

Listas

Programação I

2016.2017

Teresa Gonçalves
tcg@uevora.pt

Departamento de Informática, ECT-UÉ

Reminder...



Como aprender?

Estudar, estudar, estudar...

Praticar, praticar, praticar...

Cometer erros, cometer erros,
cometer erros...

Aprender com os erros, aprender
com os erros, aprender com os
erros ...

Sumário

Propriedades, operadores e indexação

Iteração

Métodos

Argumentos de função

Propriedades, operadores e indexação

Lista

Sequência de elementos

Os elementos podem ser de tipos diferentes

Tamanho da sequência

Função `len()`

Lista vazia: `[]`

`len([]) = 0`

Propriedades

Sequencial

Relação de ordem entre elementos

Indexável

Acesso direto a cada elemento através do índice

Mutável

É possível alterar

Valor de um elemento

Adicionar ou remover elementos

Operadores básicos

Concatenação

```
>>> a=[2,1]; b=[4,5,1]  
>>> print(a+b)  
[2,1,4,5,1]
```

Repetição

```
>>> lista=(a+b)*2  
>>> lista  
[2,1,4,5,1,2,1,4,5,1]
```

Indexação

Acesso direto a um elemento da lista

Operador []

Índice

Pode ser uma expressão que contém variáveis e operadores... mas tem de ser um inteiro

1º elemento: 0

Último elemento: len()-1

Exemplo

```
>>> numbers=[1,5,3,4]
```

```
>>> numbers[1]
```

```
5
```


Índices negativos

Contam do fim para o início

```
>>> numbers=[ 1,5,3,4 ]
```

```
>>> numbers[-1]
```

```
4
```

```
>>> numbers[-2]
```

```
3
```

Operador ==

Verificação de equivalência

Operador binário booleano

Retorna True se as listas tiverem os mesmos elementos

A ordem tem importância

Exemplo

```
>>> a=[1,2,3]; b=[]; c=[1,3,2]; d=[1,2,3]
```

```
>>> a==b
```

```
False
```

```
>>> a==c
```

```
False
```

```
>>> a==d
```

```
True
```

Operador is

Verificação de identidade

Operador binário booleano

Retorna True as variáveis referem a mesma lista

Exemplo

```
>>> a=[1,2]; b=a; c=[1,2]
```

```
>>> a==b
```

```
True
```

```
>>> a is b
```

```
True
```

```
>>> a==c
```

```
True
```

```
>>> a is c
```

```
False
```

Operador in

expr in lista

Operador binário booleano

Operação entre um elemento e uma lista

Retorna True se o resultado da expressão é um elemento da lista

Exemplo

```
>>> lista=[ 'pao', 'queijo', 'fiambre', 'manteiga' ]
```

```
>>> 'pao' in lista
```

```
True
```

```
>>> 'azeite' in lista
```

```
False
```

Operadores <, >

Comparação

Comparam o 1º elemento

Se maior/menor, os restantes elementos não são considerados

Se igual compara-se o seguinte...

Exemplo

```
>>> a=[1,2,3]; b=[1,3,3]; c=[0,3]
```

```
>>> a<b
```

```
True
```

```
>>> a>b
```

```
False
```

```
>>> a>c
```

```
True
```

Segmento da lista

Operador [m:n:k]

Devolve os elementos da sequência da posição m até à posição n-1, seleccionando os elementos de k em k posições

Omissão

1º índice – segmento começa no início da sequência

2º índice – segmento termina no final da sequência

3º índice – são seleccionados todos os elementos

Exemplo

```
>>> s=[1,2,3,4,5,6,7,8,9];
>>> s[1:6:2]
[2,4,6]
>>> s[1:3]
[2,3]
>>> s[:3]
[1,2,3]
>>> s[3:]
[4,5,6,7,8,9]
>>> b=s[:]; b
[1,2,3,4,5,6,7,8,9]
>>> b==s
True
>>> b is s
False
```

Segmento à esquerda da afetação

Substitui o segmento da lista pela lista à direita da afetação

Exemplo

```
>>> a=[1,2,3,4,5]; a[:3]=[9]
```

```
>>> a
```

```
[9,4,5]
```

```
>>> a=[1,2,3,4,5]; a[:3]=[]
```

```
>>> a
```

```
[4,5]
```

```
>>> a=[1,2,3,4,5]; a[:3]=[9,8,7,6]
```

```
>>> a
```

```
[9,8,7,6,4,5]
```


Iteração

Iteração sobre listas

Iterar na sequência

```
for elemento in lista:  
    print( elemento )
```

Iterar sobre os índices (for)

```
for indice in range(len(lista)):  
    print( lista[indice] )
```

Iterar sobre os índices (while)

```
i=0  
while i<len(lista):  
    print( lista[i] )  
    i = i+1
```

Manipulação de listas

Envolve operações sobre os elementos

Operações que não modificam a lista

Pesquisa de valores

Operações que modificam a lista

Adicionar ou retirar elementos

Modificar o valor dos elementos

Atenção

Se for necessário manter a lista original, as operações que modificam a lista devem ser realizadas sobre uma cópia

```
lista2=lista[:]
```

Métodos

Método

Função aplicada a um objeto

Utilização

`var.metodo()`

`var.metodo(argumentos)`

Métodos sobre listas

`append(), insert()`

`remove(), pop()`

`index(), count()`

`reverse(), sort(), extend()`

Adicionar elemento: `append()`, `insert()`

`l.append(e)`

Adiciona o elemento `e` no final da lista

`l.insert(i,e)`

Adiciona o elemento `e` na posição `i`

Exemplo

```
>>> lista=[1,2,3]
>>> lista.append(4)
>>> lista
[1,2,3,4]
>>> lista.insert(1,8)
[1,8,2,3,4]
```

