Funções sem resultados

Marco A L Barbosa

malbarbo.pro.br

Departamento de Informática

Universidade Estadual de Maringá

Conteúdo

Introdução

Introdução

Quase todas as funções que vimos até agora devolvem resultados. Por exemplo, a função `sum` que soma os valores de uma lista

```
>>> sum([4, 3, 7])
14
```

No entanto, algumas funções, como **append**, não devolvem resultados

```
>>> xs = [1, 3, 7]
>>> xs.append(4)
>>>
```

Note que o fato de **append** não produzir resultado não tem relação com a forma que a chamada da função é especificada (lembre-se, a chamada `xs.append(4)` é equivalente a `list.append(xs, 4)`), mas com a maneira que **append** é implementada

Introdução

Porque executar uma função que não devolve resultado?

Pelo efeito colateral que ela gera

- A chamada `xs.append(4)` não devolve nenhum valor, mas tem o efeito colateral de acrescentar o valor `4` no final de `xs`

Introdução

Uma função tem **efeito colateral** se ela altera algum estado do programa ou produz algum efeito observável (como exibir na tela).

Duas questões são importantes neste contexto

- Como escrever funções com efeitos colaterais?
- Como escolher quando escrever funções com resultados e/ou com efeitos colaterais?

Funções com efeitos colaterais

Funções com efeitos colaterais

Como gerar efeitos colaterais em funções?

- Modificando um parâmetro (mutável) passado para a função
- Modificando uma variável não local
- Chamando outras funções com efeitos colaterais

Funções com efeitos colaterais

Nós vimos anteriormente que quando um objeto mutável é referenciado por duas variáveis ele pode ser alterado através de qualquer uma das duas variáveis

```
>>> xs = [1, 2]
>>> ys = xs
# xs e ys referenciam a mesma lista (que é um objeto mutável)
# A lista pode ser alterada através de xs
>>> xs.append(3)
>>> xs
[1, 2, 3]
>> ys
[1, 2, 3]
# Ou através de ys
>>> ys[1] = 5
>>> ys
[1, 5, 3]
>>> xs
[1, 5, 3]
```

Funções com efeitos colaterais

O mesmo ocorre quando uma variável que referencia um valor mutável é passada como argumento para uma função

- Tanto a variável fora da função quando a variável dentro da função (parâmetro) referenciam o mesmo objeto
- A alteração do valor através do parâmetro altera o mesmo objeto referenciado pela variável fora da função

Exemplo

```
def soma1_no_primeiro(lst):
    lst[0] = lst[0] + 1


>>> xs = [1, 2, 3]
>>> soma1_no_primeiro(xs)
>>> xs

[2, 2, 3]
>>> ys = [3, 2]
>>> soma1_no_primeiro(ys)
>>> ys

[4, 2]
```

Observe que não usamos **`return`** na função `soma1_no_primeiro`. A função não devolve um valor, mas tem o efeito colateral de alterar o parâmetro `lst`

Exemplo

Dado uma lista de números, defina uma função que some 1 a cada elemento da lista.

1. Defina uma função que devolva uma nova lista.
2. Defina uma função que altere a lista passada como parâmetro.

Exemplo

```
def soma_1(xs):
    '''
    Lista de números -> Lista de números
    Cria uma lista de números somando
    1 a cada elemento de xs.
    >>> soma_1([1, 3, 4])
    [2, 4, 5]
    >>> soma_1([5])
    [6]
    >>> soma_1([])
    []
    '''
    ys = []
    for x in xs:
        ys.append(x + 1)
    return ys
```

```
def soma_1_mod(xs):
    '''
    Lista de números -> None
    Modifica xs somando 1 a cada
    elemento.
    Exemplos
    >>> xs = [1, 3, 4]
    >>> soma_1_mod(xs)
    >>> xs
    [2, 4, 5]
    >>> xs = []
    >>> soma_1_mod(xs)
    >>> xs
    []
    '''
    i = 0
    while i < len(xs):
        xs[i] = xs[i] + 1
        i = i + 1
```

Funções com efeitos colaterais

Para algumas funções em Python que produzem resultado existe também uma versão que não produz resultado, mas tem efeito colateral.

Funções com efeitos colaterais

Enquanto a função `sorted` devolve uma nova lista ordenada, a função `list.sort` ordena a própria lista passada como parâmetro

```
>>> xs = [7, 3, 5]
>>> sorted(xs)
[3, 5, 7]
>>> xs
[7, 3, 5]
>>> xs.sort()
>>> xs
[3, 5, 7]
```

Funções com efeitos colaterais

Enquanto a função `reversed` devolve uma nova lista com os elementos em ordem invertida, a função `list.reverse` inverte a ordem dos elementos na própria lista passada como parâmetro

```
>>> xs = [7, 3, 5]
>>> reversed(xs)
[5, 3, 7]
>>> xs
[7, 3, 5]
>>> xs.reverse()
>>> xs
[5, 3, 7]
```

Funções com efeitos colaterais vs funções com resultados

Funções com efeitos colaterais vs funções com resultados

Quando escrever funções com resultados e/ou com efeitos colaterais?

- Como as funções sem efeitos colaterais são mais simples de escrever, entender e testar, deve-se dar preferência a funções sem efeitos colaterais
- As funções com efeitos colaterais são usadas para economizar memória e/ou tempo de execução
    - As funções `reverse` e `sort` não precisam armazenar duas cópias da lista que está sendo processada

Atividades

Atividades

Para cada exemplo e exercício do módulo “Listas, conjuntos e dicionários” escreva uma versão da função (se possível) que não devolva resultado mas produza o efeito de alterar o(s) argumento(s) da função.