Lab 1

* ++ to concatenate strings ex. “Hello” ++ “World” => “Hello World”
* Elm has type inference
* Variable shadowing is an error
* Function signatures
  + funcName : Pram1Type -> … -> ParamNType -> ReturnType
* If experssions
  + Else branch is mandatory!!!!!
* Tail recursion
  + Concept: A function is tail recursive if it returns either something computed directly or something returned by its recursive call (the last thing it does is to call itself)

Lab 2

* Tuples
  + Are limited by design to contain at most 3 items
  + Can contain different types of data
  + Help us keep related data close together, or pair up related values temporarily
* Records
  + Collection of named fields
  + ex. { firstName : String, lastName : String }
  + Call function  
      
    Text

    Description automatically generated
* Type aliases
  + Used to give a new name to existing types (in addition to the existing name)
  + Most common used to give name to records ex. type alias User = {firstName: String, lastName: String}
  + Create instances
    - v1

Text

Description automatically generated

* + - v2

Text

Description automatically generated

* Type definitions
  + Allow us to create **new types**
  + Used to create enumerated types
  + ex.
    - type Color = Red | Green | Blue
    - type Point = Point Int Int
      * first Point = type name
      * second Point = constructor name
* Sum types and Product types
  + For the sum types, the cardinality is equal to the number of variants of the given type
    - Color type has cardinality = 3
    - Int ^ a cardinality = 2 ^ 32
  + For product types, the cardinality is equal to the product of the cardinality of each field
    - Point type has two Int fields => cardinality = 2 ^ 32 \* 2 ^ 32

Graphical user interface

Description automatically generated with low confidence

* Let … in
  + With let ... in we can declare bindings and use them in a local scope
* Destructuring
  + A picture containing diagram

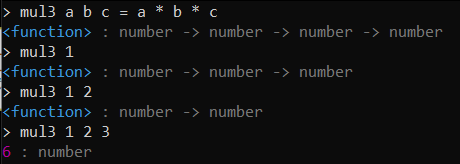
    Description automatically generated
  + Text

    Description automatically generated with low confidence
  + \_ : for the variables we don’t want to destructure
* Case
  + \_ ⬄ others
  + Patterns are checked from top to bottom until one matches and that branch is chosen
  + Multiple conditions case
    - case (windSpeed < 61, cloudLayer < 1400) of

Lab 3

* Type variables
  + Variable ranging over types
  + Names are in lowercase and may contain more than just one character
  + Can appear more than once to indicate that these values must have the same type, but this type cand be any type
* Equality
  + By default, Elm automatically implements **deep structural equality** for all values through the == operator
* Maybe type
  + For nullability
  + type Maybe a = Just a | Nothing
* Result type
  + For failure
  + type Result = Ok value | Err error
  + Signaling the possibility of failure
    - Using a string to return an error message
    - Defining an enum type that represents each possible error
* Lists part 1
  + Define lists
    - [1, 2, 3]
    - 1 :: 2 :: 3 :: []

Lab 4

* Controlling exported items
  + module NumeFisier exposing (functionName, TypeName(..) …)
* Open imports
  + import NumeFisier
  + NumeFisier.functionName
* Qualified imports
  + import NumeFisier as n
  + n.functionName
  + import NumeFisier as n exposing( etc )
* Higher order functions
  + A function which manipulates other functions: it either takes other functions as parameters or returns a function
* Partial application and Currying
  + Every function, when applied to fewer arguments than the number of parameters returns a function
  + Currying
    - A curried function can take its arguments, one at the time
    - Each time we provide one or more (but not all arguments), the function will return a new function which expects the remaining arguments  
        
      
  + Point free
    - the main goal is to hide the parameters (points) the function is applied to
* Lambdas and closures
  + Lambda expressions
    - Syntax: \arguments -> returnedValue
    - Ex. with multiple arguments  
        
      Text

      Description automatically generated
  + Closures
    - A function that captures its environment when it is created
    - Closures must be local definitions as the environment they can capture consists of the parameters and local definitions of the function they are defined in
* Combinator functions
  + Functions that only refer to their arguments
  + Const function
    - Takes one argument and returns a function which always returns this argument
  + Flip function
    - Takes a function as argument and returns a function which takes the arguments of the first function in reverse order
  + Uncurry function
    - Takes a curried function, which takes 2 arguments and returns a function which takes a 2-tuple
* Lists part 2
  + Take
    - Take the first *n* members of a list
  + Drop
    - Drop the first *n* members of a list
  + Take while
    - Take the members of a list while a predicate function
  + Drop while
    - Drop the members of a list while a predicate function
  + Zip and unzip  
      
    Graphical user interface, text, application

    Description automatically generated
  + Map  
      
    A picture containing chart

    Description automatically generated
  + Filter  
      
    Text

    Description automatically generated with low confidence
  + Foldl and Foldr  
      
    Text

    Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

* Strings
  + String.toList
  + All and Any  
      
    Text

    Description automatically generated

Text

Description automatically generated