02

Funções

Introdução

Aprendemos nos últimos capítulos a lidar com os tipos de variáveis do Python. Neste capítulo aprenderemos a criar funções.

Temos o seguinte convite:

```
>>> convite = 'Romulo Henrique'
```

Lembra o problema hipotético no primeira aula quando o nome do perfil era grande demais para aparecer no convite da rede social? Pois bem, nosso cliente também hipotético (cliente de verdade só quando terminar o curso, né?) nos disse que o nome do perfil que aparecerá no convite serão os quatro primeiros caracteres do nome seguido de espaço e por fim os quatro últimos caracteres. Fácil, sabemos que através do operador slice podemos extrair fatias de uma string:

```
>>> parte1 = convite[0:4]
Romu
>>> parte2 = convite[11:15]
ique
```

E por fim precisamos imprimir no console o resultado da concatenação de ambos. Que tal usarmos a função print que já vimos?

```
>>> print "%s %s" % (parte1, parte2)
Romu ique
```

O que aconteceria se aplicássemos este mesmo código, mas com outra string:

```
>>> convite = 'Flavio Henrique Almeida'
>>> parte1 = convite[0:4]
>>> parte2 = convite[11:15]
>>> print "%s %s" % (parte1, parte2)
Flav ique
```

Repare que o resultado não é o esperado. Precisamos ter um código mais genérico. Qual tal se pudéssemos obter o tamanho da string? Daí poderíamos ajustar nosso slice do final da string com base neste valor. Para isso e várias outras tarefas, o Python disponibiliza uma série de funções. Um código guardado que pode ser chamado quando quisermos. Por exemplo, podemos utilizar a função `len` que recebe como parâmetro uma lista e retorna seu tamanho:

```
>>> convite = 'Flavio Henrique Almeida'
>>> tamanho = len(convite)
>>> parte1 = convite[0:4]
>>> parte2 = convite[tamanho-4:tamanho]
>>> print "%s %s" % (parte1, parte2)
Flav eida
```

Podemos deixar ainda mais legível nosso código declarando as variáveis `posicao_inicial` e `posicao_final`:

```
>>> convite = 'Flavio Henrique Almeida'
>>> posicao_final = len(convite)
>>> posicao_inicial = posicao_final - 4
>>> parte1 = convite[0:4]
>>> parte2 = convite[posicao_inicial:posicao_final]
>>> print "%s %s" % (parte1, parte2)
Flav eida
```

Código reutilizável através de funções

Pronto, mas imagine reescrever esse código toda vez que precisarmos repetir esta **funci**onalidade. Gostou da ideia? Nem eu! Será que existe alguma forma de guardarmos um código para podermos chamar mais tarde? Sim, tanto isso é verdade que utilizamos em nosso código a função **len**, porém ela já foi criada e disponibilizada pelo próprio Python.

O primeiro passo para **def**inirmos uma função que poderá ser reutilizada por qualquer desenvolvedor é criarmos um arquivo com a extensão `.py`. Por uma questão de organização, criaremos a pasta `python` e dentro dela o arquivo `biblioteca.py`. É neste arquivo que **def**iniremos nossa função.

DICA: cada fragmento de código a partir de agora começará com um comentário indicando a qual arquivo pertence. Isso o ajudará a saber rapidamente a qual arquivo o código pertence durante o treinamento

Você é bom em inglês? Como escrevemos o verbo `definir` em inglês? Vou dar uma cola: **define**. Agora, advinha a palavra chave utilizada para **def**inirmos nossa funções? Se você chutou define, quase acertou, mas o pessoal do mundo da programação não gosta muito de digitar, por isso usamos a palavra chave **def** seguido do nome da função terminando com dois pontos:

```
# python/biblioteca.py

def gera_nome_convite():
```

Veja que utilizamos um nome expressivo para a função, ou você prefere o nome ger_inf_cr? Cruzes! Excelente, criamos uma função que nada faz! Vamos copiar o código que fizemos agora a pouco para dentro da função:

```
# python/biblioteca.py

def gera_nome_convite():
  convite = 'Flavio Henrique Almeida'
  posicao_final = len(convite)
  posicao_inicial = posicao_final - 4
  parte1 = convite[0:4]
  parte2 = convite[posicao_inicial:posicao_final]
  print "%s %s" % (parte1, parte2)
```

Muito cuidado com a indentação!. Diferente de outras linguagens na qual um bloco de código é delimitado por caracteres especiais como `{` e `begin`, no Python blocos são delimitados por espaços ou tabulações formando uma **indentação visual**. Se não utilizarmos uma indentação padronizada, nosso código será rejeitado pelo interpretador do Python. Editores de texto como o Sublime nos auxiliam nesta tarefa. Por exemplo, este código indentado desta forma é considerado inválido pelo interpretador:

```
def gera_nome_convite():
convite = 'Flavio Henrique Almeida'
posicao_final = len(convite)
posicao_inicial = posicao_final - 4
parte1 = convite[0:4]
parte2 = convite[posicao_inicial:posicao_final]
print "%s %s" % (parte1, parte2)
```

