Listas, conjuntos e dicionários

Marco A L Barbosa

malbarbo.pro.br

Departamento de Informática

Universidade Estadual de Maringá

Conteúdo

Introdução

Introdução

As funções que escrevemos até agora trabalham com uma quantidade fixa de dados. Neste módulo veremos como escrever funções que processam uma quantidade variável de dados.

Introdução

Defina uma função que calcule a média de uma lista de valores.

Antes de resolver este problema precisamos aprender como armazenar uma lista de valores.

Listas

Listas

O Python fornece um tipo de dado chamado `list` que é utilizado para armazenar uma lista de valores.

Listas

Criamos uma lista escrevendo os seus elementos entre colchetes

```
>>> xs = [1, 4, 5]  # cria uma lista com 3 elementos
>>> xs
[1, 4, 5]
>>> len(xs)         # quantidade de elementos de xs
3
>>> ys = []         # cria uma lista com 0 elementos
>>> ys
[]
>>> len(ys)         # quantidade de elementos de ys
0
```

Listas

Cada elemento de uma lista pode ser acessado individualmente usando um índice (posição). O primeiro elemento tem índice 0, o segundo 1, e assim por diante

```
>>> xs = [1, 4, 5]
>>> xs[0]
1
>>> 2 * xs[1] + xs[2]
13
>>> xs[3] # não existe elemento no índice 3!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Listas

Além das função `len`, outras operações sobre listas são definidas pelo Python

Repetição

```
>>> ["casa"] * 3
["casa", "casa", "casa"]
>>> [4, -1, 2] * 2
[4, -1, 2, 4, -1, 2]
>>> [1, 2, 3] * 0
[]
```

Concatenação

```
>>> [7, 1] + [1, 2, 3]
[7, 1, 1, 2, 3]
>>> [5] + [2] + []
[5, 2]
# concatenação só com outra lista!
>>> [1, 2, 3] + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
```

Listas

Listas são objetos mutáveis, ou seja, elas podem ser alteradas

```
# reserva uma célula de memória com 3 posições
>>> xs = [1, 2, 3]
# altera o valor da posição 1 da célula de memória
>>> xs[1] = 10
>>> xs
[1, 10, 3]
# reserva uma nova célula de memória com 3 posições
>>> xs = [4, 5, 6]
```

Observe a diferença entre alterar **um valor** da lista associada com `xs` e alterar **a lista** associada com `xs`

- Quando um elemento de `xs` é alterado (`x[1] = 10`), a célula de memória associada com `xs` permanece a mesma, o valor armazenado da célula de memória é que é alterado
- Quando `xs` é alterado (`xs = [4, 5, 6]`), uma nova célula de memória é reservada e associada com `xs`, a antiga célula de memória associada com `xs` é devolvida para o sistema

Listas

O fato das listas serem objetos mutáveis pode gerar alguns resultados inesperados

```
>>> xs = [5, 8, 2]
>>> ys = xs
>>> xs[0] = 2
>>> xs
[2, 8, 2]
>>> ys
[2, 8, 2]
```

Vamos discutir este assunto em um próximo módulo, por hora, este questão não influenciará as funções que vamos escrever

As vezes precisamos criar uma lista de forma incremental, um elemento por vez, neste caso utilizamos a função **append** ao invés de concatenação de listas

```
>>> xs = []
# adiciona o valor 10 ao final da lista
>>> xs.append(10)
>>> xs
[10]
# adiciona o valor 20 ao final da lista
>>> xs.append(20)
>>> xs
[10, 20]
```

A forma de executar a função `append` é diferente das outras funções

- Quando `xs` é uma lista, a chamada `xs.append(valor)` é equivalente a `list.append(xs, valor)`
- As funções que são executadas desta forma são chamadas de **métodos**

Observe que a função `append` não produz nenhum valor, mas ela altera o valor de `xs` (vamos discutir com mais detalhes esta questão em um próximo módulo)

Também é possível remover um elemento de uma posição de uma lista

