UNIVERSIDADE PORTUCALENSE

CONTEÚDO

1. Funções de ordem superior:
    1. map()
    2. functools.reduce()
    3. filter()
2. Funções lambda
3. Compreensão de listas
    1. Definição
    2. Condicional
4. Compreensão de dicionários
    1. Definição
    2. Condicional
5. Geradores
    1. Gerador de expressões

Funções de ordem superior – `map()`

Aplica uma função a cada elemento da lista e retorna a lista resultante

Exemplo: obter a raiz quadrada (módulo `math`, função `sqrt`) dos elementos de uma lista

```
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]
```

Exemplo: obter o dobro (implementar a função `dobro(x)`) dos elementos de uma lista

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

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

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

Funções de ordem superior – `functools.reduce()`

Aplica uma função aos elementos de uma lista para os agregar (módulo `functools`)

Exemplo: somar (módulo `operator`, função `add`) os elementos de uma lista

```
import operator
import functools

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

Exemplo: somar (implementar a função `soma(x,y)`) os elementos de uma lista

```
import functools

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

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

Funções de ordem superior – `filter()`

Filtra elementos de uma lista

Exemplo: extrair os elementos pares (implementar a função `ePar(x)`) de uma lista

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

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

Funções lambda

Funções pequenas e anónimas

Sintaxe:

```
lambda argumentos : resultado
```

Exemplo: função lambda para determinar o quíntuplo de um número

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

Compreensão de listas

Aplica uma **`expressão`** a cada elemento **`x`** de uma **`lista`** (de forma semelhante ao map)

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

Exemplo: obter a raiz quadrada (módulo `math`, função `sqrt`) dos elementos de uma lista

```
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]
```

Exemplo: obter o dobro (implementar a função `dobro(x)`) dos elementos de uma lista

```
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]
```

Compreensão de listas condicional

Aplica **`exp`** a cada elemento **`x`** de uma **`lst`** que obedeça a uma **`condição`**

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

Exemplo: obter a raiz quadrada (módulo `math`, função `sqrt`) dos elementos ímpares de uma lista

```
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]
```

Exemplo: obter o dobro (implementar a função `dobro(x)`) dos elementos ímpares superiores a 5 de uma lista

```
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]
```

Compreensão de dicionários

A compreensão pode também ser aplicada a dicionários, com o mesmo sintaxe

Exemplo: obter a raiz quadrada (módulo `math`, função `sqrt`) dos elementos de um dicionário

```
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}
```

Exemplo: obter o dobro (implementar a função `dobro(x)`) dos elementos de um dicionário

```
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}
```

Compreensão de dicionários condicional

A compreensão pode também ser aplicada a dicionários com condições

Exemplo: obter a raiz quadrada (módulo `math`, função `sqrt`) dos elementos ímpares de um dicionário

```
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}
```

Exemplo: obter o dobro (implementar a função `dobro(x)`) dos elementos ímpares superiores a 5 de um dicionário

```
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}
```

Geradores

- Procedimentos especiais para controlar ciclos e iteradores
- Funções que usam `yield` em vez de `return`
- Semelhantes a funções que retornam arrays

Exemplo: implementar um gerador para retornar os números pares inferiores a um determinado número

```
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]
```

Gerador de expressões

- Semelhantes à compreensão de listas, mas usam `()` em vez de `[]`
  - Em vez da lista, retorna o gerador → usa menos memória

Exemplo: obter a raiz quadrada (módulo `math`, função `sqrt`) dos elementos de uma lista

```
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.