Computação 1, 2020.1
Lista 7

Para entrega até 8/2 às 10:00

Submeta suas soluções colocando os arquivos correspondentes na sua pasta do Google Drive

Lembre-se de escolher bons nomes para suas funções e variáveis, de documentar seu código com *docstrings* (documentação de função) e comentários onde for apropriado, e de fazer testes para suas funções (na documentação usando o módulo `doctest` ou em funções dedicadas).

Questão 1. Transforme os seguintes programas em *list comprehensions*

a.

```
L1 = []
for i in range(10):
    L1.append(i**2+3)
```

b.

```
L2 = [5,4,3,2,1]*3
L3 = []
for i in L2[2:8]:
    if 2**i < 99:
        L3.append(i**2+3)
```

Questão 2. Explique porque cada um dos programas abaixo não pode ser tão simplesmente convertido para *list comprehensions*

a.

```
L4 = []
for i in range(10):
    s = sum(L4)
    L4.append(i**2+s)
```

b.

```
L5 = [1]
for i in range(-10,10,3):
    if 2**i < 99:
        L5.append(i**2+3)
L5.extend([1000, 2000])
```

Questão 3 (Merge sort)**.**

a. Faça uma função em `python` que receba como entrada duas listas ordenadas e retorne seu "*merge*": uma lista ordenada cujos elementos são exatamente os elementos das listas dadas como entrada.

b. Implemente em `python` o seguinte algoritmo, chamado *merge sort*: dada uma lista `L` a ser ordenada, se `L` é vazia ou tem tamanho 1 não há nada se fazer a não ser retornar `L`; nos outros casos, **recursivamente** (ou seja, usando o próprio merge sort) ordenamos a primeira e a segunda metades de `L`, e depois retornamos o *merge* das metades já ordenadas.

Questão 4 (Matrizes)**.** Podemos representar uma matriz real M com ℓ linhas e c colunas,

$$M = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & \cdots & m_{0,c-1} \\ m_{1,0} & m_{1,1} & m_{1,2} & \cdots & m_{1,c-1} \\ m_{2,0} & m_{2,1} & m_{2,2} & \cdots & m_{2,c-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{\ell-1,0} & m_{\ell-1,1} & m_{\ell-1,2} & \cdots & m_{\ell-1,c-1} \end{pmatrix},$$

em `python` como uma lista `L` de comprimento ℓ, onde cada elemento é por sua vez uma lista de comprimento c, de forma que dados $i < \ell$ e $j < c$ tenhamos

$$\texttt{L}[i][j] = \tilde{m}_{i,j},$$

sendo $\tilde{m}_{i,j}$ o `float` correspondente ao real $m_{i,j}$.

Em outras palavras,

$$\begin{aligned} \texttt{L} = [&[\tilde{m}_{0,0}, \tilde{m}_{0,1}, \tilde{m}_{0,2}, \ \ldots, \tilde{m}_{0,c-1}], \\ &[\tilde{m}_{1,0}, \tilde{m}_{1,1}, \tilde{m}_{1,2}, \ \ldots, \tilde{m}_{1,c-1}], \\ &[\tilde{m}_{2,0}, \tilde{m}_{2,1}, \tilde{m}_{2,2}, \ \ldots, \tilde{m}_{2,c-1}], \\ &\ldots, \\ &[\tilde{m}_{\ell-1,0}, \tilde{m}_{\ell-1,1}, \tilde{m}_{\ell-1,2}, \ \ldots, \tilde{m}_{\ell-1,c-1}]]. \end{aligned}$$

(Note: essa forma de representar uma matriz privilegia as linhas em detrimento das colunas; poderíamos muito bem ter escolhido uma forma que privilegiasse as colunas em detrimento das linhas.)

a. Faça uma função que receba uma matriz na forma descrita acima e retorne a quantidade de linhas da matriz.

b. Faça uma função que receba uma matriz na forma descrita acima e retorne a quantidade de colunas da matriz.

c. Faça uma função que receba uma matriz na forma descrita acima e um número inteiro i e retorne a i-ésima linha da matriz dada, caso exista, ou algum tipo de erro informativo em caso contrário[1].