```
>>> xs = [5, 8, 2, 9]
>>> del xs[1]
>>> xs
[5, 2, 9]
>>> del xs[0]
[2, 9]
```

Listas

Para processar uma lista de valores temos que usar repetição

Podemos utilizar a instrução **while** para acessar cada item de `xs` usando um índice `i` que começa com `0` e é incrementado de `1` em `1`

```
>>> xs = [3, -1, 4]
>>> soma = 0
>>> i = 0
>>> while i < len(xs):
...     soma = soma + xs[i]
...     i = i + 1
...
>>> (i, soma)
```

(?, ?) (3, 6)

O Python oferece a instrução **`for`** que é mais adequada quando estamos interessados nos elementos da lista, sem se importar com os índices

No exemplo a seguir a linha `soma = soma + x` é executada uma vez para cada valor da lista `[3, -1, 4]`. Na primeira vez `x` é `3`, na segunda vez `x` é `-1` e na terceira vez `x` é `4`

```
>>> xs = [3, -1, 4]
>>> soma = 0
>>> for x in xs:
...     soma = soma + x
...
>>> soma
? 6
```

```
def f(xs):
    i = 0
    s = 0
    while i < len(xs):
        s = s + xs[i]
        i = i + 2
    return s
```

Que resultado será exibido após a avaliação da seguinte expressão?

Descreva o que a função `f` faz e dê outro exemplo de uso da função.

```
>>> f([1, 2, 4, 7, 1, 2])
? 6
```

```
def f(xs):
    p = 1
    for x in xs:
        p = p * x
    return p
```

Que resultado será impresso após a avaliação da seguinte expressão?

Descreva o que a função `f` faz e dê outro exemplo de uso da função.

```
>>> f([4, 2, 7])
```

? 56

```
def f(xs):
    p = True
    a = xs[0]
    for x in xs:
        if x != a:
            p = False
    return p
```

Que resultado será impresso após a avaliação da seguinte expressão?

Descreva o que a função `f` faz e dê outro exemplo de uso da função.

```
>>> f([2, 2, 2])
? True
```

```
def f(n, c):
    xs = []
    i = 0
    while i < c:
        xs.append(i * n)
        i = i + 1
    return xs
```

Que resultado será impresso após a avaliação da seguinte expressão?

Descreva o que a função `f` faz e dê outro exemplo de uso da função.

```
>>> f(3, 5)
? [0, 3, 6, 9, 12]
```

```
def f(xs, n):
    s = 0
    i = 0
    while i < len(xs) and i < n:
        s = s + xs[i]
        i = i + 1
    j = len(xs) - 1
    while j >= 0 and j >= len(xs) - n:
        s = s - xs[j]
        j = j - 1
    return s == 0
```

Que resultado será impresso após a avaliação da seguinte expressão? Descreva o que a função `f` faz e dê outro exemplo de uso da função.

```
>>> f([3, 2, 1, 7, 2, 9, 8, 1, 1, 4], 3)
? True
```

Defina uma função que calcule a média dos valores de uma lista não vazia.

```
def media(xs):
    '''
    Lista de Números -> Número
    Calcula a média dos valores da lista não vazia xs.
    Exemplos
    >>> media([3.0])
    3.0
    >>> media([3.0, 4.0, 5.0])
    4.0
    '''
    soma = 0
    for x in xs:
        soma = soma + x
    return soma / len(xs)
```

Defina uma função que encontre o valor máximo de uma lista não vazia.

