Programação em Python

Processamento de listas

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

Agregações

Somar valores duma sequência

A função pré-definida `sum` soma os valores duma sequência.

Exemplo:

```
>>> sum([0.5, 1, 0.5])
2.0
```

Vamos definir a nossa própria implementação desta função.

Somar valores duma sequência

```
def soma(xs):
    # Somar valores duma lista xs
    # variável 'total' vai acumular a soma
    total = 0
    # percorre todos os valores
    for x in xs:
        # acrescenta ao total
        total = total + x
    # o resultado da função é 'total'
    return total
```

Exemplos

```
>>> soma([1, 2, 3])
6

>>> soma([1.0, 0.5, 2.0])
3.5

>>> soma([1.0])
1.0

>>> soma([])
0
```

Outras agregações

- A média aritmética
- O produto
- O máximo ou mínimo

Média aritmética duma sequência

```
def media(xs):
    # Calcula a média aritmética duma lista
    # média = soma / número de elementos
    return soma(xs)/len(xs)
```

Exemplos

```
>>> media([2.0, 1.0, 1.0])
1.33333333333

>>> media([2.0])
2.0
```

A média não está definida para a lista vazia:

```
>>> media([])
ZeroDivisionError: integer division or
modulo by zero
```

Produto duma lista

Análogo à soma, substituindo:

- o operador $+$ por $\star$;
- o valor inicial 0 por 1 (elemento neutro de $\star$).

Produto duma lista

```
def produto(xs):
    # Produto duma lista 'xs'
    # variável 'total' vai acumular o producto
    total = 1
    # percorrer todos os valores
    for x in xs:
        # acrescenta ao total
        total = total * x
    # retorna o resultado
    return total
```

Exemplos

```
>>> produto([2, 3, 4])
24

>>> produto(range(1,5))
24

>>> produto([])
1
```

Valor máximo duma lista

- Guardar o maior valor já encontrado
- Inicialmente é o primeiro elemento da sequência
- Operação de agregação: tomar o máximo

Valor máximo duma lista

```
def maximo(xs):
    # Maior valor duma sequência 'xs'
    # inicialmente: maior é o primeiro valor
    maior = xs[0]
    # percorrer os restantes elementos
    for x in xs[1:]:
        maior = max(maior, x)
    # retorna o maior
    return maior
```

Exemplos

```
>>> maximo([1.0, 2.5, 2.0, -1.0])
2.5

>>> maximo([2.0])
2.0
```

O máximo não está definido para a sequência vazia:

```
>>> maximo([])
IndexError: list index out of range
```

Eliminar repetições

Eliminar elementos repetidos

Vamos definir uma função para eliminar os elementos repetidos duma lista.

Duas soluções:

1. construir uma lista nova sem elementos repetidos;
2. modificar a lista original, apagando os elementos repetidos.

Eliminar elementos repetidos (1)

- Construir uma nova lista *ys*
- Inicialmente $ys = [\,]$
- Para cada elemento *x* na lista dada:
  se $x \notin ys$ acrescenta *x* a *ys*

Eliminar elementos repetidos (1)

```
def elimrep1(xs):
    # Constrói uma nova lista sem repetidos
    # ys é a lista sem repetidos
    # inicialmente vazia
    ys = []
    for x in xs:
        # se x não é repetido
        if not (x in ys):
            # acrescenta 'x' a 'ys'
            ys.append(x)
    # retorna a lista sem repetidos
    return ys
```

Exemplos

```
>>> elimrep1([2,1,3,3,1,4,1])
[2, 1, 3, 4]

>>> elimrep1([2,2,2])
[2]

>>> elimrep1([1,2,3])
[1, 2, 3]
```

Eliminar elementos repetidos (2)

- Uma lista *xs* com *n* elementos
- Um ciclo sobre os índices $0 \leq i \leq n-1$
- Invariante: não há repetidos nos índices inferiores a *i*

$$\overbrace{xs[0], \quad \ldots, \quad xs[i-1],}^{\text{sem repetidos}} \overbrace{xs[i], \quad \ldots, \quad xs[n-1]}^{\text{pode haver repetidos}}$$

- Actualização: se $xs[i] \in \{xs[0], xs[1], \ldots, xs[i-1]\}$ então apaga $xs[i]$; senão $i \leftarrow i+1$

Eliminar elementos repetidos (2)

```
def elimrep2(xs):
    # Eliminar elementos repetidos em 'xs'
    i = 0
    while i<len(xs):
        if xs[i] in xs[0:i]:
            # xs[i] é repetido: apaga-o
            del xs[i]
        else:
            # xs[i] não é repetido: avança
            i = i+1
```

Note que o `elimrep2` modifica a lista `xs` em vez de retornar uma nova lista.

Exemplos

```
>>> lista = [2,1,3,3,1,4,1]
>>> elimrep2(lista)
>>> lista
[2, 1, 3, 4]

>>> lista = [2,2,2]
>>> elimrep2(lista)
>>> lista
[2]

>>> lista = [1,2,3]
>>> elimrep2(lista)
>>> lista
[1, 2, 3]
```

Sumário

- Ambas as soluções são correctas
- A primeira solução:
  - constrói uma nova lista;
  - é mais simples de compreender.
- A segunda solução:
  - modifica a lista original;
  - pode ser mais eficiente (necessita de menos memória).

