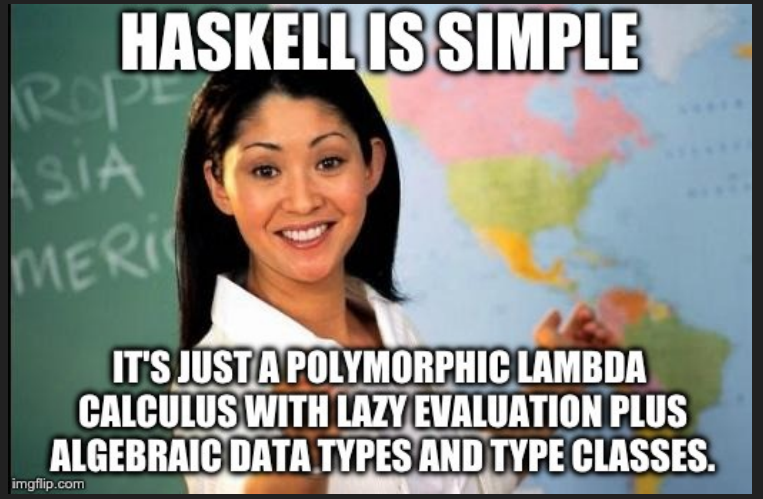
FIT2102 Programming Paradigms Tutorial 6

Discovering Haskell



Faculty of Information Technology

Getting Started!

1. Installing Stack
    - Stack is a cross-platform program for developing Haskell projects. It is aimed at Haskellers both new and experienced.
2. Download the workshop code from moodle, unzip and open the folder then:
3. Run the command `stack ghci`
    - This is the Glasgow Haskell Compiler interactive interpreter, you will be able to load and run your code directly in there. This interactive console has tab completion, information on the code loaded, etc.
4. Running the tests `stack exec doctest src/List.hs` for one file or `stack test` for all files
    - `Examples: 11  Tried: 11  Errors: 0  Failures: 0`

Let’s start with Quick Sort

A simple pseudo-code version of the quick sort algorithm:

```
QuickSort list:
  Take head of list as a pivot
  Take tail of list as rest
  return

        Quicksort(lesser) ++ [pivot] ++ Quicksort(greater)
                              ⇧          ⇧
                           concat     concat

        where
              lesser = filter(x <= pivot)
              greater = filter(x > pivot)
```

How about Haskell?

Wow. Beautiful

```
sort [] = []
sort (pivot:rest) = lesser ++ [pivot] ++ greater
 where
   lesser = sort (filter (<pivot) rest)
   greater = sort (filter (>=pivot) rest)
```

What is Haskell?

Haskell is a purely functional language, which means that functions generally have no side effects.

Haskell has many nice features such as:

- Everything is curried!
- Lazy evaluation
- Lambda expressions
- Pattern matching
- List comprehension
- Type classes
- Etc.

-- (Wikipedia, 2019)

Pattern Matching!

```
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

⇦ Base Case

⇧ Pattern Matching

```
length' [] = 0
length' (_:xs) = 1 + length' xs
```

⇦ Base Case

⇧ Pattern Matching

Type Declaration ⇩

```
capital :: String -> String
capital "" = "Empty string, whoops!"
capital all@(x:xs) = "The first letter of " ++ all ++ " is " ++ [x]
```

⇧ Reference Whole Input & Pattern Matching

⇧ Concat

Some Syntax

- *where* - keyword which lets us create locally scoped variables

```
f x = y
   where y = x * 2
```

- *If* - statements

```
min a b = if a > b then b else a
```

- Type Declaration

```
function :: InputType1 -> InputType2 -> InputType3 -> OutputType
```

- Common Types

```
Bool,Int,[Int],(String or [Char])
```

Revisiting Quicksort

Now let's analyse this!

```
sort [] = []
```

⇦ Base Case

Pattern Matching ⇩

```
sort (pivot:rest) = lesser ++ [pivot] ++ greater
 where
   lesser = sort (filter (<pivot) rest)
   greater = sort (filter (>=pivot) rest)
```

⇧ Currying an operator (<, >=)

Revisiting Quicksort

Now let's analyse this!

```
sort [] = []
```

⇦ Base Case

Pattern Matching ⇩

```
sort (pivot:rest) = lesser ++ [pivot] ++ greater
 where
   lesser = sort $ filter (<pivot) rest
   greater = sort $ filter (>=pivot) rest
```

⇧ Currying an operator (<, >=)

Housekeeping

1. Assignment is due next week!
    - Hopefully you have started by now, if not, start ASAP!