09

Para saber mais: Parâmetros opcionais e nomeados

Parâmetros opcionais

Já sabemos como definir e agrupar comportamentos dentro de funções. E que uma função pode ter retorno e receber ou não parâmetros. Praticamos bastante essas sintaxes nesse capítulo mas até agora não tínhamos nenhuma novidade em relação a outras linguagens de programação.

Acontece que o Python possui algumas facilidades que não estão presentes em todas as outras linguagens.

Ficou curioso? Aposto que sim! Python é uma linguagem viciante!

Relembre o começo da função `carrega_palavra_secreta` desenvolvida no nosso projeto:

```
def carrega_palavra_secreta():
    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))
    palavra_secreta = palavras[numero].upper()

    return palavra_secreta
```

Essa função como o próprio nome diz carrega uma palavra secreta das contidas em um arquivo chamado de `palavras.txt` .

E se quiséssemos deixar essa função um pouco mais flexível permitindo que fosse passado o nome do arquivo usado para o carregamento? Permitindo assim as seguintes chamadas:

```
carrega_palavra_secreta("frutas.txt")
carrega_palavra_secreta("nomes.txt")
```

Para que a chamada funcione basta declararmos que a função recebe um parâmetro com o nome do arquivo, além usá-lo no carregamento. Como a seguir:

```
def carrega_palavra_secreta(nome_arquivo):
    arquivo = open(nome_arquivo, "r")
   ...
```

E se essa função já estivesse sendo chamada em outros lugares sem o parâmetro:

```
palavra_secreta = carrega_palavra_secreta()
```

O código anterior deixaria de funcionar, já que houve uma obrigatoriedade de passarmos um nome de arquivo na definição da função. Se fosse necessário manter as duas chamadas válidas:

```
carrega_palavra_secreta("frutas.txt")
carrega_palavra_secreta()
```

Poderíamos definir que o parâmetro é opcional, ou seja, podemos ou não querer passar o nome do arquivo.

Para isso precisamos definir um nome de arquivo que seria o padrão usado quando não fosse especificado algum arquivo.

Esse arquivo padrão vai ser o `palavras.txt` usado anteriormente. O nosso código ficaria como a seguir:

```
def carrega_palavra_secreta(nome_arquivo="palavras.txt"):
    arquivo = open(nome_arquivo, "r")
   ...
```

Logo quando temos:

```
carrega_palavra_secreta()
```

O arquivo carregado será o `palavras.txt` . Por outro lado quando o parâmetro é especificado como em:

```
carrega_palavra_secreta("frutas.txt")
```

O arquivo carregado será o passado: `frutas.txt` .

Isso só é possível porque no Python temos como definir um valor padrão para os parâmetros e assim permitindo os **`parâmetros opcionais`** .

Parâmetros nomeados

Analisando a função completa podemos ver que o número gerado é sempre de `0` até o número de palavras do arquivo.

```
def carrega_palavra_secreta(nome_arquivo="palavras.txt"):
    arquivo = open(nome_arquivo, "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))
```

```
    palavra_secreta = palavras[numero].upper()

    return palavra_secreta
```

E se quiséssemos permitir a definição **de a** partir de qual linha deveria ser gerado o número? Uma mudança poderia ser feita na função de modo que ela recebesse essa indicação:

```
def carrega_palavra_secreta(primeira_linha_valida, nome_arquivo="palavras.txt"):
    arquivo = open(nome_arquivo, "r")
    palavras = []

    ….

    numero = random.randrange(primeira_linha_valida, len(palavras))
    palavra_secreta = palavras[numero].upper()

    return palavra_secreta
```

Poderíamos inclusive definir o valor padrão do parâmetro como sendo zero:

```
def carrega_palavra_secreta(nome_arquivo="palavras.txt", primeira_linha_valida=0):
    arquivo = open(nome_arquivo, "r")
    palavras = []

    ….

    numero = random.randrange(primeira_linha_valida, len(palavras))
    palavra_secreta = palavras[numero].upper()

    return palavra_secreta
```

As seguintes utilizações da função são válidas e não geram erros na execução:

```
carrega_palavra_secreta()
carrega_palavra_secreta("frutas.txt")
```

Onde na primeira chamada temos a utilização do valor padrão nos dois parâmetros, ou seja, usa-se o arquivo `palavras.txt` e a primeira linha válida como sendo `0`.

A segunda chamada já temos a utilização do arquivo `frutas.txt` e a primeira linha válida como sendo a `0`.

Para deixar mais claro que o valor passado refere-se ao nome do arquivo e não a primeira linha válida, podíamos inclusive repetir o nome do parâmetro na própria chamada:

```
carrega_palavra_secreta(nome_arquivo="frutas.txt")
```

Esse recurso é possível pois no Python podemos nomear os parâmetros passados.

É importante saber que não é obrigatório para o correto funcionamento do código pois quando temos mais de um parâmetro opcional na definição da função, a chamada omitindo algum deles vai usar sempre a ordem de definição.

Em outras palavras, fazendo:

```
carrega_palavra_secreta(5)
```

Ao contrário do que possa parecer ele não define o parâmetro `primeira_linha_valida` como `5` e sim o `nome_arquivo` como sendo `5`, o que é péssimo para o correto funcionamento da função. Logo, aqui percebe-se uma necessidade real da utilização do recurso de nomeação dos parâmetros. Ficando correta tanto na sintaxe quando na execução a seguinte chamada:

```
carrega_palavra_secreta(primeira_linha_valida=5)
```

Já na seguinte abordagem:

```
carrega_palavra_secreta("frutas.txt", 5)
```

Temos a utilização do arquivo `frutas.txt` e a primeira linha válida como sendo a quinta.

Poderíamos inclusive nomeá-los:

```
carrega_palavra_secreta(nome_arquivo="frutas.txt", primeira_linha_valida= 5)
```

E até mesmo trocar a ordem, já que estão nomeados:

```
carrega_palavra_secreta(primeira_linha_valida=5, nome_arquivo="frutas.txt")
```

Parâmetros opcionais e nomeados são recursos bastante poderosos para um código mais flexível e legível. Nem todas as linguagens possuem isso, aproveite o seu Python!