1 Haskell: An Introduction

Haskell is essentially Lambda Calculus plus

- Better syntax.
- Types.
- Built-in features like primitives (booleans, numbers, etc.), records, lists, recurison, and more.

1.1 Types

In Haskell, every expression either **has a type** or is **ill-typed** and rejected statically (at compile-time, before execution begins). This is similar to Java (which is statically typed), and differs from languages like Python (which is dynamically typed).

1.1.1 Type Annotations

While the Haskell compiler can infer types, you should still annotate your bindings with their types using `::`, like so:

```
aBoolean :: Bool
aBoolean = True

message :: String
message = if aBoolean
            then "True!"
            else "False."

rating :: Int
rating = if aBoolean
            then 10
            else 0
```

Note that we can use the GHCi command `:t` to inspect types. For example,

```
> :t if x then 'a' else 'b'             -- Char
> :t \b -> if b then 'a' else 'n'       -- Bool -> Char
```

Note that we can also have functions with mulitple parameters; that is:

```
pair :: String -> (String -> (Bool -> String))
pair :: String -> String -> Bool -> String      -- Same as above.
pair x y z = if b then x else y                 -- Definition of function.
```

2 Lists

A list is either:

- An empty list

```
[]              -- pronounced "nil"
```

- Or a head element attached to a tail list.

```
x:xs            -- pronounced "x cons xs"
```

So, for example, we have:

```
[]                              -- A list with 0 elements.
1:[]                            -- A list with 1 element.
(:) 1 []                        -- Same thing
1:(2:(3:(4:([]))))              -- A list with 4 elements.
1:2:3:4:[]                      -- Same thing (: is right associative)
[1, 2, 3, 4]                    -- Same thing (syntactic sugar)
```