[1] No caso de erro, "retornado" deve ser entendido em sentido amplo, não necessariamente usando um `return`. O mesmo comentário vale para as questões seguintes.

d. Faça uma função que receba uma matriz na forma descrita acima e um número inteiro j e retorne a j-ésima coluna da matriz dada, caso exista, ou algum tipo de erro informativo em caso contrário.

e. Faça uma função que receba uma matriz na forma descrita acima e retorne sua transposta.

f. No mesmo espírito, podemos representar um vetor real

$$v = (v_0, v_1, v_2, \ldots, v_{n-1})$$

em `python` usando a lista

$$[\tilde{v}_0,\ \tilde{v}_1,\ \tilde{v}_2,\ \ldots,\ \tilde{v}_{n-1}],$$

onde novamente $\tilde{v}_i$ é o `float` que representa o real v_i.

Dados dois vetores reais de mesmo comprimento $u = (u_0, \ldots, u_{n-1})$ e $v = (v_0, \ldots, v_{n-1})$, seu *produto escalar* é o número real

$$\sum_{i=0}^{n-1} (u_i \cdot v_i).$$

Faça uma função que receba como entrada dois vetores (representados como acima) e retorne seu produto escalar, caso exista, ou algum tipo de erro informativo em caso contrário.

g. O *produto* de duas matrizes reais M e N existe quando (e apenas quando) o número de colunas de M é igual ao número de linhas de N. Neste caso, o produto é a matriz real cujo número de linhas coincide com o de M e cujo número de colunas coincide com o de N, e tal que o real que ocupa a i-ésima linha e j-ésima coluna é o produto escalar da i-ésima linha de M com a j-ésima coluna de N (vistos como vetores). Implemente essa operação em `python`, usando as representações descritas acima; caso o produto das matrizes dadas não exista, novamente, algum tipo de erro informativo deve ser retornado.

Desafio (opcional)

Questão 5 (Números surreais)**.** Os *números surreais* (inventados pelo recém falecido John H. Conway, mas nomeados por Don Knuth) são uma *enorme* generalização dos números reais (principalmente da construção de reais por cortes de Dedekind), mas que ainda se comportam como os reais de diversas formas. Nesta questão trataremos apenas de alguns surreais—aqueles que têm alguma representação finita.

Definimos números surreais, bem como relações de ordem $\leq$ e igualdade $=$ entre eles, por recursão:

1. para todos conjuntos E e D de números surreais (já definidos) tais que

   $$\forall e \in E\, \forall d \in D: \quad e \leq d \ \ \&\ \ e \neq d,$$

   o par (E, D) é um número surreal.[2] Os conjuntos E e D são respectivamente chamados de *parte esquerda* e *parte direita* do surreal (E, D). (O caso base é dado por $E = D = \varnothing$.)

[2] Intuitivamente, imaginamos o surreal (E, D) como sendo o número mais "simples" possível que é estritamente maior que todos os números em E e estritamente menor que todos os números em D.

2. para surreais $x = (E_x, D_x)$ e $y = (E_y, D_y)$, definimos $x \leq y$ sse

$$(\forall e_x \in E_x : \quad y \not\leq e_x) \quad \& \quad (\forall d_y \in D_y : \quad d_y \not\leq x)$$

3. para surreais x e y, definimos $x = y$ sse $x \leq y$ e $y \leq x$.

(Atenção: pelo que chamamos *vacuidade*, se S é um conjunto vazio, então **qualquer** fórmula que comece com um quantificador do tipo "$\forall s \in S \ldots$" é **verdadeira**!)

Assim, $(\varnothing, \varnothing)$ é um número surreal que chamaremos (sugestivamente) de $\bar{0}$, e os seguintes números também são surreais:

- $\bar{1} = (\{\bar{0}\}, \varnothing)$;
- $\bar{2} = (\{\bar{1}\}, \varnothing)$;
- $\frac{\bar{1}}{\bar{2}} = (\{\bar{0}\}, \{\bar{1}\})$;
- $-\bar{1} = (\varnothing, \{\bar{0}\})$;

etc.

