Programação II

Exercícios 6
Funções de ordem superior

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Tecnologias da Informação

2021/2022

Nota prévia. Recorde que as funções de ordem superior que vimos (`map`, `filter` e afins) retornam iteráveis (e não listas), pelo que o seu conteúdo não pode ser diretamente observado.

```
>>> iterador = map(lambda x: 2 * x, [3, 4, 5])
>>> iterador
<map object at 0x10a4a8940>
```

No entanto, podemos testar o nosso código se convertermos primeiro o iterável numa lista. A maneira mais simples é através da função `list`.

```
>>> list(iterador)
[6, 8, 10]
```

1. Escreva expressões `lambda` para as seguintes funções:
   (a) Dobro de x
   (b) Primeiro elemento de um tuplo
   (c) Triplo da soma de x e y
   (d) Produto de três números
   (e) Concatenação de duas listas
   (f) A primeira metade de uma lista
2. Escreva uma expressão `lambda` para uma função que devolve `True` apenas quando aplicada a caracteres não brancos, isto é, quando aplicada a caracteres que não pertencem à lista `[' ', '\t', '\n']`.

3. Qual o valor de cada expressão?

   (a) `map (lambda x: x + 1, range(1, 4))`

   (b) `map (lambda x: x > 0, [3, -5, -2, 0])`

   (c) `filter (lambda x: x > 5, range(1, 7))`

   (d) `filter (lambda x: x % 2 == 0, range(1, 11))`

   (e) `filter (lambda x: x > 0, map (lambda y: y ** 2, range(-3, 4)))`

   (f) `map (lambda x: x ** 2, filter (lambda x: x > 0, range(-3, 4)))`

   (g) `map (lambda x: x + 's', ['As', 'armas', 'e', 'os', 'barões'])`

   (h) `map (lambda x: 's'+ x, ['As', 'armas', 'e', 'os', 'barões'])`

   (i) `map (lambda x: map (lambda y: y * y, x), [[1, 2], [3, 4, 5]])`

4. Defina uma função `mapa_seletivo` que recebe uma função, um predicado e um iterável, e devolve um iterável. O iterável resultado contém os resultados de aplicar a função aos elementos do iterável que satisfaçam o predicado. Os elementos que não satisfazem o predicado são descartados. Exemplo:

```
>>> list(mapa_seletivo (lambda x: x * 3, lambda y: y
    > 0, range(-4, 5)))
[3, 6, 9, 12]
```

   (a) Utilize iteradores por compreensão.

   (b) Utilize as funções **`map`** e **`filter`**.

5. Escreva a função `aplica_todas` que chama várias funções com o mesmo agumento e coleciona os resultados num iterável. Por exemplo

```
>>> list(aplica_todas([e_menor_100, e_maior_10,
    e_primo], 71))
[True, True, True]
```

   (a) Utilize iteradores por compreensão.

   (b) Utilize a função **`map`**.

6. Escreva uma função `maior_zero` que transforme um iterável de iteráveis de inteiros num iterável de iteráveis de valores lógicos. Cada entrada no iterável resultante indica se o valor na respetiva posição do iterável original é ou não maior do que zero. Lembre-se que um iterável é qualquer objeto que possa ser convertido num iterador. Por exemplo:

```
>>> [list(it) for it in maior_zero([[-1, -2, 3], [2,
    -1, 3, 7]])]
[[False, False, True], [True, False, True, True]]
```

(a) Utilize iteradores por compreensão.

(b) Utilize a função **map**.

7. Defina uma função `produto_escalar` que calcule o produto escalar $\sum_{i=1}^{n} x_i \cdot y_i$ de dois vetores $\vec{x}$ e $\vec{y}$. Os vetores são dados por iteráveis de números com o mesmo comprimento. Utilize as funções **sum** e **map**.

8. Dado um par de iteráveis, a função **zip** devolve um iterável sobre pares. O i-ésimo par é composto pelo i-ésimo elemento do primeiro iterável e pelo i-ésimo do segundo iterável. O iterável resultante contém tantos elementos quantos os do mais curto dos dois iteráveis parâmetro. Neste exercício estamos interessados numa função `zip_with`, variante da função **zip**, que recebe uma função que combina os dois elementos.

```
def zip_with(f, lista1, lista2):
    """
    >>> list(zip_with(lambda x, y: x, [], [1, 2]))
    []
    >>> list(zip_with(lambda x,y : x, ['a'], []))
    []
    >>> list(zip_with(lambda x, y: (x, y),
        ['a', 'b'], [2, 3, 4]))
    [('a', 2), ('b', 3)]
    >>> list(zip_with(max, [5, 2, 0, 9], [2, 3, 4]))
    [5, 3, 4]
    """
```

(a) Escreva a função utilizando iteradores por compreensão e a função **zip**.

(b) Escreva uma variante recorrendo às funções **zip** e **map**.

(c) Proponha uma solução recorrendo apenas à função **map** (a função **map** pode ser utilizada com mais do que um iterador).

9. Escreva a função **zip** recorrendo à função **map** (a função **map** pode ser utilizada com mais do que um iterador).

10. Determine o valor de cada uma das expressões seguintes.

(a) `reduce (operator.mul, range(-3, 0, 1), 1)`

(b) `reduce (operator.mul, range(-3, 0, -1), 1)`

(c) `reduce (operator.sub, [1, 2, 3])`

(d) `reduce (operator.sub, [1, 2, 3], 10)`

(e) `reduce (lambda acc, z: acc * 3 + z, range(1, 5))`

(f) `reduce (lambda acc, y: acc + y if acc > 0 else y, [4, -3, -2, -1])`

(g) `reduce (lambda acc, y: acc ** 2 + y, range(5))`

Não se esqueça de fazer `import operator`[1] para as primeiras quatro alíneas. A função `reduce` pode ser importada do módulo `functools`[2].

