How To Think Recursively

① What are recursive functions?
→ A function which is defined itself

② Why are they useful?
→ Most functions naturally are recursive in Haskell

③ Why are they diffcult at first?
→ Switching over from imperative, loop based programming

7 Step Process for Recursion functions

① Name the function.
② Write down it's type.
③ Enumerate the cases.
④ Define the simple cases.
⑤ List the "ingredients".
⑥ Define the other cases.
⑦ Think about the result.

↓ Helper for writing the functions

Problem: Define a function that calculates the sum of a list of numbers → Any numeric type

```
①  sum :: Num a ⇒ [a] → a        ② & ⑦
③  sum []     = 0                ④
③  sum (x:xs) = x + sum xs       ⑥
```

⑦ Can it be generalized the type and simplify the cases

⑦ `sum = foldr (+) 0`

Problem: Define a function that drops a given number of elements from the start of a list.

```
drop :: Int -> [a] -> [a]
drop 0 []     = []
drop 0 (x:xs) = x:xs
drop n []     = []
drop n (x:xs) = drop (n-1) xs
```

→ The original list is unchanged

Don't care about this

↳ can get rid of this

```
drop 0 xs     = xs
drop _ []     = []
drop n (_:xs) = drop (n-1) xs
```