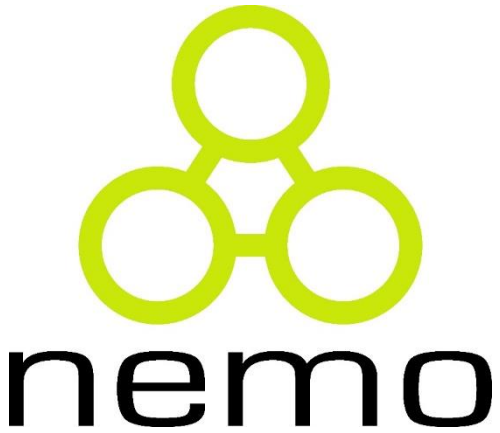


Programação I

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

LISTAS

- Vimos que as tuplas são agrupamentos de tamanhos pré-definidos e heterogêneos.

- Em contrapartida, listas são tipos compostos para agrupar quantidades indefinidas de elementos de um mesmo tipo.

- Uma lista é uma sequência de zero ou mais elementos de um mesmo tipo. Entende-se por sequência uma quantidade qualquer de itens dispostos linearmente. Podemos representar uma lista pela enumeração dos seus elementos, separados por vírgulas e cercados por colchetes.

[*e1*, *e2*, ..., *en*]

LISTAS – Definição por Enumeração

- Exemplos:
- >>> [ ]
- >>> [1,2,3,4,5]
- >>> ['a', 'e', 'i', 'o', 'u']
- >>> [(22,04,1500), (07,09,1822), (31,03,1964)]
- >>> [[1,2,5,10], [1,11], [1,2,3,4,6,12], [1,13], [1,2,7,14], [1,3,5,15]]

Formas de definir LISTAS - Definição por Intervalo

- Intervalo:
  - **range(**<limite inferior>**,**<limite superior + 1>**)**
  - **range(**<num_de_elementos>**)** {inicio em 0}

```
>>> range(1,6)
[1, 2, 3, 4, 5]
>>> range(-2,3)
[-2, -1, 0, 1, 2]
>>> range(10,3)
[ ]
>>> range(3)
[0, 1, 2]
```

nemo

Formas de definir LISTAS - Definição por Progressão Aritmética

- PA:
  - **range(**<limite inferior>**,**<limite superior>,<razão>**)**

```
>>> range(1,6,1)
[1, 2, 3, 4, 5]
>>> range(-2,3,2)
[-2, 0, 2]
>>> range(10,3,-1)
[10, 9, 8, 7, 6, 5, 4]
```

nemo

Operações Básicas

- Como nos demais tipos da linguagem, valores descritos por listas podem e devem ser usados na descrição de novos valores através da utilização de operações definidas sobre eles.

- **in :** avalia se um determinado elemento é membro de uma lista.

<elemento> **in** <lista>

```
>>> 7 in range(1,8)
True
>>> 3 in range(0,10,2)
False
```

Operações Básicas

- **len:** descreve o tamanho de uma lista.

len(<lista>**)**

```
>>> len([])
0
>>> len(range(1,234,3))
78
```

- **indexação:** podemos acessar cada termo de uma lista indicando sua a posição dentro da lista, considerando que o primeiro elemento tem a ordem zero.
- <lista> **[**<índice>**]**

nemo

Operações Básicas

- Em particular, o primeiro (conhecido como head) e o último elementos da lista são obtidos como:
- <lista> [0] e <lista> [<tamanho da lista>-1]