Crivo de Eratóstenes

Crivo de Eratóstenes

Construir a tabela dos números primos até n:

1. escrevemos todos os inteiros de 2 até n
2. o primeiro número (2) é primo e riscamos todos os seus múltiplos
3. o segundo número (3) é primo e riscamos todos os seus múltiplos
4. repetimos o processo: o próximo número não riscado é primo e riscamos todos os seus múltiplos
5. no fim: a tabela contém todos os números primos menores do que n

Crivo de Eratóstenes

Começamos com todos os inteiros de 2 até 30

2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	

Crivo de Eratóstenes

O número 2 é primo e riscamos os seus múltiplos

2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11
~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21
~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~	

Crivo de Eratóstenes

O número 3 é primo e riscamos os seus múltiplos

2	**3**	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~	

Crivo de Eratóstenes

O número 4 não é primo (já foi riscado)

2	**3**	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~	

Crivo de Eratóstenes

O número 5 é primo e riscamos os seus múltiplos

2	**3**	~~4~~	**5**	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~	

Crivo de Eratóstenes

Repetimos o processo até esgotar a tabela; restam apenas os primos inferiores a 30.

2	**3**	*4*	**5**	*6*	**7**	*8*	*9*	*10*	**11**
12	**13**	*14*	*15*	*16*	**17**	*18*	**19**	*20*	*21*
22	**23**	*24*	*25*	*26*	*27*	*28*	**29**	*30*	

Implementação do crivo

- Representamos a tabela de números por uma lista
- Removemos da lista os números compostos
- No final restam apenas os primos
- Nota: modificamos a lista original

Implementação do crivo

```
def crivo(xs):
    # Remove números compostos da lista 'xs'
    i = 0
    while i<len(xs):
        p = xs[i]    # p é primo
        j = i+1      # vamos remover múltiplos
        while j<len(xs):
            if xs[j]%p == 0:
                del xs[j]
            else:
                j = j+1
        i = i+1      # próximo primo
    # fim do crivo
```

Construir uma lista de primos

```
def primos(n):
    # Constrói a lista de primos menores que n
    xs = list(range(2,n)) # inteiros entre 2 e n-1
    crivo(xs)             # remove os compostos
    return xs             # lista de primos
```

Note que o `crivo` modifica a lista `xs` em vez de retornar uma nova lista.

Exemplos

Calcular todos os primos até 100:

```
>>> primos(100)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

Quantos primos inferiores a 1000?

```
>>> len(primos(1000))
168
```

Listas em compreensão

Listas em compreensão

É muito comum construir uma lista partindo de uma outra:

- selecionando elementos usando uma condição;
- aplicando uma transformação a cada elemento.

Exemplo: calcular quadrados

Construir a lista dos quadrados dos números inteiros de 1 a 9.

Exemplo: calcular quadrados (cont.)

Solução usando um ciclo for:

```
lista = []
for x in range(1, 10):
    lista.append(x**2)
print(lista)
```

Solução usando uma lista em compreensão:

```
lista = [x**2 for x in range(1,10)]
print(lista)
```

Sintaxe

```
[x**2 for x in range(1,10)]
```

Notação inspirada na teoria de conjuntos:

$$\{x^2 : x \in \{1, \ldots, 9\}\}$$

Mais geralmente:

```
[expressão for variável in sequência]
```

Mais exemplos

```
>>> [i**2 for i in [2,3,5,7]]
[4, 9, 25, 49]

>>> [1+x/2 for x in [0, 1, 2]]
[1.0, 1.5, 2.0]

>>> [ord(c) for c in "ABCDEF"]
[65, 66, 67, 68, 69, 70]

>>> [len(s) for s in
     "As armas e os barões assinalados".split()]
[2, 5, 1, 2, 6, 11]
```

Listas em compreensão com condições

Exemplo: quadrados dos múltiplos de 3 inferiores a 10.

```
[x**2 for x in range(10) if x%3==0]
```

Mais geralmente:

```
[expr for variável in sequência if condição]
```

Outro exemplo

Um número natural *n* é *primo* se não tem divisores próprios (i.e. maiores do que 1 e menores do que *n*).

Para testar se *n* é primo podemos testar se a lista dos divisores próprios é vazia.

Testar primos

```
def primo(n):
    # lista dos divisores próprios
    divs = [d for d in range(2,n) if n%d==0]
    # n é primo se e só se a lista for vazia
    return len(divs)==0
```

Nota: esta solução é ineficiente (calcula *todos* os divisores em vez de terminar após encontrar o primeiro...)

Listas em compreensão imbricadas

Podemos usar uma lista em compreensão dentro de outra.

Exemplo:

```
>>> [[i*j for j in range(1,11)] for i in range(1,11)]
```

produz a matriz da multiplicação:

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
 [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
 [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
...
 [9, 18, 27, 36, 45, 54, 63, 72, 81, 90],
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]]
```

Compreensões com múltiplas sequências

A ordem do resultado depende da ordem das sequências:

```
>>> [(x,y) for x in "ABC" for y in range(3)]
[('A', 0), ('A', 1), ('A', 2),
 ('B', 0), ('B', 1), ('B', 2),
 ('C', 0), ('C', 1), ('C', 2)]

>>> [(x,y) for y in range(3) for x in "ABC"]
[('A', 0), ('B', 0), ('C', 0),
 ('A', 1), ('B', 1), ('C', 1),
 ('A', 2), ('B', 2), ('C', 2)]
```

Sintaxe geral

```
[expr for var1 in seq1 if cond1
      for var2 in seq2 if cond2
      ⋮
      for varN in seqN if condN]
```