COMP105 Lecture 21

Maybe and Either

The Maybe type

```
data Maybe a = Just a | Nothing
```

```
ghci> :t Just "hello"
Maybe [Char]
```

```
ghci> :t Just False
Maybe Bool
```

```
ghci> :t Nothing
Maybe a
```

The `Maybe` type

The `Maybe` type is used in pure functional code **that might fail**

```
safe_head [] = Nothing
safe_head (x:_) = Just x
```

```
ghci> safe_head [1,2,3]
Just 1
```

```
ghci> safe_head []
Nothing
```

Maybe example

```
safe_get_heads list =
    let
        mapped = map safe_head list
        filtered = filter (/=Nothing) mapped
        unjust = (\ x -> case x of Just a -> a)
    in
        map unjust filtered
```

```
ghci> safe_get_heads [[], [1], [2,3]]
[1,2]
```

Exceptions in Haskell

Haskell does include support for **exceptions**

```
ghci> head []
*** Exception: Prelude.head: empty list
```

Exceptions are **not** pure functional

- Every function returns exactly one value
- You can't catch exceptions in pure functional code
- Exceptions are mostly used in IO code

Exceptions in Haskell

The `Maybe` type provides a way to do **exception-like** behaviour in pure functional code

Can this function fail for some inputs?

- use the `Maybe` type

Exceptions should only be used in **IO** code

- File not found, could not connect to server, etc.
- These are unpredictable events

The Either type

```
data Either' a b = Left a | Right b
```

```
ghci> :t Left 'a'
Either Char b
```

```
ghci> :t Right 'b'
Either a Char
```

The Either type

The either type is useful if you want to store **different types** in the same list

```
ghci> let list = [Left "one", Right 2,
                             Left "three", Right 4]
```

```
is_left (Left _) = True
is_left _        = False
```

```
ghci> map is_left list
[True,False,True,False]
```

The Either type

```
get_lefts list =
    let
        filtered = filter is_left list
        unleft = (\ (Left x) -> x)
    in
        map unleft filtered
```

```
ghci> get_lefts list
["one","three"]
```

Example: squaring mixed number types

```
ghci> let nums = [Left pi, Right (4::Int), Left 2.7182]

square (Left x)  = Left (x ** 2)
square (Right x) = Right (x ^ 2)

ghci> map square nums
[Left 9.86,Right 16,Left 7.38]
```

Meaningful error messages

Either can be used to give **detailed errors**

```
safe_head_either []    = Right "empty list"
safe_head_either (x:_) = Left x
```

```
ghci> safe_head_either []
Right "empty list"

ghci> safe_head_either [1,2,3]
Left 1
```

Exercises

1. Write a function `safeTail :: [a] -> Maybe [a]` that is the safe version of `tail`
2. Write a function `safeDiv :: Int -> Int -> Maybe Int` that is the safe version of `div`
3. Write a function `safeGet :: [a] -> Int -> Either a String` that is the safe version of `!!`, ie., `safeGet list i` should return `list !! i` if the index is small enough. If the index is out of range, then a string error message should be produced