Computação 1, 2021.1
Lista 10

Data limite para entrega: 05/10 às 23:59

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 (Fibonacci negativos)**.** Em aula, definimos os números de Fibonacci através da recorrência

$$\begin{cases} F_0 & = 0 \\ F_1 & = 1 \\ F_{n+2} & = F_{n+1} + F_n \text{ para todo } n \geq 0. \end{cases}$$

Vamos generalizar a função `Fibonacci(n)` para que ela aceite um argumento negativo.

a. Qual deve ser o valor de F_{-1} para ainda valer a fórmula de recorrência?

b. Qual fórmula de recorrência usar para calcular F_{-n}?

c. Escreva o código correspondente à esta função "Fibonacci negativo", i.e., a função que recebe um natural $n > 0$ e retorna o valor F_{-n}.

d. Agora, escreva uma função "geral" de Fibonacci, que funcione para todos os argumentos inteiros.

Questão 2 (Reescrevendo expressões)**.** Em matemática, o uso de reticências "..." em expressões é bastante comum; por exemplo, a função $f : \mathbb{N} \to \mathbb{N}$ dada por

$$f(n) = \text{ a soma dos } n \text{ primeiros números naturais}$$

é comumente escrita da forma

$$f(n) = 1 + 2 + 3 + \cdots + (n-1). \qquad (\star)$$

Entretanto, o uso de reticências pode causar problemas de incerteza e ambiguidade, pois assume que o leitor será capaz de *deduzir* o conteúdo ocultado pelas

*Link recebido por email em 19/7/2021 — o nome é parecido com `<seu nome> - Computação 1 - Submissões e Feedback`.

reticências, o que pode não ser imediato. Por exemplo, é bem questionável deduzir $f(0) = 0$ a partir da expressão $(\star)$.

Em geral, o uso de reticências esconde uma definição recursiva; *formalmente*, a função f acima é definida por

$$\begin{cases} f(0) = 0 \\ f(n) = f(n-1) + (n-1), \quad \text{para } n > 0. \end{cases}$$

Em cada item abaixo, reescreva a expressão que define $g : \mathbb{N} \to \mathbb{N}$ de forma recursiva, sem o uso de reticências (nem de *somatórios*, *produtórios* ou afins). Esta é uma questão de definições, ou seja, não escreva uma função em Python, mas dê a fórmula *matemática* da recorrência para cada caso.

Lembrete: nesta disciplina, 0 é o menor número natural.

a. $g(n) = 1^2 + 2^2 + \cdots + (n+1)^2$

b. $g(n) = \dfrac{1}{1 \cdot 2} + \dfrac{1}{2 \cdot 3} + \cdots + \dfrac{1}{(n+1)(n+2)}$

c. $g(n) = 2 \cdot 4 \cdot 6 \cdots (2(n+1))$

d. $g(n) = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdots p$, onde p é o *maior* primo tal que $p \leq n$. (Logo $g(3) = 6 = g(4)$ e $g(5) = 30$, por exemplo. Qual deve ser a definição do caso base para que a definição recursiva *funcione bem*?) Você pode usar a expressão "x é primo" na definição da sua função, onde x é qualquer número natural que seja útil para você.

e. $g(n) = \displaystyle\sum_{i=0}^{n-1} f(i) = f(0) + f(1) + f(2) + \cdots + f(n-1)$, onde $f : \mathbb{N} \to \mathbb{N}$ é uma função dada. (A sua resposta pode e deve usar f na definição recursiva de g).

Questão 3 (Dígitos de `int`s)**.** Nesta questão, utilize apenas variáveis de tipo `int` do Python!

a. Implemente uma função que receba um natural n e retorne seu último dígito.

b. Implemente uma função que receba um natural $n \geq 10$ e retorne o número n sem seu último dígito.

c. Implemente uma função **recursiva** que receba um natural n e retorne a soma dos dígitos de n.

Questão 4 (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.

Observações. 1: esta será uma das etapas do algoritmo de ordenação a seguir, portanto, não use `sort` ou outra função equivalente. 2: você não precisa verificar que as listas de entrada estão ordenadas.

b. Implemente em Python o seguinte algoritmo recursivo para ordenar uma lista `L`, chamado *merge sort*: o caso base é, como no `quicksort`, para listas vazias ou de tamanho 1; nos outros casos, recursivamente (e separadamente) ordenamos a primeira e a segunda metades de `L`, e depois retornamos o *merge* das metades já ordenadas.

Desafio (opcional)

Questão 5 (Torre de Hanói de 4 pinos)**.** Suponha que você tem 4 pinos para jogar a Torre de Hanói. Isso deveria reduzir o número de passos necessários. Suponha que temos $N = n(n+1)/2$ discos.

a. Mostre que é possível (i.e., que está correto) usar o caso de 3 pinos como "caso base" da seguinte recursão:

1. Transferir $n(n-1)/2$ discos do pino `origem` para o pino `auxiliar_1`, usando os 4 pinos
2. Transferir os n maiores discos do pino `origem` para o pino `destino`, usando apenas os pinos `origem`, `destino` e `auxiliar_2`
3. Transferir de volta os $n(n-1)/2$ discos mais leves do pino `auxiliar_1` para o pino `destino`, usando os 4 pinos,

b. Implemente o algoritmo acima em `python`.