Mini Course: Functional Programming

Session 1: Haskell syntax, functions

Jeremie Gillet

Outline

- Session 1:
  - Haskell syntax
  - Pure functions
  - Function composition
  - Currying
- Session 2:
  - Pattern Matching
  - Recursion
  - Folds
  - Maybe
- Session 3:
  - Immutability
  - Laziness
  - Parallelization
  - Exercises
- Session 4:
  - Algebraic Data Types
  - Mapping over ADTs
  - Reader, Writer, State...

Pattern Matching

Pattern matching

Using the structure of the input

- Pattern matching is the act of finding a specific pattern in an input
- In Haskell, it is fully integrated
- Exercise: ExercisesDay2, 15 minutes

Recursion

Recursion

No loop in FP

- No for or while control structures in FP
- Instead, use recursion
  - Functions that call themselves, breaking into smaller pieces until base case
- Exercise: ExercisesDay2, 15 minutes
- Tail recursion
  - Tail recursive if final call of the function returns the value itself
  - Can avoid using adding a frame to the stack => equivalent to a loop
  - Haskell optimizes tail calls automatically

Folds

Folds

Aggregating your data

- A fold is a higher order function that processes a data structure and returns a value
- In Haskell: foldl, foldr, foldl1, foldr1, scanl, scanr, scanl1, scanr1
- Note the difference in types: (b -> a -> b) for left and (a -> b -> b) for right, b is the “accumulator”
- Example: sum = foldr (+) 0
- foldl f a [b, c, d] = f (f (f a b) c) d)
  foldr f e [b, c, d] = f b (f c (f d e))
- Exercise: ExercisesDay2, 20 minutes

Maybe

Maybe

Basic FP Error handling

- Computations may fail, let them do so gracefully
- Two possible values for Maybe a: Nothing and Just a
- You can compose with “regular” functions using fmap
- You can compose with “failable” functions using <=<
- You can compose several “failable” functions with liftM
- Exercise: ExercisesDay2, until the end