Programação

Funções de ordem superior

Compreensão de listas

Geradores

Fátima Leal

DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UPT UNIVERSIDADE PORTUCALENSE

Até agora …

- Módulos em Python
- Funções recursivas
- Nesta aula:
  - Funções de ordem superior
  - Geração e compreensões em listas

Funções de ordem superior – Map()

- `Map()` é uma função *builtin* do Python
- Função implementada diretamente no interpretador
- Não necessita da importação de um módulo específico
- O `map()` aplica uma função a cada elemento de uma lista
- Retornando uma nova lista contendo os elementos resultantes da aplicação da função
- Exemplo:
  - Considera a lista [1, 4, 9, 16, 25]. Pretende-se saber a raiz quadrada de cada elemento da lista

```
import math
lista = [1,4,9,16,25]
listsqrt=map(math.sqrt, lista)
print(list(listsqrt))
```

Funções de ordem superior – Reduce()

- `reduce()` era outra função *builtin*
- Passou a estar disponível no módulo `functools` a partir da versão 3000
- A utilidade está na aplicação de uma função a todos os valores do conjunto, de forma a agregá-los todos em um único valor.

- Exemplo
  - Considera a lista anterior e soma todos os seus elementos.

```
import functools as ft
import operator as o
lista = [1,4,9,16,25]
listA=ft.reduce(o.add, lista)
print(listA)
```

Funções de ordem superior – Filter()

- `filter()` filtra os elementos de uma sequência
- O processo de filtragem é definido a partir de uma função
- O resultado serão os elementos para os quais a chamada da função retornar `True`.
- Considera novamente a lista anterior.
  - Quais os valores maiores que 9?

```
def maior9(x):
    return x>9

lista=[1,4,9,16,25]
listaF=filter(maior9, lista)
print(list(listaF))
```

Funções - lambda

- Funções lambda são pequenas e anónimas

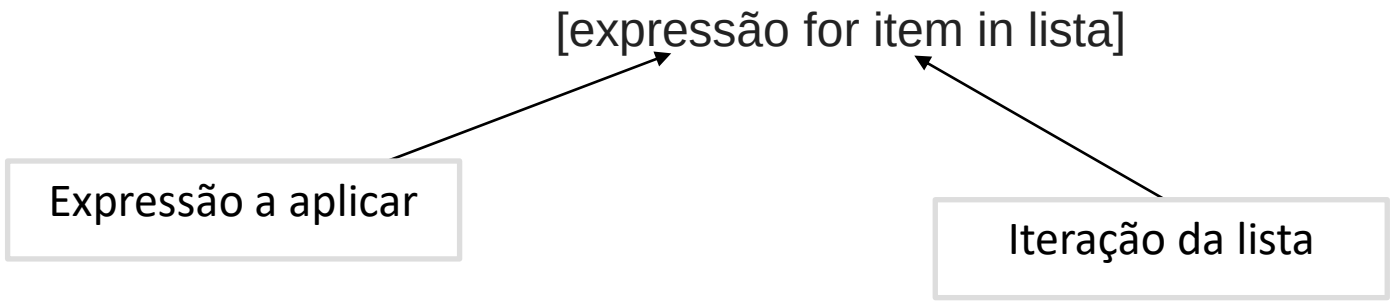
`lambda` *`arguments`* `:` *`expression`*

- Exemplo: adiciona 10 a um argumento a

```
x = lambda a : a + 10
print(x(5))
```

Compreensão de listas

- Compreensão de listas trata-se de uma outra solução de criar listas através do resultado de operações sobre listas



