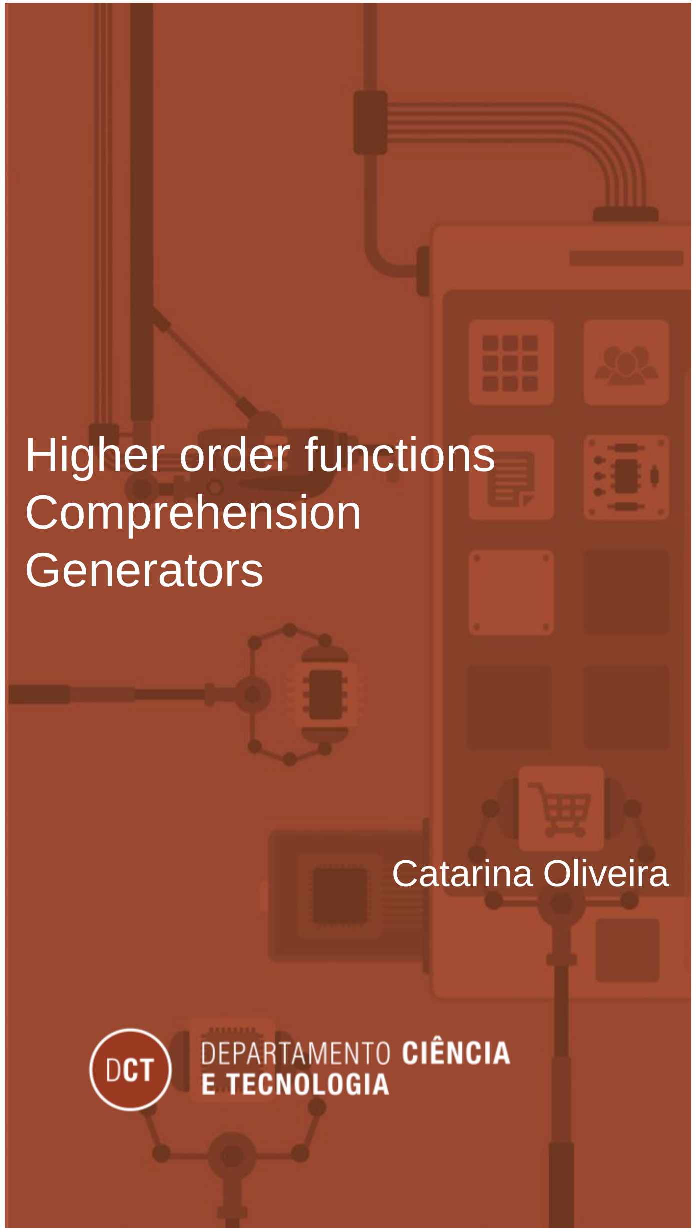

Higher order functions
Comprehension
Generators

Catarina Oliveira

DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UPT UNIVERSIDADE PORTUCALENSE

CONTENT

Higher order functions – `map ()`

Applies a function to each element of the list and returns the resulting list

Example : get the square root (`math module` , `sqrt function`) of the elements of a list

```
import math

lista = [1, 4, 9, 16, 25]

lista2 = map(math.sqrt, lista)
print(list(lista2))  # [1.0, 2.0, 3.0, 4.0, 5.0]
```

Example : obtaining the double (implement the function `dobro(x)`) of the elements of a list

```
def dobro(x):
    return x * 2

lista = [1, 4, 9, 16, 25]

lista2 = map(dobro, lista)
print(list(lista2))  # [2, 8, 18, 32, 50]
```

Higher order functions – `functools.reduce ()`

Applies a function to the elements of a list to aggregate them (module `functools`)

Example : add (`operator module` , `add function`) the elements of a list

```python
import operator
import functools

lista = [1, 4, 9, 16, 25]
resultado = functools.reduce(operator.add, lista)
print(resultado) # 55
```

Example : add (implement the function `somar( x,y )`) the elements of a list

```python
import functools

def somar(x,y):
    return x + y

lista = [1, 4, 9, 16, 25]
resultado = functools.reduce(somar, lista)
print(resultado) # 55
```

Higher order functions – `filter ()`

Filter elements from a list

Example : extract even elements (implement `ePar (x) function`) from a list

```
def ePar(x):
    return x % 2 == 0

lista = [1, 4, 9, 16, 25]
lista2 = filter(ePar, lista)
print(list(lista2))  # [4, 16]
```

lambda functions

Small and anonymous functions

Syntax:

```
lambda argumentos : resultado
```

Example : lambda function to determine the quintuple of a number

```
x = lambda a : a * 5
print(x(3)) #15
```

