COMP 141: Haskell — Part 3

Instructions: In this exercise, we are going to review a bunch of Haskell structures.

In the first three questions, we are aiming to define greatest common divisor of two numbers, in a step-by-step fashion.

(1) Define function `divisors` that receives a number and returns a list of all divisors of that number. For example, if the input is `20`, then the output would be `[1,2,4,5,10,20]`. You may use list comprehension, list ranges, and function `mod` for this purpose.

(2) Define function `commonDiv` that receives two numbers as input, and returns the list of all common divisors of those two input numbers. For example if the inputs are `24` and `30`, the output would be `[1,2,3,6]`. You may use list comprehension and function `divisor` defined earlier.

(3) Define function `greatestCommonDiv` that receives two numbers as input and returns the greatest common divisor of those two input numbers. For example, if the inputs are `24` and `30`, then the output would be `6`. You may use `maximum` function along with `commonDiv` defined earlier.
Note: function `gcd` is already part of standard library. Do not use that!

(4) Define function `sumProduct` that receives a list of numbers and returns a pair, where the first component is the summation of list elements, and the second component is the multiplication of list elements. For example, if the input is `[2,4,3,5]` then the output is `(14, 120)`.

(5) Define function `pyth` that receives a list triples of numbers and filters the ones that satisfy Pythagorean theorem. That is, all (a, b, c) where $a^2 = b^2 + c^2$. For example if the input list is `[(1, 6, 2), (10, 8, 6), (4,2,1), (5,3,4)]`, then the output would be `[(10,8,6), (5,3,4)]`. You may use list comprehension for this purpose.

(6) Use list comprehensions to define function `trimAlpha` that receives a string and eliminates every alphabetic (uppercase and lowercase) characters from that string. For example, if the input is `"Ks%bU#n9x"` then the output must be `"%#9"`.

(7) Define function `cartesian3` that receives three lists as input and creates the cartesian product list of them.

(8) Let's assume we have a database of people names associated with their age and nationality. We can simply show that with a list of triples as below:

```
ppl = [ ("joe", 26, "usa"), ("amy", 18, "uk"), ("greg", 20, "usa"), ("ed",
25, "canada"), ("joan", 17, "usa"), ("fred", 19, "usa"), ("carla", 20,
"germany")]
```

Use list comprehensions to define function `american` that ranges over such list of triples and returns names of americans who are 20 years or younger. For example, `americans ppl` would return `["greg","joan","fred"]`.

(9) A vector is represented in a 3D $\mathbb{R}^3$ space with a triple of numbers, e.g., $v_1 = \langle 1, 2, 3\rangle$ and $v_2 = \langle 10, 20, 30\rangle$. Define function

  (a) `addVector` that receives two such 3D vectors as input and computes the their addition. For example, `addVectors v1 v2` returns `(11,22,33)`.

  (b) `dotProduct` that receives two such 3D vectors as input and computes the their dot product. For example, `dotProduct v1 v2` returns `140`.

  (c) `scalarMult` that receives a scalar value and a 3D vector, and computes their scalar multiplication. For example, `scalarMult 5 v1` returns `(5,10,15)`.

(10) Define function `isPrime` that receives a number and returns true if the input is a prime number. Otherwise, it returns false. For example, `isPrime 53` returns `True`, whereas `isPrime 57` returns `False`.

Note: You cannot use any prime-related function from Haskell library.

Hint: You can `divisors` function from above, or use list comprehension to define the function.

(11) Define function `primeRange` that receives a positive number and returns the list of all prime numbers less-than-or-equal-to that number. For example, `primeRange 20` returns `[2,3,5,7,11,13,17,19]`.

Note: You cannot use any prime-related function from Haskell library.

Hint: Use `isPrime` and list comprehensions to define it.