1. Lists (Cont.)

```
Prelude> [‘a’,’b’,’c’]
"abc"
Prelude> "abc"++"def"
"abcdef"
Prelude> [1,2,3] ++ [4,5,6]
[1,2,3,4,5,6]
Prelude> [1,2,3] ++ [1.5,2.5,3.5]
[1.0,2.0,3.0,1.5,2.5,3.5]
Prelude> [1,2,3]++”abc”
   * No instance for (Num Char) arising from the literal `1'
   * In the expression: 1
     In the first argument of `(++)', namely `[1, 2, 3]'
     In the expression: [1, 2, 3] ++ "abc"
Prelude> 1:[2,3,4]
[1,2,3,4]
Prelude> ‘a’:”bcd”
“abcd”
Prelude> 1:2:[4,5,6]
[1,2,4,5,6]
Prelude> 1:2:3:[]
[1,2,3]
Prelude> [1,2,3,4,5]!!3                      Indexing
4
```

2. Functions with different patterns

```
----------------firstHaskell.hs--------------------
myFunction 1 = "I am 1 year old"
myFunction 2 = "I am 2 years old"
myFunction 3 = "I am 3 years old"
myFunction _ = "I am older than 3 years old"

listFunction [] = “I am an empty list”
listFunction [x] = "I am a list with only one element:"++[x]
listFunction (x:xs) = x: “is my first element and my other elements are: “++xs

showAll x@(y:ys) = x ++ " is the whole, " ++ [y] ++ " is the first," ++ ys ++ " is the rest!"
-------------------------------------------------------
```

```
*Main> myFunction 1
"I am 1 year old"
*Main> myFunction 3
"I am 3 years old"
*Main> myFunction 5
"I am older than 3 years old"
*Main> listFunction []
“I am an empty list”
*Main> listFunction “a”
"I am a list with only one element: a"
*Main> listFunction “abc”
"a is my first element and my other elements are: bc”
*Main> showAll “Example”
"Example is the whole, E is the first, xample is the rest!"
```

3. **Recursive Functions**

```
----------------firstHaskell.hs--------------------
factorial 0 = 1
factorial n = n * factorial (n-1)
---------------------------------------------------
*Main> factorial 4
24
```

```
4 * factorial 3                                    4 * 6 = 24
        3 * factorial 2                   3 * 2 = 6
              2 * factorial 1     2 * 1 = 2
                          1
```

Re-implementation length function using a recursive function

```
----------------firstHaskell.hs--------------------
myLen [] = 0
myLen (x:xs) = 1 + myLen xs
---------------------------------------------------
Main> myLen [1,2,3]
3
```

```
1 + myLen [2, 3]                          1 + 2 = 3
   1 + myLen [3]                  1 + 1 = 2
       1 + myLen []       1 + 0 = 1
               0
```

4. Ranges

```
Prelude> [1..5]
[1,2,3,4,5]
Prelude> [1,3..11]
[1,3,5,7,9,11]
Prelude> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
Prelude> ['A'..'Z']
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
Prelude> take 5 [1..]                    Lazy Evaluation
[1,2,3,4,5]
Prelude> take 3 (repeat 5)
[5,5,5]
Prelude> replicate 3 5
[5,5,5]
Prelude> take 5 (cycle [1,2,3])
[1,2,3,1,2]
Prelude> drop 2 (cycle [2,3,4])
Infinite loop of numbers
```

5. List Comprehensions

```
Prelude> let xs = [1,2,3,4,5,6,7,8,9,10]
Prelude> [x | x<-xs, even x]
[2,4,6,8,10]
Prelude> [if x<5 then "Hello" else "Hi" | x<-xs, even x]
["Hello","Hello","Hi","Hi","Hi"]
Prelude> let removeUpperCase cs = [c | c<-cs, not(c `elem` ['A'..'Z'])]
Prelude> removeUpperCase "Computer Engineering"
"omputer ngineering"
Prelude> [ x | x <- “METU NCC”, x/= ‘N’, x==’C’]
“CC”
Prelude> [ x | x <- “METU NCC”, x== ‘N’, x==’C’]
“”
```

6. Data Types

a. Basic Data Type

```
Prelude> :type 'a'
'a' :: Char
Prelude> :type "CNG"
"CNG" :: [Char]
Prelude> :type True
True :: Bool
Prelude> :type 242
242 :: Num a => a
Prelude> :type [1,2,3]
[1,2,3] :: Num t => [t]
```

b. Function Type

```
*Main> :t factorial
factorial :: (Eq a, Num a) => a -> a
*Main> :t listFunction
listFunction :: [Char] -> [Char]
```

c. Dealing with numbers: An example

fromIntegral function takes an integral number and turns it into a more general number.

```
Prelude> length [1,2,3,4,5] + 3.2

<interactive>:21:22:
    No instance for (Fractional Int) arising from the literal `3.2'
    Possible fix: add an instance declaration for (Fractional Int)
    In the second argument of `(+)', namely `3.2'
    In the expression: length [1, 2, 3, 4, ....] + 3.2
    In an equation for `it': it = length [1, 2, 3, ....] + 3.2

Prelude> :t length [1,2,3,4,5]
length [1,2,3,4,5] :: Int
Prelude> fromIntegral(length [1,2,3,4,5]) + 3.2
8.2
Prelude> :t fromIntegral(length [1,2,3,4,5])
fromIntegral(length [1,2,3,4,5]) :: Num b => b
```

Practical Exercises:

a. Using **recursion**, implement a function that shows first x natural numbers.

sample run:

```
*Main> natural 10
[1,2,3,4,5,6,7,8,9,10]
*Main> natural 20
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
```

b. Implement the power function recursively without using ** or ^ operator, where power x y evaluates x^y.

sample run:

```
*Main> power 3 4
81
*Main> power (-3) 3
-27
*Main> power 0 4
0
*Main> power 4 0
1
```

c. Implement f(x) by using pattern matching (Explained in 2). (Without using if..else statements)

$$f(x) = \begin{cases} 1 - x, & x = 5 \\ \frac{x}{x^2}, & x = 10 \\ x, & Otherwise \end{cases}$$

sample run:

```
*Main> fx 5
-4.0
*Main> fx 10
0.1
*Main> fx 13
13.0
```

d. Write a function that takes a string and returns the number of vowels in the string.

sample run:

```
*Main> vowelcount “abc”
1
*Main> vowelcount “metu ncc”
2
*Main> vowelcount “”
0
```

e. Write a function that takes two lists where one list contains letter grades, and another list contains the credits. You function should calculate and then return the GPA (Grade Point Average). The weight of the letter grades are as follow: A = 4, B = 3, C = 2, D= 1 and F = 0.

sample run:

```
*Main> calculateGPA "ABAC" [4, 2, 3, 3]
3.3333333333333335
*Main> calculateGPA "BAD" [2,4,2]
3.0
```

References

Miran Lipovača, Learn You a Haskell for Great Good! A beginner's guide to Haskell, No Starch Press, Daly City, California, United States, 2011

Learn You a Haskell <http://learnyouahaskell.com/chapters>