Activity 6

We can create our own data types by simply listing the possible values that the type may have. Consider this example:

```
data Thing = Shoe | Ship | SealingWax | Cabbage | King
        deriving Show
```

This declares a new type called Thing with five possible values (Shoe, Ship, etc) which are the only values of type Thing. The *deriving Show* is a magical incantation which tells Haskell to automatically generate default code for printing values of type Thing.

We can write functions on type Thing by *pattern matching*:

```
isSmall:: Thing -> Bool
isSmall Shoe = True
isSmall Ship = False
isSmall SealingWax = True
isSmall Cabbage = True
isSmall King = False
```

Try

```
> isSmall Cabbage
```

In a function, the cases are tried in order from top to bottom, so we could also make the definition of isSmall a bit shorter by using a default pattern _. You can read the _ as meaning "everything else".

```
isSmall2 :: Thing -> Bool
isSmall2 Ship = False
isSmall2 King = False
isSmall2 _ = True
```

Try

```
>isSmall2 Cabbage
```

Data type values need not be a simple list like we saw above. The types may also include arguments. Consider the following data type:

```
data FailableDouble = Failure | OK Double
        deriving Show
```

This says that the FailableDouble type has two values. The second case, OK, takes an argument of type Double. So OK by itself is not a value of type FailableDouble; we need to give it a Double. For example, OK 3.4 is a value of type FailableDouble

Here's one way we might use our new FailableDouble type:

```
safeDiv:: Double -> Double -> FailableDouble
safeDiv _ 0 = Failure
safeDiv x y = OK (x / y)
```

Try

```
> safeDiv 2 0
> safeDiv 3 4
```

Complete the function failureToZero which converts a FailableDouble to a regular double by changing the Failure values to zero but leaving the OK values to be the double value that has been deemed OK.

```
failureToZero :: FailableDouble -> Double
failureToZero _____ =  0
failureToZero (OK d) = _____
```

You can test your function with:

```
>failureToZero (safeDiv 2 0)
>failureToZero (safeDiv 3 4)
```