Working with lists

In this section we learn some very basic list operations.

Lists in Haskell

Lists of elements are one of the most primitive and most useful value types in Haskell. A list is simply a *sequence of zero or more elements of the same type*. These are defining characteristics of a list:

- All elements in a list must be of the same type (can't mix numbers and strings).
- A list can contain any number of elements.
- The elements in a list are accessed in a particular order.

We can create a list in two ways:

- By enclosing the desired elements in square brackets, and separated by commas: `[4, 6, 1, 2]`. A special notation `['a'..'g']`, called *enumeration*, allows us to form a list of all values from a specific value to another.
- By appending an element to the front of an existing list: `x:xs`. Here `x` is the new element, and `xs` is the list of existing elements. For instance the list `[1, 2]` can be written as `1:[2]` or also as `1:2:[]`. In this way lists are a little like linked lists in other languages.

There are many built-in functions that work on lists. They are all part of what is known as "Standard Prelude", and most can be seen in appendix B.8 from the book. You can also access the online documentation[1], though that takes a bit getting used to.

`head` Returns the first element of a non-empty list.

`tail` Returns all but the first element of a non-empty list.

`take` Given an integer `n` and a list, returns the first `n` elements from the list.

`drop` Given an integer `n` and a list, returns the remaining of the list after the first `n` elements are removed.

`length` Returns the length of a list.

`sum` Returns the sum of the elements in the list, assuming they can be added.

`product` Returns the product of the elements in the list, assuming they can be multiplied.

`++` Appends to lists.

`reverse` Reverses a list.

[1] https://hackage.haskell.org/package/base-4.10.0.0/docs/Prelude.html#g:13

List Practice

Here are some example uses (recall that function application does not require parentheses):

```
head [4..6]        -- Returns 4
tail [1..3]        -- Returns [2, 3]
take 3 [1..10]     -- Returns [1, 2, 3]
length "abc"       -- Returns 3. Strings are lists of characters.
length [1..10]     -- Returns 10
product [1..5]     -- Returns 120
reverse [1..5]     -- Returns [5, 4, 3, 2, 1]
```

Some practice questions:

1. How many characters are there in the string "The big bad wolf"? Have Haskell count them!
2. What is the product of all the numbers from 5 to 10?
3. How can we test if a string is a palindrome? (You can use == to compare two strings).

Function-writing practice

Write functions that accomplish the following:

1. prefix takes a list and returns the first three elements of the list.
2. isPalindrome takes a string and returns whether the string is a palindrome.
3. addInterest takes two arguments: A "principle" amount, and an interest "rate". It returns the new amount if we added the appropriate interest.
4. hasEnough takes a number and a list and returns whether the list has at least that many elements.
5. isDoubled takes a list and returns whether the list is the result of appending a list to itself (in other words, if the first half of the lst is exactly equal to the second half).
6. suffix takes a list and returns the last three elements of the list.