Agora que temos uma ideia do que não devemos fazer, vamos voltar com a indentação válida de nosso código:

```
def gera_nome_convite():
  convite = 'Flavio Henrique Almeida'
  posicao_final = len(convite)
  posicao_inicial = posicao_final - 4
  parte1 = convite[0:4]
  parte2 = convite[posicao_inicial:posicao_final]
  print "%s %s" % (parte1, parte2)
```

Importar funções

E agora? Como faremos para executar esse código? Precisamos importá-lo em nosso terminal. Para isso, execute o comando `python` **dentro da pasta python que criamos**. Este passo é importantíssimo, caso contrário nosso terminal não conseguirá encontrar nosso arquivo.

Com o terminal aberto, vamos importar o arquivo `biblioteca.py` . Se o interpretador do Python entendesse português, poderíamos dizer para ele `da biblioteca importe tudo` . Porém, pela hegemonia da língua inglesa, precisamos traduzir nossa sentença para o inglês, com a única diferença de que omitimos a extensão do arquivo `.py` e trocamos a palavra `tudo` por asterisco:

```
>>> from biblioteca import *
```

Excelente. Se tudo ocorreu bem, já podemos chamar nossa função através do seu nome seguido de `()` :

```
>>> gera_nome_convite()
Flav eida
```

Parâmetros de funções

Muito bem! Definimos nossa função dentro do arquivo `biblioteca.py` , inclusive conseguimos importá-lo no próprio console do Python. Só uma pergunta: você se lembra da razão de termos feito isso? Queremos criar um código que seja reaproveitável e com certeza nosso código ainda não é. O valor da variável convite está fixo dentro da função. Precisamos que o nome do convite seja passado para a função, algo assim:

```
>>> gera_nome_convite('Flavio Almeida')
>>> gera_nome_convite('Romulo Henrique')
```

Para que isso seja possível, nossa função precisa receber um parâmetro. Vamos voltar para nosso arquivo `biblioteca.py` e mudar a linha que define seu nome para:

```
def gera_nome_convite(convite):
  convite = 'Flavio Henrique Almeida'
  posicao_final = len(convite)
  posicao_inicial = posicao_final - 4
  parte1 = convite[0:4]
  parte2 = convite[ posicao_inicial:posicao_final]
  print "%s %s" % (parte1, parte2)
```

Estamos indicando que a nossa função está preparada para receber um valor que será guardado na variável convite, sendo assim, podemos apagar a linha na qual atribuímos um valor para esta variável, já que ele será passado como parâmetro para quem chamar a função:

```
def gera_nome_convite(convite):
  posicao_final = len(convite)
  posicao_inicial = posicao_final - 4
  parte1 = convite[0:4]
  parte2 = convite[ posicao_inicial:posicao_final]
  print "%s %s" % (parte1, parte2)
```

Agora, vamos voltar para o console do Python. Precisamos fechá-lo e abri-lo novamente para carregarmos nosso arquivo recém modificado. Não se preocupe, mais tarde veremos uma maneira mais eficaz de lidar com atualizações de arquivos:

```
>>> from biblioteca import *
```

Agora, vamos chamar a nossa função passando um parâmetro:

```
>>> gera_nome_convite('Flavio Almeida')
Flav eida
>>> gera_nome_convite('Romulo Henrique')
Romu ique
```

Excelente! Conseguimos que a nossa função `gera_nome_convite` receba como parâmetro o dado do convite que desejamos gerar a informação do convite! Quando nossa função for executada, internamente ela guardará como valor da variável `convite` a string que passamos como parâmetro.

Espere um minuto. E se quisermos obter o resultado da função para associá-lo com texto "O código gerado foi %s". Não podemos simplesmente alterar nosso arquivo `biblioteca.py` e adicionar esse texto. Os utilizadores da nossa função esperam apenas como **retorno** a informação do convite. Precisamos fazer com que a nossa função, no lugar de imprimir no console o resultado, retorne o resultado da operação.

Vamos voltar para nosso arquivo `biblioteca.py` e realizar a seguinte alteração:

```
def gera_nome_convite(convite):
  posicao_final = len(convite)
  posicao_inicial = posicao_final - 4
  parte1 = convite[0:4]
```

```
    parte2 = convite[ posicao_inicial:posicao_final]
    return parte1 + ' ' + parte2
```

Estamos retornando a concatenação das duas string!

Não esqueça de fechar e abrir o terminal, importar nosso arquivo `biblioteca.py` mais uma vez para testar o resultado (não se desespere, melhoraremos isso em breve):

```
>>> info = gera_nome_convite('Flavio Almeida')
>>> print 'O código gerado foi %s' % (info)
O código gerado foi Flav eida
```

Bem, aprendemos a criar uma função que recebe e devolve um valor em seu próprio arquivo, garantindo assim o reuso de código. Que tal praticarmos agora com os exercícios?