/valor é separado por dois-pontos e o nome de cada chave está contido entre

aspas como um literal de cadeia de caracteres. Como a chave é um literal de cadeia de caracteres, você pode usar qualquer nome apropriado para descrever o valor.

Vamos criar um dicionário para armazenar o nome do planeta Terra e o número de luas que a Terra tem:

Python

```
planet = {
    'name': 'Earth',
    'moons': 1
}
```

Temos duas chaves, `'name'` e `'moons'`. Cada chave se comporta de modo muito semelhante a uma variável: elas têm um nome exclusivo e armazenam um valor. No entanto, elas estão contidas dentro de uma variável maior, chamada `planet`.

Assim como com variáveis regulares, você precisa garantir o uso dos tipos de dados corretos. No valor `moons` de `1` no exemplo anterior, você não incluiu aspas em torno do número, pois deseja usar um inteiro. Se você tivesse usado `'1'`, o Python entenderia essa variável como uma cadeia de caracteres, o que afetaria sua capacidade de executar cálculos.

Ao contrário das variáveis regulares, os nomes de chave *não* precisam seguir as regras de nomenclatura padrão do Python. Você pode nomes de chave para melhorar a descrição no código.

Ler valores do dicionário

Você pode ler valores dentro de um dicionário. Os objetos de dicionário têm um método `get` que você pode usar para acessar um valor usando a chave correspondente. Se você quiser imprimir o `name`, poderá usar o seguinte código:

Python

```
print(planet.get('name'))
```

Output

```
Earth
```

Como seria de se esperar, o acesso aos valores de um dicionário é uma operação comum. Felizmente, há um atalho. Você também pode passar a chave na notação de colchete (`[ ]`).

Esse método usa menos código do que `get` e a maioria dos programadores usa essa sintaxe. Você pode reescrever o exemplo anterior usando o seguinte código:

Python

```
# planet['name'] is identical to using planet.get('name')
print(planet['name'])
```

Output

```
Earth
```

Embora o comportamento de `get` e dos colchetes (`[ ]`) geralmente seja o mesmo para recuperar itens, há uma diferença importante. Se a chave não estiver disponível, `get` retornará `None` e `[ ]` vai gerar um `KeyError`.

Python

```
wibble = planet.get('wibble') # Returns None
wibble = planet['wibble'] # Throws KeyError
```

Modificar valores do dicionário

Você também pode modificar valores dentro de um objeto de dicionário usando o método `update`. Esse método aceita um dicionário como um parâmetro e atualiza todos os valores existentes com os novos que você fornecer. Se quiser alterar `name` para o dicionário `planet`, você poderá usar o seguinte código, por exemplo:

Python

```
planet.update({'name': 'Makemake'})

# No output: name is now set to Makemake.
```

Semelhante ao uso do atalho de colchetes (`[ ]`) para ler valores, você pode usar o mesmo atalho para modificar valores. A principal diferença na sintaxe é que você usa `=` (às vezes chamado de operador de *atribuição*) para fornecer um novo valor. Para reescrever o exemplo anterior para alterar o nome, você pode usar o seguinte código:

Python

```
planet['name'] = 'Makemake'

# No output: name is now set to Makemake.
```

A principal vantagem de usar `update` é a capacidade de modificar vários valores em apenas uma operação. Os dois exemplos a seguir são logicamente os mesmos, mas a sintaxe é diferente. Você é livre para usar a sintaxe que achar mais apropriada. A maioria dos desenvolvedores prefere colchetes para atualizar valores individuais.

O exemplo a seguir faz as mesmas edições na variável `planet`, atualizando o nome e as luas. Observe que, usando `update`, você está fazendo apenas uma chamada à função, enquanto o uso de colchetes envolve duas chamadas.

Usando update:

Python

```
planet.update({
    'name': 'Jupiter',
    'moons': 79
})
```

Usando colchetes:

Python

```
planet['name'] = 'Jupiter'
planet['moons'] = 79
```

Adicionar e remover chaves

Você não precisa criar todas as chaves ao inicializar um dicionário. Na verdade, você não precisa criar nenhuma, Sempre que você quiser criar uma chave, atribua ela da mesma forma que faria com uma chave existente.

Digamos que você queira atualizar `planet` para incluir o período orbital em dias:

Python

```
planet['orbital period'] = 4333

# planet dictionary now contains: {
#   name: 'jupiter'
#   moons: 79
```

```
#   orbital period: 4333
# }
```

> **Importante**
>
> Os nomes de chave, como todo o resto em Python, diferenciam maiúsculas de minúsculas. Como resultado, `'name'` e `'Name'` são vistos como duas chaves separadas em um dicionário do Python.

Para remover uma chave, use `pop`. `pop` retorna o valor e remove a chave do dicionário. Para remover `orbital period`, você pode usar o seguinte código:

Python

```
planet.pop('orbital period')

# planet dictionary now contains: {
#   name: 'jupiter'
#   moons: 79
# }
```

Tipos dados complexos

Os dicionários são capazes de armazenar qualquer tipo de valor, incluindo outros dicionários. Isso permite que você modele dados complexos conforme necessário. Imagine se precisar armazenar o diâmetro de `planet`, que pode ser medido ao redor da linha do de ou nos polos. Você pode criar outro dicionário dentro de `planet` para armazenar essas informações:

Python

```
# Add address
planet['diameter (km)'] = {
    'polar': 133709,
    'equatorial': 142984
}

# planet dictionary now contains: {
#   name: 'Jupiter'
#   moons: 79
#   diameter (km): {
#      polar: 133709
#      equatorial: 142984
#   }
# }
```

Para recuperar valores em um dicionário aninhado, você deve encadear colchetes ou chamadas a `get`.

Python

```
print(f'{planet["name"]} polar diameter: {planet["diameter (km)"]["polar"]}')
```

Output

```
Jupiter polar diameter: 133709
```

Unidade seguinte: Exercício – Criar dicionários do Python

Continuar >