Computer Science 320 (Fall, 2016)
Concepts of Programming Languages

Homework Assignment 1: Python vs. Haskell Warm-Up Exercises

Out: Tuesday, September 6, 2016
Due: Wednesday, September 21, 20162

(20 pts) **Problem 1.** In the file PSet1.tar.gz, there are four scripts, two in Python and two in Haskell.

(a) Convert the two Haskell scripts to Python scripts.

(b) Convert the two Python scripts to Haskell scripts.

In both parts you should preserve the structure of the program, *e.g.*, recursive code should remain recursive in the converted script and nested function definitions should remain nested in the converted script.

(20 pts) **Problem 2.** The formula for the nth Tetranacci number T_n is defined as follows:

$$\begin{aligned} T_0 &= 0 \\ T_1 &= 1 \\ T_2 &= 1 \\ T_3 &= 2 \\ T_n &= T_{n-1} + T_{n-2} + T_{n-3} + T_{n-4}. \end{aligned}$$

Your task is to implement a recursive function which accepts an integer n (you may assume that $n \geq 0$), and computes the nth Tetranacci number (don't worry about efficiency, only about making the definition as simple as possible).

(a) Implement the requested function in Python.

(b) Implement the requested function in Haskell. In the Haskell code, insert a type that ensures that the function accepts and returns only arguments of type `Int`.

In both (a) and (b), give an estimate of the number of times the function will be called on an input of size n.

(30 pts) **Problem 3.** A more efficient version of the function for computing Tetranacci numbers can be defined by following three steps:

(i) Implement a function which takes a list of integers and adds the sum of the top four elements to the head of the list (*e.g.*, in Haskell, `1:1:1:1:nil` should become `4:1:1:1:1:nil`).

(ii) Implement a recursive function which accepts an integer n as input (again, assume $n \geq 0$), and returns a list of integers from 0 to n in ascending order.

(iii) Implement a recursive function which computes the nth Tetranacci number in linear time. This function may use a linear amount of space to compute its result, but the number of recursive calls it makes must be linear in n.

Your task is to follow the three steps above, in order to:

(a) Implement the requested function in Python.

(b) Implement the requested function in Haskell. Again, in the Haskell code, insert a type that ensures that the function accepts and returns only arguments of type `Int`.

(30 pts) **Problem 4.** The Fibonacci k-step numbers are a generalization of the Fibonacci and Tetranacci sequences, where $F_i = 0$ for $i \leq 0$, $F_1 = 1$, $F_2 = 1$, and F_j for $j \geq 3$ is the sum of the k previous numbers in the sequence. We want you to implement a function for computing these numbers efficiently, in two steps:

(i) Implement a function which accepts an integer k along with a list of integers, and sums up the first k elements of the list. If the list has length less than k, simply sum all the elements in the list (assume that an empty list evaluates to a sum of 0).

(ii) Implement a function that accepts two integers n and k, and computes in time $O(nk)$ the nth Fibonacci k-step number.

Based on the preceding two steps:

(a) Implement the two requested functions in Python.

(b) Implement the two requested functions in Haskell.

In the Haskell code in part (b), insert types liberally to ensure that the functions accept and return only arguments of type `Int`.