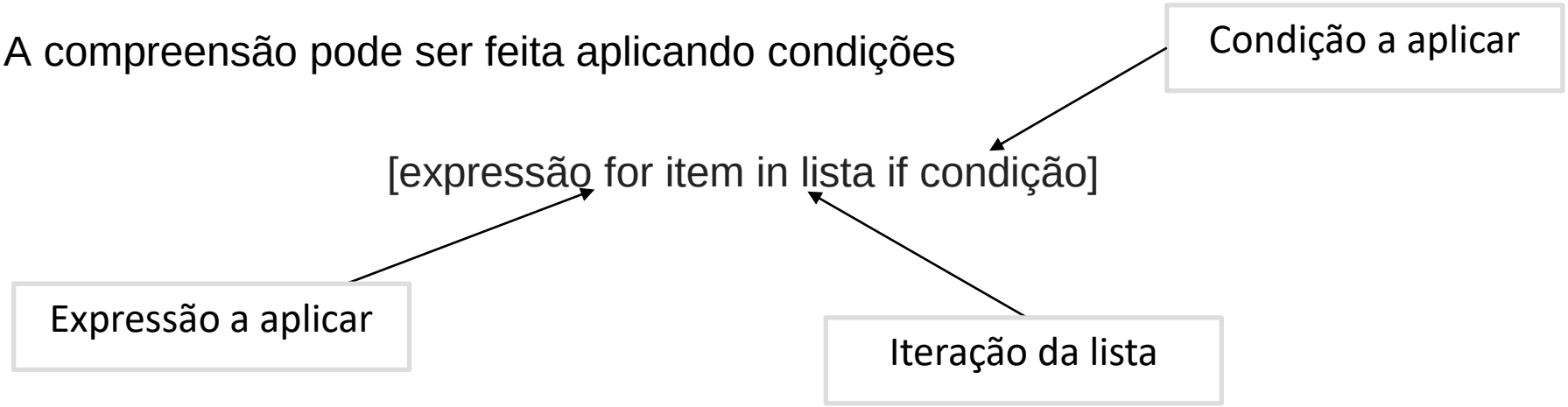
- Tomemos o exemplo de obter a raiz quadrada de cada elemento de uma lista.

```
import math
lista=[1,4,9,16,25]

listasqrt =[math.sqrt(x) for x in lista]
print(listasqrt)
```

Compreensão de listas com condições

- A compreensão pode ser feita aplicando condições



- Consideremos o exemplo:
  - Aplica a raiz quadrada aos múltiplos de 2 menores que 10

```
import math
lista=[1,4,9,16,25]
listasqrt =[math.sqrt(x) for x in lista if x<10 and x%2==0]
print(listasqrt)
```

Compreensão de listas em dicionários

- Compreensão de listas podem ser aplicadas a dicionários, alterando tanto suas chaves como valores.

- Exemplo:
  - Considera o seguinte dicionário:

    {‘a’:1, ‘b’:2, ‘c’:3, ‘d’:4, ‘e’:5}

    Para cada chave retorna o dobro do seu valor

```
dict1 = {'a':1, 'b':2, 'c':3, 'd':4, 'e':5}

dictDobro ={k:v*2 for (k,v) in dict1.items()}
print(dictDobro)
```

Geradores

- Geradores são procedimentos especiais para controlar loops e iteradores
- Uma função que usa `yield` em vez de `return` denomina-se por gerador
- São semelhantes às funções que retornam arrays

- Exemplo: Executa as diferentes funções. Quais são as diferenças?

```
def funcao():
    return 1
    return 2
    return 3
```

```
def generator():
    yield 1
    yield 2
    yield 3
```

- Cria uma função que retorna todos os números pares até 30 utilizando geradores

```
def generator():
    for x in range(31):
        if x%2==0:
            yield x
```

Geradores de expressões

- São semelhantes à compreensão mas usam () em vez de []
- Tomemos o exemplo de calcular a raiz quadrada de todos os elementos da lista [1, 4, 9, 16, 25].

```
import math
lista = [1, 4, 9, 16, 25]

listG=(math.sqrt(x) for x in lista)

print(list(listG))
```

- Quais as diferenças?
- Verifica o tipo de dados
- Quantidade de memória utilizada

Geradores de expressões

- Compreensões devolvem listas
- Geradores devolvem geradores

- Utilizam menos memória que em projetos com limitação de memória são muito importantes

```
import math
import sys
lista = [1, 4, 9, 16, 25]

listG=(math.sqrt(x) for x in lista)
listC=[math.sqrt(x) for x in lista]

print(sys.getsizeof(listG))
print(sys.getsizeof(listC))
```

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.