```
def maximo(xs):
    '''
    Lista de Números -> Número
    Encontra o valor máximo da lista não vazia xs.
    Exemplo
    >>> maximo([5, 7, 1, 2])
    7
    >>> maximo([-3, -2, -1, -5])
    -1
    '''
    max = xs[0]
    for x in xs:
        if x > max:
            max = x
    return max
```

Defina uma função que indique se um dado valor está presente em uma lista.

Defina uma função que devolva o índice (posição) da primeira ocorrência do valor máximo de uma lista não vazia.

Defina uma função que devolva os índices (posições) de todas as ocorrências do valor máximo de uma lista não vazia.

Defina uma função que receba como parâmetro dois números inteiros não negativos a e b, onde $a \leq b$, e devolva uma lista com todos os números (em ordem crescente) no intervalo $[a, b]$.

Defina uma função que receba como parâmetro uma lista `xs` e crie uma nova lista com os elementos positivos de `xs`.

Algumas das funções que definimos nos exemplos também estão definidas na biblioteca do Python.

Definir estas funções é interessante para entender repetição e listas, mas na prática usamos as funções já existentes.

Soma

```
>>> xs = [3, -1, 4]
>>> sum(xs)
6
>>> sum([2, 4, 6, 2])
14
```

Máximo e mínimo

```
>>> xs = [3, -1, 4]
>>> max(xs)
4
>>> min(xs)
-1
```

Verificar se um valor está em uma lista

```
>>> xs = [3, -1, 4]
>>> 2 in xs
False
>>> 3 in xs
True
>>> 4 in xs
True
>>> 2 not in xs
True
```

Conjuntos

Conjuntos

Um conjunto é coleção não ordenada de elementos. Cada elemento de uma coleção é único e deve ser imutável.

Conjuntos

Criamos um conjunto não vazio escrevendo os seus elementos entre chaves

```
>>> xs = {4, -2, 3, 7, 3}
# A ordem dos elementos em um conjunto é indefinida
>>> xs
{3, 4, -2, 7}
>>> len(xs)
4
```

Conjuntos

Um conjunto vazio é criado com a função `set`

```
>>> xs = set()
>>> xs
set()
>>> len(xs)
0
```

Conjuntos

Os elementos de um conjunto devem ser imutáveis (números, strings, etc)

```
>>> xs = {'casa', 'carro', 'carro'}
>>> xs
{'carro', 'casa'}
```

Listas são mutáveis, por isso não é possível ter um conjunto de listas

```
>>> xs = {[1, 2, 3], [4, 5], [6]}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Conjuntos

Um conjunto também pode ser criado a partir de uma lista

```
>>> xs = [1, 2, 3, 2, 3, 1, 3]
>>> ys = set(xs)
>>> ys
{1, 2, 3}
```

Conjuntos

Algumas funções e operações sobre conjuntos são definidas pelo Python.

Conjuntos

União

```
>>> {3, 4, 7} | {8, 6, 4, 3}
{3, 4, 6, 7, 8}
```

Interseção

```
>>> {3, 4, 7} & {8, 6, 4, 3}
{3, 4}
```

Diferença (está no primeiro conjunto mas não no segundo)

```
>>> {3, 4, 7} - {8, 6, 4, 3}
{7}
>>> {8, 6, 4, 3} - {3, 4, 7}
{8, 6}
```

Diferença simétrica (está em apenas um dos dois conjuntos)

```
>>> {3, 4, 7} ^ {8, 6, 4, 3}
{6, 7, 8}
>>> {8, 6, 4, 3} ^ {3, 4, 7}
{6, 7, 8}
```

Conjuntos

Assim como uma lista, um conjunto é um objeto mutável.

