Cheatsheet – Haskell Workshop 1

Stack

- Run the REPL (Read-Eval-Print Loop):

```
$ stack repl
```

- Build the project:

```
$ stack build
```

Syntax

- Import the `mapMaybe` function from the `Data.Maybe` module:

```
import Data.Maybe (mapMaybe)
```

- Declare a new module called `Main`, implementation goes after the `where` keyword:

```
module Main where
...
```

Types

- `::` is used to denote the type of a term, for example `3` is a value of type `Int` and also a value of type `Double`:

```
3 :: Int
3 :: Double
```

- `id` is a function (its type is `->`) taking any value of type `a` and returning a value of type `a`. How many functions of this type are there?

```
id :: a -> a
```

- `Bool` is a sum type:

```
data Bool = False | True
```

- `Maybe` is a sum type, with a type parameter `a`, as is `[]` (List):

```
data Maybe a = Nothing | Just a
data [] a = [] | [] : [a]
```

- `String` is exactly a list of `Char`:

```
type String = [Char]
```

Useful Functions

-
```
putStrLn                   :: String       -> IO ()
Data.ByteString.Lazy.putStr :: B.ByteString -> IO ()
```

- Encode a value to JSON:

```
Data.Aeson.encode :: ToJSON a => a -> B.ByteString
```

- Keep only the first element of a list:

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

- Remove the first element of a list:

```
tail :: [a] -> [a]
```

- Reverse the order of a list:

```
reverse :: [a] -> [a]
```

- Split a string on `\n`, split a `String` on `Char`:

```
lines                  ::         String -> [String]
Data.List.Split.splitOn :: Char -> String -> [String]
```

- Read a `String` into a Haskell value:

```
Data.Read.readMaybe :: Read a => String -> Maybe a
```

- Apply a function to all elements of a list, returning a new list.

```
map                 :: (a ->       b) -> [a] -> [b]
Data.Maybe.mapMaybe :: (a -> Maybe b) -> [a] -> [b]
```

- Read/Write a full file into/from a `String`:

```
readFile :: FilePath -> IO String
writeFile :: FilePath -> String -> IO ()
```

- Reverse the order of a list:

```
show :: Show a => a -> String
```

REPL tricks

- Show the type of an expression:

```
:t Just True
```

- Reload the currently loaded module:

```
:r
```