Functional Programming
CS3016

INSTRUCTOR: Andrew Butterfield
Tim.Fernando@tcd.ie

3) Types and Typeclasses

Types in Haskell are determined at compile time and any issues found will prevent the program from running until these issues have been resolved. Haskell also has **type inference** where Haskell will attempt to pre-determine the type of an object for you.

The expressions:

- True is a boolean
- "hello" is a string

```
ghci> :t 'a'
'a' :: Char
ghci> :t True
True :: Bool
ghci> :t "HELLO!"
"HELLO!" :: [Char]
ghci> :t (True, 'a')
(True, 'a') :: (Bool, Char)
ghci> :t 4 == 5
4 == 5 :: Bool
```

Functions in Haskell also have their own types. When writing functions it is generally a good idea to give them a type declaration:

```
removeNonUppercase :: [Char] -> [Char]
removeNonUppercase st = [ c | c <- st, c `elem` ['A'..'Z']]
```

When a function takes multiple parameters its type declaration can be formatted as follows:

```
addThree :: Int -> Int -> Int -> Int
addThree x y z = x + y + z
```

This function takes three integers (x, y and z) and returns another ingerer.

Common Haskell types include the following:

- Int - Whole numbers
- Integer - Bigger whole numbers
- Float - Floating point with single precision
- Double - Floating point with double precision
- Bool - Boolean (true/false)
- Char - Single character

Types - Type Variables

Types of functions that can take many different parameters of different types produce abstract type variables. Consider the head of a list function below:

```
ghci> :t head
head :: [a] -> a
```

This function takes a List of type a and returns a single element of type a.

Consider *fst* which takes a tuple and returns the first element of a tuple:

```
ghci> :t fst
fst :: (a, b) -> a
```

This function takes a Tuple of types a,b and returns a single element of type a.

Types - Type Classes

Typeclasses are a sort of interface that defines some behaviour. If a type is a part of a typeclass it means that it supports and implements the *behaviour* that the typeclass describes.

```
ghci> :t (==)
(==) :: (Eq a) => a -> a -> Bool
```

This can be read as the equality function of typeclass (Eq) takes any two elements of the same type a and returns a Bool.

The Eq typeclass is an interface for testing equality. All standard Haskell types except for IO and functions are part of the Eq typeclass.

The elem function has a type of `(Eq a) => a -> [a] -> Bool` as it uses `==` over a list to check if some value we are looking for is present.

Some basic typeclasses include:

1. Eq - Used for types that support equality testing

```
ghci> 5 == 5
True
ghci> 5 /= 5
False
ghci> 'a' == 'a'
True
ghci> "Ho Ho" == "Ho Ho"
True
ghci> 3.432 == 3.432
True
```

2. Ord - Used for types that have an ordering

```
ghci> "Abrakadabra" < "Zebra"
True
ghci> "Abrakadabra" `compare` "Zebra"
LT
ghci> 5 >= 2
True
ghci> 5 `compare` 3
GT
```

3. Show - Convert variable to string

```
ghci> show 3
"3"
ghci> show 5.334
"5.334"
ghci> show True
"True"
```

4. Read - Convert string to variable

```
ghci> read "True" || False
True
ghci> read "8.2" + 3.8
12.0
ghci> read "5" - 2
3
ghci> read "[1,2,3,4]" ++ [3]
[1,2,3,4,3]
```

This must be used with a read follow by an action so that Haskell can infer the type of the result based on the result of the action.

```
ghci> :t read
read :: (Read a) => String -> a
```

5. Enum - Sequentially ordered types - they can be enumerated. Can be used for various objects that support successors (succ) and predecessors (pred). Types in this class include `()`, `Bool`, `Char`, `Ordering`, `Int`, `Integer`, `Float` and `Double`.

```
ghci> ['a'..'e']
"abcde"
ghci> [LT .. GT]
[LT,EQ,GT]
ghci> [3 .. 5]
[3,4,5]
ghci> succ 'B'
'C'
```

6. Bounded - Members have an upper and lower bound

```
ghci> minBound :: Int
-2147483648
ghci> maxBound :: Char
'\1114111'
ghci> maxBound :: Bool
True
ghci> minBound :: Bool
False
```

7. Num - Numeric typeclass. Its members have the property of being able to act like numbers.

```
ghci> :t 20
20 :: (Num t) => t
```

```
ghci> 20 :: Int
20
ghci> 20 :: Integer
20
ghci> 20 :: Float
20.0
ghci> 20 :: Double
20.0
```

```
ghci> :t (*)
(*) :: (Num a) => a -> a -> a
```

8. Integral - Also a numeric typeclass. Num includes all numbers, including real and integral numbers. Whereas Integral only included Int and Integer (whole numbers).

9. Floating - Includes only floating point numbers (Float and Double).