ASGS1115 - Support for Science

Semester 1 - 2019

LAB week 2: List and Functions

Lists

Load SumAvg.hs script from last week which contains the following functions:

```
1 -- Sum of 5 values:
2 s x1 x2 x3 x4 x5 = x1+x2+x3+x4+x5
3
4 -- Average of 5 values:
5 average x1 x2 x3 x4 x5 = s x1 x2 x3 x4 x5 / 5
```

We can store values to be inputted to the arguments of our function by executing these variable assignment expressions in GHCI:

```
> x1 = 2
> x2 = 3
> x3 = 4
> x4 = 5
> x5 = 6
```

But, imagine if there are more than 10 arguments:

```
s x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 ... = x1 + x2 + x3 + x4 + x5 +
 ↪  x6 + x7 + x8 + x9 + x10 + ...
```

making more than 10 variables to store each of the values would be tedious.

However, Lists in Haskell can store more than one values and solve this problem!

```
> nums = [2,3,4,5,6,7]
> words = ["Haskell", "is", "cool!"]
```

Q1

Try running these expressions in GHCI:

```
> nums2 = [1,[1],[1,1],[[1],[1,1]]]
> ones = [1,1,1,1,1,1,1]
> otherOnes = [[1],[1],[1],[1],[1],[1],[1]]
```

Are these lists equal?

Think about how they are (not) equal by running these expressions:

```
> nums == nums2
> nums2 == ones
> ones == otherOnes
```

Q2

Try List Concatenation by running these expressions:

```
> moreOnes = [1,1,1] ++ [1,1]
> mixOnes = [1,1,1] ++ ["one", "one"]
```

Is there anything unexpected happened? Why?

Q3

Try List Append by running these expressions:

```
> moreOnes2 = 1:moreOnes
> moreOnes = 1:moreOnes
```

Is there anything unexpected happened? Why?

Q4

Try Haskell built-in functions on lists:

```
> drop 3 nums
> sum nums
> product nums
> reverse nums
> head nums
> take 2 nums
> null [1,1,1]
> null []
> null [[]]
> head [[]]
> head []
```

Is there anything unexpected happened? Why?

String and lists are closely related. Try running the following expressions:

```
> :type "Haskell"
> :type ['H', 'a', 's', 'k', 'e', 'l', 'l']
> "Haskell" == ['H', 'a', 's', 'k', 'e', 'l', 'l']
```

Think about why the the last equation returned the value.

Functions

We have manipulated numbers ans strings with functions.

Try these functions:

```
> double a = a*a
> ddouble a = double (double a)
```

Note that we can re-use functions that we defined previously to define a new function.

Q5

Try running the script ANN.hs which contains a simple Artificial Neural Network algorithm with two inputs and one output

```
> :l ANN.hs
```

Note that it contains the following functions:

```
-- activation function
activation :: Double -> Double
activation z = 1 / (1 + (2.718**z))

-- bias
w0 = 0.1
w1 = 0.4

-- input weighting
z :: Double -> Double -> Double
z in0 in1 = w0*in0 + w1*in1

-- wrapper function
neuralNetwork :: Double -> Double -> Double
neuralNetwork x0 x1 = (activation (z x0 x1))
```

Function `neuralNetwork x0 x1` is the main function with x1 and x2 as inputs to the algorithm. Whereas `activation z`, and `z x0 x1` are supporting functions to get the appropriate output without flooding one function with mathematical expressions (note that all three functions are used in the definition).

Q6

Suppose we want our modify Artificial Neural Network to accept three-valued input. Modify the functions accordingly with 0.1, 0.4, and 0.7 as the first, second, and third weights accordingly.

The formula for the output of three input values is:

$$\text{output} = \text{activation}(\ \text{weight0} \times \text{input0} + \text{weight1} \times \text{input1} + \text{weight2} \times \text{input2}\ )$$

Try to modify the program before looking at the solution!

Simple reference about ANN: https://towardsdatascience.com/a-beginners-guide-to-neural-networks-b6be0d442fa4.

Bonus

Functions can take lists and do operations on the values stored in it. some of them are `sum` and `product`

Note the functions we defined in SumAvg.hs:

```
1 -- Sum of 5 values:
2 s x1 x2 x3 x4 x5 = x1+x2+x3+x4+x5
3
4 -- Average of 5 values:
5 average x1 x2 x3 x4 x5 = s x1 x2 x3 x4 x5 / 5
```

How do we put a list as an argument in our function so that we only have to input one list object to these `s` and `average` functions?