

ASIMOV

Funções

Introdução às Funções

Esta aula consistirá em explicar o que é uma função em Python e como criar uma. As funções serão um dos nossos principais blocos de construção quando construiremos quantidades maiores e maiores de código para resolver problemas.

Então, o que é uma função?

Formalmente, uma função é um dispositivo útil que agrupa um conjunto de instruções para que elas possam ser executadas mais de uma vez. Eles também podem nos permitir especificar parâmetros que possam servir como entradas para as funções.

Em um nível mais fundamental, as funções nos permitem não ter que repetidamente escrever o mesmo código repetidas vezes. Se você lembrar de volta às lições em strings e listas, lembre-se de que usamos uma função `len()` para obter o comprimento de uma string. Uma vez que verificar o comprimento de uma sequência é uma tarefa comum, você provavelmente vai querer escrever uma função que pode fazer isso repetidamente.

As funções serão um dos níveis mais básicos de código de reutilização em Python, e também nos permitirá começar a pensar no design do programa (mergulhaemos muito mais nas idéias de design quando aprendemos sobre programação orientada a objetos).

def

Vamos ver como construir a sintaxe de uma função em Python. Ela tem a seguinte forma:

```
In [3]: def name_of_function(arg1, arg2):  
        '''  
        A documentação da função ficará aqui  
        '''  
        # Faça coisas aqui  
        return # retorne o resultado desejado aqui
```

Começamos com `def`, seguido do nome da função. Tente manter os nomes relevantes, por exemplo `len()` é um bom nome para uma função `length()`. Também tenha cuidado com os nomes, você não gostaria de chamar uma função do mesmo nome que uma [função interna em Python](#) (como `len`).

Em seguida, venha um par de parênteses com vários argumentos separados por uma vírgula. Esses argumentos são as entradas para sua função. Você poderá usar essas entradas em sua função e fazer referência a elas. Depois disso, você coloca dois pontos.

Agora, aqui é o passo importante, você deve indentar para começar o código dentro de sua função corretamente. Python faz uso de *espaço em branco* para organizar o código. Muitas outras linguagens de programação não fazem isso, então tenha isso em mente.

Em seguida, você verá o doc-string, é aqui que você escreve uma descrição básica da função. Usando iPython e iPython Notebooks, você será capaz de ler estes documentos pressionando Shift + Tab após um nome de função. Documentações não são necessárias para funções simples, mas é uma boa prática colocá-las para que você ou outras pessoas possam facilmente entender o código que você escreve.

Depois de tudo isso, você começa a escrever o código que deseja executar.

A melhor maneira de aprender funções é através de exemplos. Então, vamos tentar passar por exemplos que se relacionam com os vários objetos e estruturas de dados que aprendemos antes.

Exemplo 1: Uma função simples de 'Olá'

```
In [5]: def say_hello():  
        print('hello')
```

Chame a função

```
In [6]: say_hello()
```

hello

Exemplo 2: Uma função de saudação simples

Vamos escrever uma função que cumprimenta pessoas com seu nome.

```
In [9]: def greeting(name):  
        print('Hello, %s' %name)
```

```
In [10]: greeting('Rodrigo')
```

Hello, Rodrigo

Usando o return

Vamos ver um exemplo que usa uma declaração de retorno. Return permite uma função para *retornar* um resultado que pode ser armazenado como uma variável, ou usado de qualquer maneira que um usuário deseje.

Exemplo 3: função de adição

```
In [11]: def add_num(num1, num2):  
        return num1+num2
```

```
In [12]: add_num(4,5)
```

Out[12]: 9

```
In [13]: result = add_num(4,5)
```

```
In [15]: print(result)
```

9

O que acontece se inserimos duas strings?

```
In [12]: print add_num('one', 'two')
```

onetwo

Note que, porque não declaramos tipos de variáveis em Python, esta função pode ser usada para adicionar números ou seqüências em conjunto! Mais tarde, aprenderemos sobre a adição de verificações para garantir que um usuário coloque os argumentos corretos em uma função.

Vamos também começar a usar as instruções *break*, *continue* e *pass* no nosso código. Nós apresentamos estes durante a palestra de tempo.

Finalmente, vamos passar por um exemplo completo de criar uma função para verificar se um número é primo (um exercício de entrevista comum).

Nós sabemos que um número é primordial se esse número é apenas divisível em 1 e em si mesmo. Vamos escrever a nossa primeira versão da função para verificar todos os números de 1 a N e executar verificações de módulo.

```
In [17]: def is_prime(num):
        '''
        Método para checar se é primo
        '''
        for n in range(2,num):
            if num % n == 0:
                print('Não primo')
                break
        else: # Se o módulo nunca for zero, é primo
            print('Primo')
```

```
In [18]: is_prime(16)
```

Não primo

Observe como quebramos o código após a declaração de impressão! Na verdade, podemos melhorar isso ao verificar somente a raiz quadrada do número-alvo, também podemos ignorar todos os números pares depois de verificar 2. Também mudaremos para retornar um valor booleano para obter um exemplo de usar declarações de retorno:

```
In [19]: import math

def is_prime(num):
    '''
    Melhor método para checar primos
    '''
    if num % 2 == 0 and num > 2:
```

```
        return False
    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if num % i == 0:
            return False
    return True
```

```
In [20]: is_prime(14)
```

```
Out[20]: False
```

Ótimo! Você deve agora ter uma compreensão básica sobre como criar suas próprias funções para salvar-se de escrever repetidamente o código!