<lista> **[0]** e <lista> **[**<tamanho da lista>-1**]**

```
>>> [1,2,3][0]
1
>>> [0,1,2,3][4-1]
3
>>> range(0,20,4)[3]
12
```

nemo

Operações Básicas

- **sub-listas**: podemos descrever sublistas a partir de uma lista.
- Para tanto basta indicarmos a posição inicial e final da sub-lista, separadas pelo símbolo “:”.
- A omissão de uma ou outra posição indica início ou final da lista respectivamente.
- <lista> [<índice inicial>:<índice final> +1]
- Em particular, as sublistas contendo todos os elementos menos o primeiro (conhecida como tail), e aquela contento todos menos o último são obtidas como:
- <lista> [1:]
- <lista> [:<tamanho da lista>-1]

nemo

Operações Básicas

<lista> **[1:]** e <lista> **[:**<tamanho da lista>-1**]**

```
>>> [0,1,2,3][:]
[0, 1, 2, 3]
>>> [0,1,2,3][:3]
[0, 1, 2]
>>> [0,1,2,3][1:]
[1, 2, 3]
>>> [0,1,2,3][:2]
[0, 1]
>>> [0,1,2,3][2:]
[2, 3]
>>> [0,1,2,3][1:3]
[1, 2]
```

nemo

Operações Básicas

- **concatenação**: descreve uma nova lista obtida pela concatenação de uma lista de listas.

<lista> + <lista>

```
>>> [1,2,3] + [4,5]
[1, 2, 3, 4, 5]
>>> range(1,6) + range(4,0,-1)
[1, 2, 3, 4, 5, 4, 3, 2, 1]
```

nemo

Operações Básicas

- Em particular, pode-se adicionar elementos em uma lista, no inicio, no final, ou ainda no meio :
- **[<elemento>] + <lista>**
- ou

<lista> + [<elemento>]

ou

<lista> [:<posição>] + [<elemento>] + <lista> [<posição>:]

```
>>> [0] + [1,2,3]
[0, 1, 2, 3]
>>> range(1,5) + [5]
[1, 2, 3, 4, 5]
>>> range(1,5)[:2] + [0] + range(1,5)[2:]
[1, 2, 0, 3, 4]
```

nemo

Formas de definir LISTAS – Definição por Compreensão

- Podemos também descrever listas através das condições que um elemento deve satisfazer para pertencer a ela.
- Em outras palavras, queremos descrever uma lista através de uma intenção. Esta forma é análoga à que já conhecemos para descrição de conjuntos.

- Por exemplo, é usual escrever a notação abaixo para descrever o conjunto formado pelo quadrado dos números naturais menores que 10

Formas de definir LISTAS – Definição por Compreensão

- Sintaxe de List Comprehension:
  - **[** <expressão> **for** <variável> **in** <lista-conhecida> **]**

```
>>> [x**2 for x in range(0,5)]
[0, 1, 4, 9, 16]
```

  - **[** <expressão> **for** <variável> **in** <lista-conhecida> **if** <condição>**]**

```
>>> [x for x in range(1,100,4) if x%2==1]
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77, 81, 85,
   89, 93, 97]
```

nemo

Formas de definir LISTAS

- Comprehension:
  - Podemos usar mais de uma variável para percorrer listas diferentes

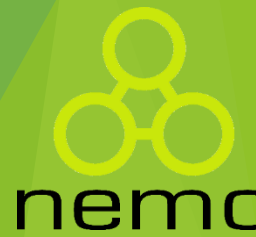
```
>>> [x*y for x in [0,1,2] for y in [5,10]]
[0, 0, 5, 10, 10, 20]
```

  - Podemos formar listas de listas ou de tuplas usando uma ou mais variáveis

```
>>> [(x, x**2) for x in range(1,9)]
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64)]
>>> [[x, x**2] for x in range(1,9)]
[[1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64]]
>>> [(x, y) for x in [1,2,3] for y in [-3,7,11]]
[(1, -3), (1, 7), (1, 11), (2, -3), (2, 7), (2, 11), (3, -3), (3, 7), (3, 11)]
```

Exercícios

- Definir uma lista dos números de 1 a 5
- Definir uma lista dos números pares de 1 a 5
- Definir uma lista de tuplas contendo um número par e o seu consecutivo num intervalo de 1 a 5
- Escreva uma função que determina as distâncias entre três pontos no plano cartesiano, duas a duas. Tanto os pontos dados como entrada, como as distâncias calculadas devem ser representadas por tuplas. Na resposta, cada par de pontos deve ser exibido, seguido da distância existente entre eles.