Representaremos surreais em `python` usando tuplas e listas: um surreal é uma tupla de tamanho 2, cujo primeiro elemento é uma lista de surreais (a parte esquerda do surreal) e cujo segundo elemento também é uma lista de surreais (a parte direita do surreal). Assim, $\bar{0}$ pode ser representado por `zero = ([],[])`, $\bar{1}$ por `um = ([zero],[])`, i.e., `um = ([([],[])],[])`, $\bar{2}$ por `dois = ([um],[]) = ([([([],[])],[])],[])`, etc.

a. Faça duas funções que recebam (cada) um surreal como entrada e retornem a lista de surreais na parte esquerda ou direita, respectivamente, do surreal dado.

b. Faça uma função que receba como entrada dois surreais e retorne `True` caso o primeiro surreal seja menor ou igual ao segundo, `False` em caso contrário.

c. Faça uma função que receba dois surreais e retorne `True` caso sejam iguais, `False` em caso contrário. Verifique que os surreais $(\varnothing, \varnothing)$ e $(\{-\bar{1}\}, \{\bar{1}\})$ são iguais.

d. Para um subclasse dos surreais, incluindo todos os surreais com representação finita (os que nos interessam nessa questão), é verdade que cada um deles tem uma representação na forma (E, D) com E e D conjuntos com 0 ou 1 elemento cada. Faça uma função que receba como entrada um surreal (portanto, como estamos representando no computador, o surreal dado estará necessariamente nesta subclasse) e retorne uma representação deste mesmo surreal com partes esquerda e direita vazias ou unitárias. *Dica:* use esta função nos próximos itens.

e. A *soma* de surreais $x = (E_x, D_x)$ e $y = (E_y, D_y)$ é o surreal que tem uma representação com parte esquerda

$$\{x + e_y \ ; \ e_y \in E_y\} \cup \{y + e_x \ ; \ e_x \in E_x\}$$

e com parte direita

$$\{x + d_y \ ; \ d_y \in D_y\} \cup \{y + d_x \ ; \ d_x \in D_x\}$$

Note que esta definição é recursiva (e as próximas também o serão).

Implemente a adição de surreais em `python`. Verifique que $\bar{1} + \bar{1} = \bar{2}$, i.e., que `igual(soma(um,um),dois) == True`.

f. O *simétrico* $-x$ de um surreal $x = (E_x, D_x)$ é o surreal que tem uma representação com parte esquerda

$$\{-d_x \ ; \ d_x \in D_x\}$$

e parte direita

$$\{-e_x \ ; \ e_x \in E_x\}.$$

Implemente essa operação em `python`. Verifique que `igual(simetrico(um),menos_um) == True`.

g. A *subtração* entre surreais x e y é definida como a soma de x com o simétrico de y. Implemente essa operação em `python`. Verifique que $\overline{1} - \overline{1} = \overline{0}$, i.e., que `igual(subtracao(um,um),zero) == True`.

h. O *produto* de surreais $x = (E_x, D_x)$ e $y = (E_y, D_y)$ é o surreal que tem uma representação com parte esquerda

$$\begin{aligned}\{e_x \cdot y \ + \ x \cdot e_y \ - \ e'_x \cdot e'_y \ \ ; \ \ e_x, e'_x \in E_x \ \wedge \ e_y, e'_y \in E_y\} \\ \cup \{d_x \cdot y \ + \ x \cdot d_y \ - \ d'_x \cdot d'_y \ \ ; \ \ d_x, d'_x \in D_x \ \wedge \ d_y, d'_y \in D_y\}\end{aligned}$$

e parte direita

$$\begin{aligned}\{e_x \cdot y \ + \ x \cdot d_y \ - \ e'_x \cdot d'_y \ \ ; \ \ e_x, e'_x \in E_x \ \wedge \ d_y, d'_y \in D_y\} \\ \cup \{x \cdot e_y \ + \ d_x \cdot y \ - \ d'_x \cdot e'_y \ \ ; \ \ d_x, d'_x \in D_x \ \wedge \ e_y, e'_y \in E_y\}.\end{aligned}$$

Implemente a multiplicação de surreais em `python`. Verifique que $\overline{2} \cdot \frac{\overline{1}}{2} = \overline{1}$, i.e., que `igual(produto(dois,meio),um) == True`.