Function Syntax

Here we will disect the construction of a Haskell function, looking at all the different parts and what they mean

Structure

There are two main parts to a Haskell function:

```
-- Type signature:
nameOfFunction :: Type
-- Definition:
nameOfFunction = definition
```

The type signatiure is opertional, but reccomented. IT acts as a summary of what the function does including its name, what inputs it takes and what it will turn them into

–

You can use the LHS of the equals to name the various input parameters so that you can refer to them easily on the RHS of the definition where you will actually compute stuff with them. These are just names, you can chose any names you want.

```
add :: Num a => a -> a -> a
add x y = x + y
```

```
plus :: Num a => a -> a -> a
plus water melon = water + melon
```

The above two functions do exactly the same thing, the only difference is that `add` uses more succinct and mathematical labels for the two inputs whereas `plus` is more fun and random. We like super short variable names because it's far easier to type short things.

–

One final trick that you may see people use is the underscore. When naming inputs if there is an input that you don't care about and don't need to name since you are not doing to use it on the RHS you can just put an underscore as a place holder.