Adição de elementos

```
>>> xs = set()
>>> xs.add(4)
>>> xs
{4}
>>> xs.add(7)
>>> xs
{4, 7}
>>> xs.add(4)
>>> xs
{4, 7}
```

Remoção de elementos

```
>>> xs = {1, 2, 4, 6}
>>> xs.remove(4)
>>> xs
{1, 2, 6}
# remove falha se o valor não estiver no conjunto
>>> xs.remove(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
```

Remoção de elementos

```
# discard é como remove, mas nunca falha
>>> xs = {1, 2, 4, 6}
>>> xs.discard(4)
>>> xs
{1, 2, 6}
>>> xs.discard(4)
>>> xs
{1, 2, 6}
```

Verificar se um valor está em um conjunto

```
>>> xs = {3, -1, 4}
>>> 2 in xs
False
>>> 3 in xs
True
>>> 4 in xs
True
>>> 2 not in xs
True
```

Conjuntos

Para processar um conjunto de valores temos que usar repetição. Não é possível usar o **`while`** porque os elementos de um conjunto não são indexados, por isso temos que usar o **`for`**

```
>>> xs = {3, -1, 4}
>>> soma = 0
>>> for x in xs:
...     soma = soma + x
...
>>> soma
6
```

Defina uma função que calcule a união de dois conjuntos (sem usar o operador `|`).

```
def uniao_conjunto(a, b):
    '''
    Conjunto de valores, Conjunto de valores -> Conjunto de valores
    Devolve um conjunto que é a união dos conjuntos a e b.
    Exemplos
    >>> uniao_conjunto({3, 4, 7}, {8, 6, 4, 3})
    {3, 4, 6, 7, 8}
    '''
    c = set()
    for x in a:
        c.add(x)
    for x in b:
        c.add(x)
    return c
```

Dicionários

Dicionários

Um dicionário é uma generalização de uma lista

- Em uma lista os índices são números começando com 0
- Os índices em um dicionário podem ser qualquer tipo imutável

Dicionários

Um dicionário contém uma coleção de índices, chamados chaves, e uma coleção de valores

- Cada chave é associada a um único valor
- Cada par chave-valor é chamado de item

Dicionários

Criamos um dicionário escrevendo os seus itens entre chaves. Cada item é escrito da forma `chave: valor`

```
>>> cores = { 2: 'verde', 7: 'preto', 1: 'rosa'}
>>> cores
{ 2: 'verde', 7: 'preto', 1: 'rosa'}
>>> len(cores)
3
>>> x = {}  # Dicionário vazio
>>> x
{}
>>> len(x)
0
```

Dicionários

Os valores de um dicionário são acessados pela chave (como se fosse um índice de uma lista)

```
>>> cores = { 2: 'verde', 7: 'preto', 1: 'rosa'}
>>> cores[2]
'verde'
>>> cores[1]
'rosa'
>>> cores[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

Dicionários

Assim como as listas e conjuntos, os dicionários são mutáveis

```
>>> freq = {'laranja': 20, 'goiaba': 4}
# Altera o valor associado com a chave 'laranja'
>>> freq['laranja'] = freq['laranja'] + 1
>>> freq
{'laranja': 21, 'goiaba': 4}
# Adiciona um nova par chave-valor
>>> freq['pera'] = 1
>>> freq
{'laranja': 21, 'goiaba': 4, 'pera': 1}
# Remove o item com chave 'goiaba'
>>> del freq['goiaba']
>>> freq
{'laranja': 21, 'pera': 1}
```

Verificar se uma chave está presente em um dicionário

```
>>> freq = {'laranja': 20, 'goiaba': 10}
>>> 'laranja' in freq
True
>>> 'pera' in freq
False
# O dicionário não tem a chave 20!
# 20 é o valor associado com a chave 'laranja'
>>> 20 in freq
False
```

Dicionários

Para processar os elementos de um dicionário, podemos usar o **`for`** de duas maneiras.

Iteração nas chaves

```
>>> cores = { 2: 'verde', 7: 'preto', 1: 'rosa'}
>>> chaves = []
>>> for x in cores:
...     chaves.append(x)
...
>>> chaves
[2, 7, 1]
```

Iteração nos itens

```
>>> cores = { 2: 'verde', 7: 'preto', 1: 'rosa'}
>>> chaves = []
>>> valores = []
>>> for chave, valor in cores.items():
...     chaves.append(chave)
...     valores.append(valor)
...
>>> chaves
[2, 7, 1]
>>> valores
['verde', 'preto', 'rosa']
```

Defina uma função que receba como entrada uma lista de valores e devolva um dicionário que associe cada valor da lista com a posição de sua primeira ocorrência na lista.