11. Recorrendo à função `reduce`, escreva uma função que calcule o factorial de um número não negativo.

12. Escreva uma função `aplica` que, dada um iterável sobre funções e um iterável sobre elementos, devolve o iterável resultante de aplicar sucessivamente as funções do iterável de funções aos valores do iterável de argumentos. Utilize as funções `map` e `reduce`. Exemplo, onde 5 resulta de multiplicar 1 por 2 e em seguida somar-lhe 3:

```
>>> list(aplica([lambda x: x*2, lambda x: x+3], [1,
    3, 0, 4]))
[5, 9, 3, 11]
```

13. Escreva um conversor de um número em representação binária para um número em representação decimal. O número binário é apresentado por uma lista de inteiros. Utilize a função `reduce`. Exemplos:

```
>>> binario_para_decimal([1, 1, 0, 1])
13
>>> binario_para_decimal([])
0
```

14. Escreva a função `filter` recorrendo às funções `map` e `reduce`. Sugestão através de um exemplo: para o predicado `lambda x: x > 0` e a lista `[2, 0, -3, 4]` comece por calcular a lista `[[2], [], [], [4]]`. Concatene depois as listas todas para obter `[2, 4]`.

15. Escreva a função `filter` recorrendo apenas à função `reduce`.

16. Escreva a função `map` recorrendo apenas à função `reduce`.

17. Defina as seguintes funções:

    (a) a função `total`, de modo a que `total(f, n)` seja `f(0)+ f(1)+ ... + f(n)`. Por exemplo:

[1] Documentação: `https://docs.python.org/3/library/operator.html`
[2] Documentação: `https://docs.python.org/3/library/functools.html`

```
>>> total(lambda x: x ** 2, 4)
30
```

(b) a função `total_superior`, de modo a que `total_superior(f)` é a função que, no ponto `n`, retorna `f(0)+ f(1)+ ... + f(n)`. Exemplo:

```
>>> total_superior(lambda x: x ** 2)(4)
30
```

18. A função `take` retorna os primeiros `n` elementos de uma lista. Escreva esta função recorrendo à função **`islice`** do módulo `itertools`[3].

19. A função `drop` retorna os últimos `n` elementos de uma lista. Escreva esta função recorrendo à função **`islice`**.

20. Quando trabalhamos com iteradores, frequentemente precisamos de saber em que iteração estamos. O Python fornece uma função **`enumerate`** que adiciona um contador a um objeto iterável.

```
>>> compras = ['pão', 'leite', 'chocolate']
>>> list(enumerate(compras))
[(0, 'pão'), (1, 'leite'), (2, 'chocolate')]
```

Escreva a função **`enumerate`** recorrendo à função **`zip`** e à função **`count`** do módulo `itertools`.

21. Na pasta Documentos do Moodle pode encontrar um ficheiro `olimpicos.py` que possui informação sobre os atletas que participaram nos Jogos Olímpicos de 2016 no Rio de Janeiro, sob a forma de uma lista de tuplos `atletas`. Descarregue este ficheiro para o seu computador. Pode obter a lista de tuplos através da instrução

```
>>> from olimpicos import atletas
```

Verifique que a tabela tem 13688 linhas e que a primeira linha da tabela é

```
>>> atletas[0]
(22, 'Andreea␣Aanei', 'F', 22, 170.0, 125.0, 'Romania
    ', 'ROU', 'Weightlifting', "Weightlifting␣Women's
    ␣Super-Heavyweight", 'NA')
```

Os vários campos de cada linha significam, por ordem: ID do atleta, nome do atleta, sexo, idade, altura, peso, país de origem, código do país de origem, desporto, evento, e medalha (`'NA'`, `'Bronze'`, `'Silver'` ou `'Gold'`). Utilizando a função **`groupby`** do módulo `itertools`, responda às seguintes questões:

[3] Documentação: `https://docs.python.org/3/library/itertools.html`

(a) Quantas medalhas foram alcançadas pelos diferentes países?

(b) Qual foi a atleta (mulher) mais medalhada?

(c) Quantas medalhas de Ouro, Prata, e Bronze, foram alcançadas pela Espanha?

(d) Quantos atletas portugueses medem 160cm de altura ou menos? Nota: conte cada atleta apenas uma vez.

(e) Que país competiu com a equipa Olímpica mais leve (em média)? Nota: Descarte atletas para os quais a informação sobre o peso não existe, `'NA'`.

22. Na pasta Documentos do Moodle pode encontrar um ficheiro `movimentos.py` que possui informação sobre várias transações de um agregado familiar entre junho de 2019 e junho de 2020. Esta informação encontra-se sob a forma de uma lista de tuplos `movimentos`. Descarregue este ficheiro para o seu computador. Pode obter a lista de tuplos através da instrução

```
>>> from movimentos import movimentos
```

Verifique que a tabela tem 2000 linhas e que a primeira linha da tabela é

```
>>> print(movimentos[0])
('Francisco', -13.06, '2019-06-20', 'curso')
```

Os vários campos de cada linha significam, por ordem: nome da pessoa, valor da transação, data de execução, e categoria. Utilizando a função **`groupby`** do módulo `itertools`, responda às seguintes questões:

(a) Quanto dinheiro foi gasto (despesa) em cada categoria?

(b) Quanto dinheiro foi obtido (receita) por cada pessoa?

(c) Qual o saldo (receita menos despesa) do Francisco durante cada mês de 2019?