Computação 1, 2020.1
Lista 5

Para entrega até 25/1 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. 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 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 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). Lembrete: nesta disciplina, 0 é o menor número natural.

*Link recebido por email em 7/12/2020 — o nome é parecido com `<seu nome> - Computação 1 - Submissões e Feedback.`

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

b. $g(n) = 1 + 3 + 5 + \cdots + (2n+1)$

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

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

e. $g(n) = 2\cdot 3\cdot 5\cdot 7\cdot 11\cdot 13\cdot 17\cdots p$, 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.

f. $g(n) = \displaystyle\sum_{i=0}^{n-1} f(i) = f(0) + f(1) + f(2) + \cdots + f(n-1)$, 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. Na aula Teórica 5, vimos que a conjectura de Collatz (abaixo) é verdadeira para todos os naturais n até um certo número muito grande L.

Conjectura de Collatz. *Para todo natural $n \geq 2$, iterar o processo*

$$n \mapsto \begin{cases} \dfrac{n}{2}, & \text{se } n \text{ é par} \\ 3n+1, & \text{se } n \text{ é impar} \end{cases} \qquad (\dagger)$$

chega a 1 após um número finito de iterações.

Implemente, **de forma recursiva**, uma função que receba um número natural n (que assumimos satisfazer $2 \leq n < L$ — não precisa verificar isto) e retorne o número de iterações necessárias do processo (†) acima, começando em n, até que se atinja o valor 1.

Questão 4.

a. Implemente uma função que receba um natural n e retorne o último dígito de n (sem converter n para nenhum outro tipo!)

b. Implemente uma função que receba um natural $n \geq 10$ e retorne o número n sem seu último dígito (sem converter n para nenhum outro tipo!)

c. Implemente uma função **recursiva** que receba um natural n e retorne a soma dos dígitos de n. Não utilize nenhum tipo de dado de Python que não sejam números `int`s.

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:

b.

Transferir $n(n-1)/2$ discos do pino `origem` para o pino `auxiliar_1`, usando os 4 pinos

Transferir os n maiores discos do pino `origem` para o pino `destino`, usando apenas os pinos `origem`, `destino` e `auxiliar_2`

Transferir de volta os $n(n-1)/2$ discos mais leves do pino `auxiliar_1` para o pino `destino`, usando os 4 pinos,

c. Implemente o algoritmo acima em `python`.