```
# Exemplo
>>> primeira_posicao([3, 3, 5, 5, 3, 4, 4, 3, 5, 4])
{3: 0, 5: 2, 4: 5}
# A primeira ocorrência do 3 é na posição 0
# A primeira ocorrência do 5 é na posição 2
# A primeira ocorrência do 4 é na posição 5
```

```
def primeira_posicao(xs):
    '''
    Lista -> Dicionario
    Devolve um dicionário que associa cada elemento de xs
    com a primeira posição que ele aparece em xs.
    Exemplos
    >>> primeira_posicao([3, 3, 5, 5, 3, 4, 4, 3, 5, 4])
    {3: 0, 5: 2, 4: 5}
    '''
    pos = {}
    i = 0
    while i < len(xs):
        if xs[i] not in pos:
            pos[xs[i]] = i
        i = i + 1
    return pos
```

Atividades

Atividades

1. Defina uma função que conte quantas vezes um determinado valor aparece em uma lista.
2. Defina uma função que verifique se uma lista tem mais valores positivos ou negativos.
3. Defina uma função que verifique se os valores de uma lista estão em ordem crescente.

Atividades

4. Defina uma função que receba como entrada uma lista `xs` e devolva um nova lista com os mesmos elementos de `xs` mas em ordem reversa (o último elemento de `xs` aparece primeiro, o penúltimo elemento aparece em segundo e assim por diante).

5. Defina uma função que receba como entrada um lista `xs` e um valor `a` e devolva uma nova lista com os elementos de `xs` diferentes de `a`.

6. Defina uma função que divida cada elemento de uma lista pelo valor máximo da lista.

Atividades

7. Defina uma função que receba como entrada uma lista e devolva a posição do valor mínimo da lista.

8. Defina uma função que receba como entrada uma lista e uma posição e devolva uma nova lista sem o elemento na posição especificada (Note que a função pré-definida `del` não é apropriada para esta situação pois ela modifica a própria lista, este exercício pede para que uma nova lista seja criada)

Atividades

9. Ordenação por seleção é uma algoritmo para ordenar uma lista de valores. A ideia do algoritmo é encontrar o menor valor na lista de entrada, remover este valor da lista e em seguida inserir este valor na lista de saída. Baseado nesta descrição, defina uma função que receba como entrada uma lista e devolva uma nova lista com os valores de entrada ordenados. Utilize as funções dos dois exercícios anteriores para fazer a implementação!

10. Defina uma função que receba como entrada uma lista (com os valores em ordem não decrescente) e um valor **n** e devolva uma nova lista com todos os valores da lista de entrada e com o valor de **n** em ordem não decrescente. Exemplo

```
>>> insere_ordenado([3, 3, 6, 10], 4)
[3, 3, 4, 6, 10]
```

Atividades

11. Ordenação por inserção é outro algoritmo para ordenar uma lista de valores. A ideia do algoritmo e analisar cada elemento da lista de entrada (na ordem que eles aparecem na lista) e colocá-lo em ordem na lista de saída. Usando a função `insere_ordenado` defina uma função que receba como entrada uma lista e devolva uma nova lista com os valores de entrada ordenados.

Atividades

12. Defina uma função que calcule a diferença simétrica entre dois conjuntos (sem usar o operador ^).

13. Defina uma função que encontre os valores que mais aparecem em uma lista (Dica: use um dicionário para armazenar a frequência dos valores).

Leitura recomendada

Livro Pense em Python 2ª edição. Allen B. Downey

- Capítulo 10 - Listas
- Capítulo 11 - Dicionários