Remover elemento: remove(), pop()

l.remove(e)

Remove a 1ª ocorrência do elemento e da lista e devolve-o

l.pop(i)

Remove o elemento na posição i e devolve-o

i opcional. Se inexistente, remove o último elemento da lista

Exemplo

```
>>> lista=[1,2,3,4,2]
>>> lista.remove(2)
>>> lista
[1,3,4,2]
>>> lista.pop(1)
3
>>> lista
[1,4,2]
```

Inverter lista: reverse()

l.reverse()

Inverte a ordem dos elementos da lista

Exemplo

```
>>> lista=[1,2,3,4]
```

```
>>> lista.reverse()
```

```
>>> lista
```

```
[4,3,2,1]
```

```
>>> l1=[1,2,3]
```

```
>>> l2=l1[::-1]
```

```
>>> l2
```

```
[3,2,1]
```


Ordenar lista: sort()

l.sort()

Ordena os elementos da lista

Exemplo

```
>>> lista=[1,2,4,3,2]
>>> lista.sort()
>>> lista
[1,2,2,3,4]
>>> lista2=['ab','b','a']
>>> lista2.sort()
>>> lista2
['a','ab','b']
```

Posição na lista: index()

l.index(e)

Devolve o índice da primeira ocorrência do elemento e

Dá erro se o elemento não existir

Exemplo

```
>>> lista=[1,2,3,4,2]  
>>> lista.index(2)  
>>> 1
```

Contagem: count()

l.count(e)

Devolve o número de elementos iguais a e existentes na lista

Exemplo

```
>>> lista=[1,2,3,4,2]
```

```
>>> lista.count(2)
```

```
2
```

```
>>> lista.count(5)
```

```
0
```

Extensão da lista: extend()

l.extend(lista2)

Estende a lista l, adicionando no final todos os elementos de lista2

Exemplo

```
>>> lista1=[1,2,3]; lista2=[4,2,1]
>>> lista1.extend(lista2)
>>> lista1
[1,2,3,4,2,1]
>>> lista1.extend( ['um elemento'] )
>>> lista1
[1,2,3,4,2,1,'um elemento']
```

Métodos – resumo

Não alteram a lista; devolvem valor

`index()`, `count()`

Alteram a lista; devolvem valor

`remove()`, `pop()`

Alteram a lista; não devolvem valor

`append()`, `insert()`, `extend()`, `sort()`, `reverse()`

Exemplo 1

Remover elementos da lista menores que valor limiar

```
def remove_menores( lista, limiar ):  
    tamanho=len(lista)  
    i=0  
    while i<tamanho:  
        if lista[i]<limiar:  
            removido=lista.pop(i)  
            tamanho=tamanho-1  
            continue  
        i= i+1
```

```
>>> lista=[2,4,5,1,2,7,8,4,9]  
>>> remove_menores(lista,5)  
>>> lista  
[5,7,8,9]
```

Exemplo 2

Devolver lista com números pares existentes na lista

```
def lista_pares( lista ):  
    pares=[]  
    for valor in lista:  
        if valor%2==0:  
            pares.append(valor)  
    return pares
```

```
>>> lista=[3,4,5,6,7,8,9]  
>>> pares=lista_pares(lista)  
>>> pares  
[4,6,8]
```

Argumentos de função

Objetos mutáveis e imutáveis

Tipos imutáveis

Os valores das variáveis não pode ser alterados. Ou melhor, ao fazer alteração a variável passa a referir outro valor.

int, float, string, (tuplo)

Tipos mutáveis

Os valores das variáveis podem ser alterados

list, (dictionary)

Argumentos de função – tipos imutáveis

A alteração de uma variável no corpo da função não altera o valor “original”

Exemplo

```
def func(val):  
    val += 'bar'
```

```
x = 'foo'  
print(x)      # 'foo'  
func(x)  
print(x)      # 'foo'
```

Argumentos de função – tipos mutáveis

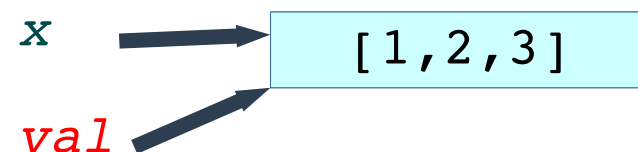
A alteração de uma variável no corpo da função altera o valor “original”

Exemplo

```
def func(val):  
    val += [3, 2, 1]
```

```
x = [1, 2, 3]  
print(x)          # [1, 2, 3]
```

```
func(x)  
print(x)          # [1, 2, 3, 3, 2, 1]
```



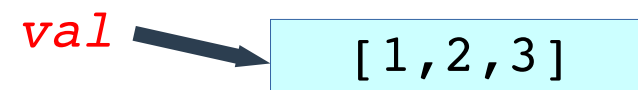
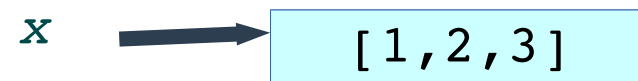
Argumentos de função – tipos mutáveis

Se quisermos manter a lista inicial é necessário passar uma cópia

```
def func(val):  
    val += [3, 2, 1]
```

```
x = [1, 2, 3]  
print(x)          # [1, 2, 3]
```

```
func(x[:])  
print(x)          # [1, 2, 3]
```



Argumentos de função – tipos mutáveis

Testes de igualdade e identidade

```
lista = [1, 2, 3]
def  testar(val):
    print(lista is val)
    print(lista == val)
```

```
testar(lista)
# True, True
```

```
testar(lista[:])
# False, True
```