Intro

Author: Matt Broadway
Date: 27/10/15
Disclaimer: This is not guaranteed to be correct (but I think it is). You decide whether to believe me :)

Currying

See this explanation of currying

One of many effects this has

when writing a type signature in OCaml, `->` is *right associative*. This means that these two type signatures are the same:

```
val f1 : 'a -> 'b -> 'c
val f2 : 'a -> ('b -> 'c)
```

This has the effect of being able to write the same function multiple ways:

`f1` and `f2` are equivalent. The difference between them is how they are conceptually treating the type signature.

```
val f1 : 'a -> 'b -> 'c

let f1 x y = ("do something to get a 'c");;
```

`f1` treats the type signature the 'normal' way, that is: "`f1` is a function that takes 2 arguments of type `'a` and `'b` and gives a type `'c`"

```
val f2 : 'a -> ('b -> 'c)

let f2 x =
  let returnedFunction y = ("do something to get a 'c")
  in
    returnedFunction;;
```

`f2` treats the type signature as meaning: "`f2` is a function that takes a single argument of type `'a` and gives a function. That function has type `'b -> 'c`"

Because of currying, these two interpretations are both valid, can be called in the same way and produce the same result.