330 2013-11-14

Note Title 2013-11-14

Product example (calculates product of a list of numbers

Step 1, define the type

product :: [Int] -> Int

(we may generalize later; we "start simple")

Step 2 enumerate the cases

product [] = (empty list)

product (x:xs) = (non-empty list)

Step 3 define the simple cases

product [ ] = 1

product (x : xs) =

(Usually, the simple cases become base cases, as it happens here.)

Step 4 Define the other cases

```
product [] = 1

product (x:xs) = x * product xs
```

(As in this example, the "other" cases often become the recursive cases)

Step 5 Generalize and simplify

product :: Num a => [a] -> a

The function defn is unchanged when we generalize from Int to any type of the Num class.

```
folder (*) 1
```

1 ← the value of the function you define when applied to []

(*) ← the function that replaces cons (:)

foldr is actually very similar to
FP's insert (!)

!* is the FP equivalent to
foldr (*) 1

!* works only on lists with at least two values

foldr1 (*) is actually closer to !*.