Processamento da Informação

Fabrício Olivetti de França

02 de Fevereiro de 2019

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Topics

1. Listas e Vetores

Listas e Vetores

Listas e Vetores

Uma lista em Python é um *container* de informações que, por algum motivo, devem estar agrupadas para serem processadas sequencialmente.

Listas e Vetores

O comportamento de uma lista é similar a de uma `string` (na verdade a `string` **é** uma lista de caracteres):

Listas e Vetores

Para criar uma lista, basta colocar elementos entre colchetes e separados por vírgula:

```
listaDeNums = [1,2,3,4,5,6,7,8,9,10]
listaDeBools = [False, True]
```

Operações com listas

A linguagem Python fornece algumas funções para manipularmos `strings`:

função	descrição
len(l)	retorna o tamanho da lista
<,>,==,!=,<=,>=	compara uma lista lexicograficamente
l1 + l2	concatena duas listas
l1 * n	replica a lista *n* vezes
l.append(x)	insere elemento *x* no final da lista

Indexação

A indexação é exatamente a mesma de uma `string`:

```
l[pos]
```

Indexação

A posição começa a contar do 0:

```
l = [1,2,3]
s[0] == 1
s[1] == 2
s[2] == 3
s[3] == ERRO!
```

Indexação

```
l = [1,2,3]
s[-1] == 3
s[-2] == 2
s[-3] == 1
```

Mutabilidade

Ao contrário da `string`, uma lista é **mutável**, ou seja, podemos alterar seu conteúdo:

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

Slices

Os `slices` funcionam da mesma forma que em uma `string`:

```
l = [1,2,3,4,5,6,7,8]
l[1:3] = ??
l[4:2:-1] = ??
l[1:6:2] = ??
l[::-1] = ??
```

Slices

Os **slices** funcionam da mesma forma que em uma **string**:

```
l = [1,2,3,4,5,6,7,8]
l[1:3] = [2,3]
l[4:2:-1] = [5,4]
l[1:6:2] = [2,4,6]
l[::-1] = [8,7,6,5,4,3,2,1]
```

Iterando

Para iterar uma lista, utilizamos o `for`:

```
l = [1,2,3,4]
for elemento in l:
    print(elemento)
```

Elementos híbridos

Note que não temos limitações em relação aos tipos de elementos em uma lista. Eles inclusivem podem se misturar:

```
hibrido = [1, False, 3.5]
```

Lista como vetores

Vamos interpretar as listas como vetores numéricos. Para tanto precisamos criar algumas funções de interesse prático como:

- operações algébricas elemento-a-elemento entre vetores
- produto interno e externo entre dois vetores
- média, variância e desvio-padrão dos elementos de um vetor
- encontrar o maior e menor valor
- buscar um elemento

dentre outros

Operações algébricas

Defina a função `somaVec` que soma dois vetores elemento a elemento. Caso o tamanho deles seja diferente, trunque o maior vetor.

Operações algébricas

```
def somaVec(v1, v2):
    v3 = []
    menor_tam = min(len(v1), len(v2))
    for i in range(menor_tam):
        v3.append(v1[i]+v2[i])
    return v3
```

Zippping

A função `zip` junta duas ou mais listas em tuplas dos elementos dessa lista:

```
v1 = [1,2,3]
v2 = [4,5,6, 7]
zip(v1,v2) = [(1,4), (2,5), (3,6)]
```

ela inclusive trunca o maior vetor!

Zippping

Com isso podemos reescrever nossa função anterior para:

```
def somaVec(v1,v2):
    v3 = []
    for x, y in zip(v1,v2):
        v3.append(x+y)
    return v3
```

Exercício

Escreva a função `multVec`!

Exercício

Defina a função `somatoria` que faz a soma dos elementos de um vetor.

Exercício

Defina a função `prodInterno` que calcula o produto interno entre dois vetores.

Exercício

Defina a função `media` que calcula a média dos elementos de um vetor.

Exercício

Defina a função `variancia`. Para isso, primeiro crie funções auxiliares!

Exercício

Como você definiria o desvio-padrão?

Exercício

Defina as funções `maior` e `menor` que retorna o maior e menor elemento de um vetor.

Exercício

Faça um algoritmo para gerar uma lista com a sequência de Fibonacci de tamanho n.

Exercício

Faça um algoritmo que receba uma lista de `string` e retorne uma lista contendo o tamanho de cada `string`.

Exercício

Faça um algoritmo que receba uma lista de `int` e retorne uma lista de `bool` indicando se cada valor é um número par ou não.

Função map

Os dois exercícios anteriores são praticamente iguais! Esse é um padrão frequente de programação.

Para facilitar, o Python fornece a função `map`:

```
def par(x):
    return x%2 == 0
list(map(par, [1,2,3,4,5,6]))
# [False, True, False, True, False, True]
```

Exercício

Faça um algoritmo que receba uma lista de `int` e retorne uma lista contendo apenas os valores ímpares.

Função `filter`

Esse padrão também é bastante comum e, por isso, temos a função `filter`:

```
def impar(x):
    return not par(x)

list(filter(impar, [1,2,3,4,5,6]))
# [1,3,5]
```

Exercício

Utilizando a função `filter` faça um algoritmo que conte o número de vogais de uma `string`.

Exercício

Crie um algoritmo que ordene os valores de uma lista!

Exercício

Utilizando a função anterior, faça um algoritmo que calcule a mediana de uma lista de números.

Exercício

Faça um programa que receba uma `string`, ordene suas letras, agrupe em uma lista de listas e retorne a frequência de ocorrência de cada letra:

```
frase = "a fila do fretado esta lotada"
frase_ordenada = ordena(frase)
# [' ', ' ', ' ', ' ', ' ', 'a', 'a', 'a', 'a', 'a', 'a',
# 'd', 'd', 'd', 'e', 'e', 'f', 'f', 'i', 'l', 'l', 'o',
# 'o', 'o', 'r', 's', 't', 't', 't']
```

Exercício

```
grupos = agrupa(frase_ordenada)
# [[' ', ' ', ' ', ' ', ' '], ['a', 'a', 'a', 'a', 'a', '
# ['d', 'd', 'd'], ['e', 'e'], ['f', 'f'], ['i'], ['l', '
# 'o', 'o'], ['r'], ['s'], ['t', 't', 't']]
```

Exercício

```
imprime_freq(grupos)
# ' ' : 5
# 'a' : 6
# 'd' : 3
# 'e' : 2
# 'f' : 2
# 'i' : 1
# 'l' : 2
# 'o' : 3
# 'r' : 1
# 's' : 1
# 't' : 3
```