Exercises, Week 42 (13–17 Oct, 2025)

DM580: Functional Programming, SDU

Learning Objectives

After doing these exercises, you will be able to:

- Write pattern matching functions in Haskell.
- Write Haskell programs that use list comprehensions.
- Write Haskell programs that use recursion.

1 Left-Biased Choice

Implement a function that implements a *left-biased choice* between two `Maybe` values.

```
(<+) :: Maybe a -> Maybe a -> Maybe a
```

If the first value is `Nothing`, the second value is returned. Otherwise, the first value is returned.

2 Calculating Sums of Squares (5.7.1 from the Book)

Using a list comprehension, give an expression that calculates the sum $1^2 + 2^2 + \ldots + 100^2$ of the first one hundred integer squares.

3 Coordinate Grid (5.7.2 from the Book)

Suppose that a *coordinate grid* of size $m \times n$ is given by the list of all pairs (x, y) of integers, such that $9 \leq x \leq m$ and $0 \leq y \leq n$. Using a list comprehension, define a function `grid :: Int → Int → [(Int, Int)]` that returns a coordinate grid of a given size.

For example:

```
λ> grid 1 2
[(0,0),(0,1),(0,2),(1,0),(1,1),(1,2)]
```

4 Coordinate Grid Without Diagonal (5.7.3 from the Book)

Using a list comprehension and the function `grid` above, define a function `square :: Int → [(Int,Int)]` that returns a coordinate square of size n, excluding the diagonal from $(0,0)$ to (n,n). For example:

```
λ> square 2
[(0,1),(0,2),(1,0),(1,2),(2,0),(2,1)]
```

5 Pythagoreans (5.7.5 from the Book)

A triple (x, y, z) of positive integers is *Pythagorean* if it satisfies the equation $x^2 + y^2 = z^2$. Using a list comprehension with three generators, define a function `pyths :: Int → [(Int,Int,Int)]` that returns the list of all such triples whose components are at most a given limit. For example:

```
λ> pyths 10
[(3,4,5),(4,3,5),(6,8,10),(8,6,10)]
```

6 List Comprehending Scalars (Based on 5.7.9 from the Book)

Using list comprehensions and the `zip` function from Haskell's `Prelude`, implement a function `scalarprod :: [Int] → [Int] → Int` that computes the scalar product of two lists of integers; i.e.:

$$\sum_{i=0}^{n-1}(xs_i \times ys_i)$$

Your function can assume that the two input lists are of equal length; i.e., no need to check for the cases where they are not.

7 Define Exponentiation Using Recursion (6.8.3 from the Book)

Define the exponentiation operator ^ for non-negative integers using the same pattern of recursion as the multiplication operator `*`, and show how the expression `2 ^ 3` is evaluated using your definition.

8 Euclid's Algorithm (6.8.4 from the Book)

Define a recursive function `euclid :: Int → Int → Int` that implements Euclid's algorithm for calculating the greatest common divisor of two non-negative integers: if the two numbers are equal, this number is the result; otherwise, the smaller number is subtracted from the larger, and the same process is then repeated. For example:

```
λ> euclid 6 27
3
```

9 All True

Implement a recursive function that checks if all numbers in a list are true.

```
allTrue :: [Bool] -> Bool
```

10 Any True

Implement a recursive function that checks if some number in a list is true.

```
anyTrue :: [Bool] -> Bool
```

11 Merge sort (6.8.7 from the Book)

Define a recursive function `merge :: Ord a ⇒ [a] → [a] → [a]` that merges two sorted lists to give a single sorted list. For example:

```
λ> merge [2,5,6] [1,3,4]
[1,2,3,4,5,6]
```

12 To Binary

Implement a recursive function that converts an integer to a binary number, represented as a list of 0s and 1s (integers).

```
toBinary :: Int -> [Int]
```

The binary numbers yielded by your function should have the most significant bit first. For example:

```
λ> toBinary 8
[1,0,0,0]
```

13 Balanced Word Recognizer

Implement a recursive function that recognizes *Dyck words*. That is, your function should return true if the input is a string of `'['` and `']'` characters that are *balanced*.

```
recognizeDyck :: String -> Bool
```

Examples of balanced strings (a.k.a. Dyck words) include:

- `[]`
- `[[]]`
- `[][]`
- `[[[][]][]]`

14 Palindrome Recognizer

Implement a function that recognizes *palindromes*.

```
recognizePal :: String -> Bool
```

Examples of palindromes include:

- `regninger`
- `ifihadahifi`
- `12321`
- `dameskibsbiksemad`
- `roma tenet amor`
- `moorddroom`

Optional: implement your recognizer using the “Getting There and Back Again” recursion pattern (Danvy, 2022).

15 Percentage of Numbers Containing the Digit 7

Implement a recursive function `percentageOf7s :: Int → Double` that takes an integer number as input, and returns the percentage of numbers betwen 0 and the input that contain the digit 7.

For example, the percentage of numbers between 0 and 10 that contain the digit 7 is 10%.

More examples:

```
λ> percentageOf7s 10
0.1
λ> percentageOf7s 100
0.19
λ> percentageOf7s 1000
0.271
```

References

O. Danvy. Getting there and back again. *CoRR*, abs/2203.00145, 2022. doi: 10.48550/ARXIV.2203.00145. URL `https://doi.org/10.48550/arXiv.2203.00145`.