09

Para saber mais: Formatação de strings

Transcrição

Segue o link da documentação que mencionei no video, nele tem vários exemplos de formatação:

https://docs.python.org/3/library/string.html#formatexamples
(https://docs.python.org/3/library/string.html#formatexamples)

Nesse vídeo veremos um pouco mais sobre interpolação de strings. Para isso, vamos utilizar o console do Python 3.

No capítulo anterior, fizemos uma interpolação semelhante a essa:

```
>>> print("Tentativa {} de {}".format(1, 3))
```

Essa interpolação é útil para formatação de strings, quando temos um texto muito grande e precisamos inserir valores no meio dele, ao invés de ficarmos concatenando, trabalhando com várias strings separadas.

Mas a função `format` tem outras utilidades, então veremos mais alguns detalhes sobre essa função. O primeiro detalhe que veremos é que os parâmetros podem ser invertidos na string. Podemos dizer que queremos nas primeiras chaves o segundo parâmetro da função, e o primeiro parâmetro nas segundas chaves.

Fazemos isso passando o **índice do parâmetro** dentro das chaves. O primeiro parâmetro tem índice **0**, o segundo **1**, e daí por diante. Logo, basta passar o índice 1 nas primeiras chaves e o 0 nas segundas chaves:

```
>>> print("Tentativa {1} de {0}".format(1, 3))
Tentativa 3 de 1
```

Formatação de floats

Agora vamos trocar o exemplo, e formatar um valor em reais, por exemplo:

```
>>> print("R$ {}".format(1.59))
R$ 1.59
```

Só que um valor pode ter vários tamanhos e até duas casas decimais, por exemplo:

```
1.59
45.9
1234.97
```

O ideal é que esses valores sempre tenham a mesma formatação:

```
  1.59
 45.9
```

```
1234.97
```

Então precisamos preencher as lacunas, os espaços em branco. E a função `format` faz isso para nós. Primeiro precisamos dizer para ela que estamos recebendo um valor do tipo float, passando `:f` dentro das chaves da string:

```
>>> print("R$ {:f}".format(1.59))
R$ 1.590000
```

Podemos reparar que só de dizer que estamos passando um float, a formatação já muda, mas podemos manipulá-la, modificá-la, dizendo quantos números devem vir antes e depois do ponto. Queremos que após o ponto tenha apenas 2 números, logo:

```
>>> print("R$ {:.2f}".format(1.59))
R$ 1.59
```

Podemos testar passando um número de apenas uma casa decimal:

```
>>> print("R$ {:.2f}".format(1.5))
R$ 1.50
```

Ótimo, agora vamos testar com um número maior:

```
>>> print("R$ {:.2f}".format(1.5))
R$ 1.50
>>> print("R$ {:.2f}".format(1234.50))
R$ 1234.50
```

Mas queremos que o ponto fique sempre no mesmo local, ou seja, ele deve ser o quinto caractere. Para essa formatação, precisamos dizer quantos caracteres o número terá no máximo, no nosso caso são 7 (4 números, mais o ponto, mais as duas casas decimais). Então vamos passar o valor 7 dentro das chaves também:

```
>>> print("R$ {:7.2f}".format(1234.50))
R$ 1234.50
>>> print("R$ {:7.2f}".format(1.5))
R$    1.50
```

Ou seja, dos 7 caracteres, os três últimos serão o ponto mais dois números das casas decimais.

Agora espaços ficam na frente quando um número for menor! Deixando o ponto sempre como quinto caractere. Se quisermos preencher os espaços em branco com zeros, é só passar um 0 antes do 7:

```
>>> print("R$ {:07.2f}".format(1.5))
R$ 0001.50
```

Formatação de inteiros

Conseguimos formatar números inteiros também, não só números flutuantes. Para números inteiros, passamos a letra `d`:

```
>>> print("R$ {:07d}".format(4))
R$ 0000004
```

Podemos usar isso para formatar uma data:

```
>>> print("Data {:02d}/{:02d}".format(9, 4))
Data 09/04
>>> print("Data {:02d}/{:02d}".format(19, 11))
Data 19/11
```

Não se preocupe em decorar a sintaxe, o importante é saber que no Python existe a funcionalidade de interpolação de strings, e quando vocês realmente precisarem usar isso, olhem na documentação (https://docs.python.org/3/library/string.html#formatexamples).