Função Map

- **> map(<função>, <lista-conhecida>) => <lista-resultante>**
- Esta operação consiste em aplicar uma dada função a cada elemento de uma lista conhecida, produzindo-se então uma lista resultante com exatamente a mesma quantidade de elementos.
- Porém, o tipo de dado da lista resultante pode diferir da lista original, pois é determinado pelo tipo de retorno da função.
- A função, por sua vez, deve ter apenas um parâmetro, compatível com os elementos da lista original.

Função Map

- Exemplo:

```
>>> def f(x): return 2*x
>>> map(f, range(1,6))
[2, 4, 6, 8, 10]

>>> def f(x): return x%2==0
>>> map(f, range(1,6))
[False, True, False, True, False]
```

nemo

Função Filter

- **> filter(<função>, <lista-conhecida>) => <lista-resultante>**

- Esta operação consiste em aplicar um filtro a uma lista conhecida. Este filtro é dado por uma função de verificação, de forma que a lista resultante terá como elementos apenas aqueles para os quais a avaliação da função é verdadeira.

- A lista resultante será sempre uma sublista da lista original, e terá uma quantidade de elementos menor ou igual.

- A função, portanto, deve ser de verificação e ter apenas um parâmetro, compatível com os elementos da lista original.

Função Filter

- Exemplo:

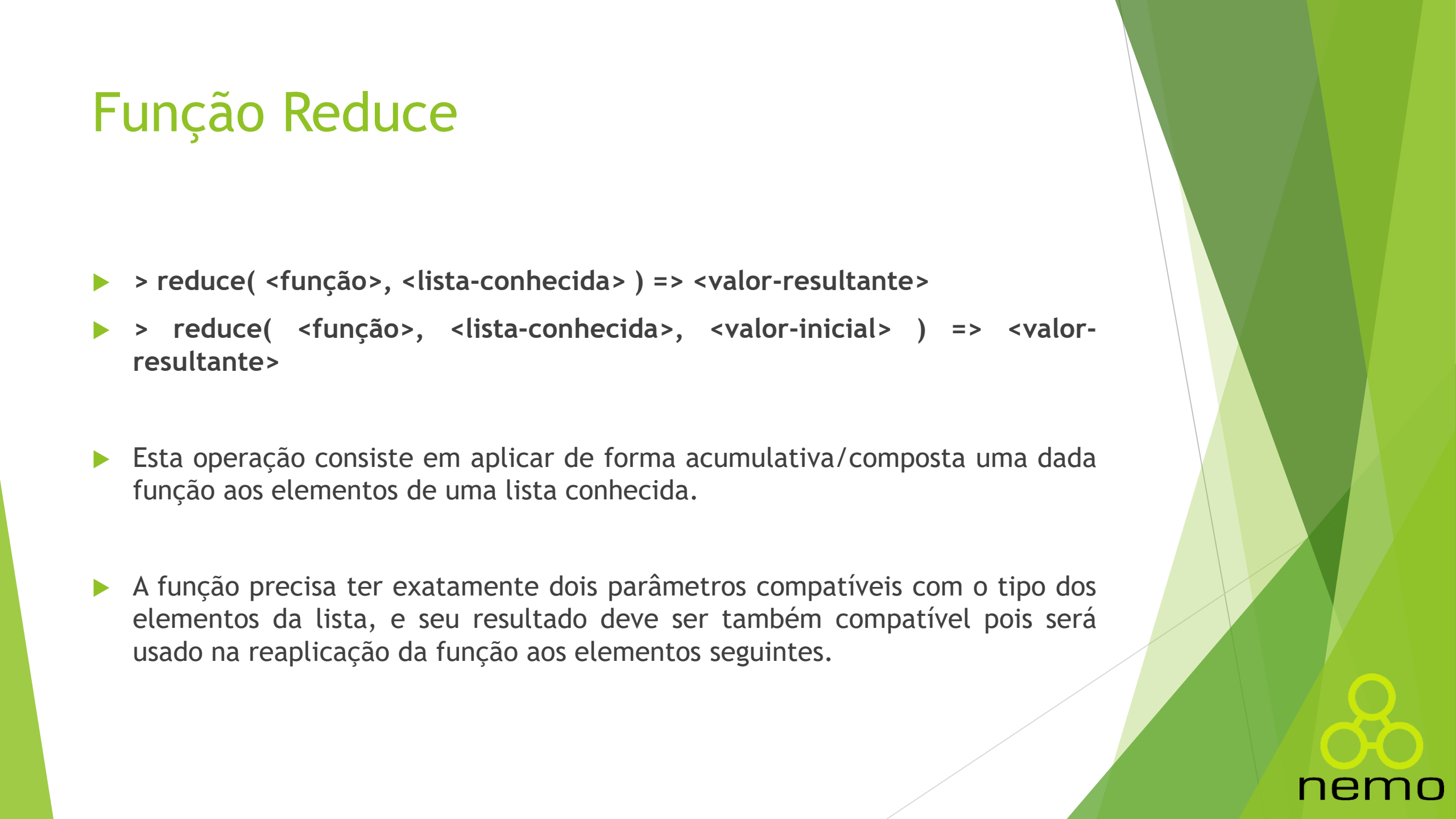
```
>>> def f(x): return x%2==0
>>> filter(f, range(1,6))
[2,4]
```

