Presentation: Week02: Intro to Functions & Intro to Types

Useful info

Information to note:

Interpreter - **directly runs code, translates just one statement at a time to machine code**
Compiler - turns entire source code into machine language (at once) that can be read by the computer
Compile time - the period when the programming code (such as C#, Java, C, Python) is converted to the machine code (i.e. binary code)
Run time - Period of time when program is actually running, and occurs after compile time

Haskell has a strong Static type system

- Makes Haskell unique
- Type of every expression is known at compile time.
- Haskell has type inference -> Haskell can infer the type on it’s own (which is what you see with the ide features)
- Static type systems allow bugs to be found at COMPILE time rather than RUN time.
- One important benefit of static typing is that many bugs are found at compile time, rather than run time.

** With a dynamically typed language, you might ship code that seems to work perfectly well. But then in production a user stumbles on an edge case where, for instance, you forget to cast something as a number. And your program crashes.

** With static typing, your code will simply not compile in the first place. This also means that errors are more likely to be found where they first occur.

In Haskell:

```
ghci> :t 'a'
'a' :: Char
```
:: is read as “has type of”

Lists:

- Allows grouping together values: i.e. [1,2,3] << square brackets
- A list can only contain the same type i.e. Integers, Characters, etc.
- [1 .. N] generates a list from 1 to n in Haskell

TUPLES

- Are specified using round brackets
- Tuples can have items of multiple types
- Would have to specify each item in the tuple. fst(x,y) = x
  - (False, True) :: (Bool, Bool)

Prelude

- A library containing predefined functions
- i.e. last, head, etc.

Functions:

- \`div\` is integer division (will only give you result as whole number i.e. 5 \`div\` 2 is 2)
    - if you want infix operator put the \` \` symbols i.e. 5 \`div\` 2
    - if you would like to treat it as a normal function you can i.e.: div (sum ns) (length ns)
- ++ to concatenate two lists i.e. [1,2,3] :: [1,2,3]
- :   to add an element to the beginning of a list i.e. 1:[1,2,3]

If the name consists of letters, -> requires \`backticks\` to be used as infix operator

```
of :: Integer -> Integer -> Integer
a `of` b = c
```

If the name consists of symbols -> requires parentheses to be used as infix operator
i.e.

```
(@@) ::Integer -> Integer -> Integer
 a @@ b = c
```

Syntax

- **FUNCTION must start with a lower case letter, usually myFunc, starts lower ends upper**
- **convention : list suffix has an s in their name i.e. ns, xs**
- **definitions must be in the same column!**
- **i.e. : a = b**
- **b = c etc.**

LET AND WHERE EXPRESSIONS

- Let and where are both equivalent and do the same thing.

TYPES

- A type is a name for a collected of related values, i.e. Bool, Integer,

i.e. Integer: -1 , 0, 1, 2, 3 -> whole numbers, including negatives.
Natural Numbers: 1,2,3,4 -> whole numbers, >0 greater than but not including zero
Real : any real number on the number line. i.e. 3.14159
String = [Char] : "Hello"

- Haskell is STRICTLY TYPED -> every variable has a type that cannot change
- Will get errors if types are incorrectly declared or inputted!

Type declarations

- **Num a => a -> a -> a; Num a is the type class, the first 'a' is a is the first parameter the function takes, the second 'a' is the second parameter, and the last 'a' is the return type of the proposed function**

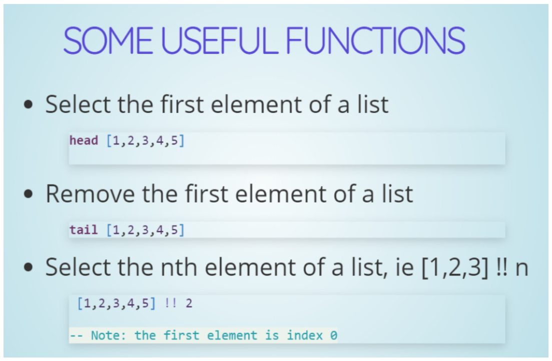
INT VS INTEGER IN HASKELL :

```
Int: Prelude> (minBound, maxBound) :: (Int, Int)
(-9223372036854775808,9223372036854775807)

Integer: range as large as you have memory for.
```

TIPS:

- Try and add individual project folders to workspace instead of opening entire folder containing multiple projects.
- If want to know a type for a certain element, type ":t" followed by any valid expression , i.e. " :t 'a' will give you char.
- If you want more information on a function, or rather anything on Haskell type, 'info'
- Go on google: type the function name, and websites like ZVON will pop up, and explain the function. i.e. foldr
- To open up current directory on terminal: MAC: open. Windows: explorer .



SOME USEFUL FUNCTIONS

- Select the first element of a list

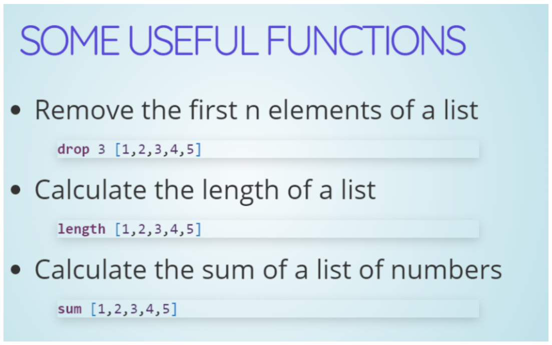
```
head [1,2,3,4,5]
```

- Remove the first element of a list

```
tail [1,2,3,4,5]
```

- Select the nth element of a list, ie [1,2,3] !! n

```
[1,2,3,4,5] !! 2
-- Note: the first element is index 0
```



SOME USEFUL FUNCTIONS

- Remove the first n elements of a list

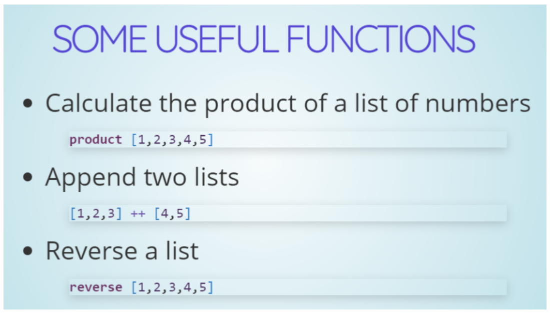
```
drop 3 [1,2,3,4,5]
```

- Calculate the length of a list

```
length [1,2,3,4,5]
```

- Calculate the sum of a list of numbers

```
sum [1,2,3,4,5]
```



SOME USEFUL FUNCTIONS

- Calculate the product of a list of numbers

```
product [1,2,3,4,5]
```

- Append two lists

```
[1,2,3] ++ [4,5]
```

- Reverse a list

```
reverse [1,2,3,4,5]
```