DEPARTAMENTO **CIÊNCIA E TECNOLOGIA**

List comprehension

Applies an **`expressao`** to each element **`x`** of a **`lista`** (similar to map)

```
[expressao for x in lista]
```

Example : get the square root (`math module` , `sqrt function`) of the elements of a list

```
import math

lista = [1, 4, 9, 16, 25]
lista2 = [math.sqrt(x) for x in lista]
print(lista2)  # [1.0, 2.0, 3.0, 4.0, 5.0]
```

Example : obtaining the double (implement the function `dobro(x)`) of the elements of a list

```
def dobro(a):
    return a * 2

lista = [1, 4, 9, 16, 25]
lista2 = [dobro(x) for x in lista]
print(lista2)  # [2, 8, 18, 32, 50]
```

Comprehension of conditional lists

Applies **`exp`** to each element **`x`** of a **`lst`** that meets a **`condicao`**

```
[exp for x in lst if condicao]
```

Example : get the square root (`math module` , `sqrt function`) of the odd elements of a list

```
import math

lista = [1, 4, 9, 16, 25]
lista2 = [math.sqrt(x) for x in lista if x % 2 == 1]
print(lista2)  # [1.0, 3.0, 5.0]
```

Example : obtaining the double (implementing the function `dobro(x)`) of the odd elements greater than 5 of a list

```
def dobro(a):
    return a * 2

lista = [1, 4, 9, 16, 25]
lista2 = [dobro(x) for x in lista if x % 2 == 1 and x > 5]
print(lista2)  # [18, 50]
```

Comprehension of dictionaries

Comprehension can also be applied to dictionaries, with the same syntax

Example : get the square root (`math module` , `sqrt function`) of the elements of a dictionary

```python
import math

d = {"a": 1, "b": 4, "c": 9, "d": 16, "e": 25}
d2 = {k: math.sqrt(v) for (k, v) in d.items()}
print(d2)
# {'a': 1.0, 'b': 2.0, 'c': 3.0, 'd': 4.0, 'e': 5.0}
```

Example : get double (implement the function `dobro(x)`) the elements of a dictionary

```python
def dobro(a):
    return a * 2

d = {"a": 1, "b": 4, "c": 9, "d": 16, "e": 25}
d2 = {k: dobro(v) for (k, v) in d.items()}
print(d2)
# {'a': 2, 'b': 8, 'c': 18, 'd': 32, 'e': 50}
```

Comprehension of conditional dictionaries

Comprehension can also be applied to dictionaries with conditions

Example : getting the square root (`math module` , `sqrt function`) of the odd elements of a dictionary

```
import math

d = {"a": 1, "b": 4, "c": 9, "d": 16, "e": 25}
d2 = {k: math.sqrt(v) for (k, v) in d.items() if v % 2 == 1}
print(d2)  # {'a': 1.0, 'c': 3.0, 'e': 5.0}
```

Example : obtaining the double (implementing the function `dobro(x)`) of the odd elements greater than 5 of a dictionary

```
def dobro(a):
    return a * 2

d = {"a": 1, "b": 4, "c": 9, "d": 16, "e": 25}
d2 = {k: dobro(v) for (k, v) in d.items() if v % 2 == 1 and v > 5}
print(d2)  # {'c': 18, 'e': 50}
```

generators

- Special procedures for controlling loops and iterators
- Functions that use `yield` instead of `return`
- Similar to functions that return arrays

Example : implement a generator to return even numbers lower than a given number

```
def geradorParesAte(limite):
    for x in range(limite + 1):
        if x % 2 == 0:
            yield x

pares10 = geradorParesAte(10)
print(list(pares10))  # [0, 2, 4, 6, 8, 10]
```

expression generator

- Similar to list comprehension, but use `()` instead of `[]`
  - Instead of list, returns generator → uses less memory

Example : get the square root (`math module` , `sqrt function`) of the elements of a list

```
import math
import sys

lista = [1, 4, 9, 16, 25]

listaG = (math.sqrt(x) for x in lista)
listaC = [math.sqrt(x) for x in lista]

print(list(listaG))
# [1.0, 2.0, 3.0, 4.0, 5.0]
print(listaC)
# [1.0, 2.0, 3.0, 4.0, 5.0]

print(sys.getsizeof(listaG)) # 104
print(sys.getsizeof(listaC)) # 120
```

```
import math
import sys

lista = [1, 4, 9, 16, 25, 36, 49, 64, 81]

listaG = (math.sqrt(x) for x in lista)
listaC = [math.sqrt(x) for x in lista]

print(list(listaG))
# [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
print(listaC)
# [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]

print(sys.getsizeof(listaG)) # 104
print(sys.getsizeof(listaC)) # 184
```

UNIVERSIDADE PORTUCALENSE

Do conhecimento à prática.