Função Reduce

- **> reduce(<função>, <lista-conhecida>) => <valor-resultante>**
- **> reduce(<função>, <lista-conhecida>, <valor-inicial>) => <valor-resultante>**

- Esta operação consiste em aplicar de forma acumulativa/composta uma dada função aos elementos de uma lista conhecida.

- A função precisa ter exatamente dois parâmetros compatíveis com o tipo dos elementos da lista, e seu resultado deve ser também compatível pois será usado na reaplicação da função aos elementos seguintes.



Função Reduce

- Exemplo:

```
>>> def f(x,y): return x+y
>>> reduce(f, range(1,6))                #f( f( f( f(1,2), 3),  4),  5 )
15
>>> reduce(f, range(1,6),50)             #f( f( f( f( f( 50,1) ,2), 3),  4),  5 )
65
```

Função Lambda (Função anônima)

- **> lambda p1, p2, ..., pn: expressão**

- Uma expressão lambda permite escrever funções anônimas, e, portanto, economizar na definição de funções simples:

```
>>> map(lambda x: 2*x, range(1,6))
[2, 4, 6, 8, 10]
>>> filter(lambda x: x%2==0, range(1,6))
[2, 4]
>>> reduce(lambda x, y: x+y, range(1,6))
15
```

nemo

Função Zip

- **> zip(lista1, lista2, ..., lista-n) -> lista-enuplas**

- Esta operação consiste produzir uma lista com enuplas formadas pelos elementos de mesma ordem de cada uma das n listas informadas como parâmetro.

- A quantidade de elementos de cada tupla é igual à quantidade de listas, e a quantidade de tuplas é igual ao tamanho da menor lista informada.

```
>>> zip(range(1,6), range(2,7), range(3,8))
[(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6), (5, 6, 7)]
>>> zip([1,2,3,4,5],[8,7,6])
[(1, 8), (2, 7), (3, 6)]
```

nemo

Exercícios

- 1) Faça uma função que receba duas listas e retorne True se são iguais ou False caso contrário.
- Duas listas são iguais se possuem os mesmos valores e na mesma ordem.

- 2) Faça uma função que receba duas listas e retorne True se têm os mesmos elementos ou False caso contrário
- Duas listas possuem os mesmos elementos quando são compostas pelos mesmos valores, mas não obrigatoriamente na mesma ordem.

- 3) Crie uma função que recebe uma lista de números e
- a. retorne o maior elemento
- b. retorne a soma dos elementos
- c. retorne o número de ocorrências do primeiro elemento da lista
- d. retorne a média dos elementos
- e. retorne o valor mais próximo da média dos elementos
- f. retorne a soma dos elementos com valor negativo
- g. retorne a quantidade de vizinhos iguais

nemo

That’s all Folks!