Chapter 7 High Order Functions

→ A function is high order if it takes a function as an argument or returns a function as a result

Ex:
```
twice :: (a → a) → a → a
twice f x = f (f x)
```

→ **Common programming patterns** can be coded using high order function

→ **Domain specific languages (DSL)** can be defined as collections of high order functions

→ **Algebraic properties** of high order functions can be helpful for programs

→ Foldr function, number of functions on lists can be applied using the pattern of primitive recursion

Ex:
```
f []     = v
f (x:xs) = x ⊕ f xs
```
⊕ ↓ Generic infix operator

```
sum []     = 0
sum (x:xs) = x + sum xs
```
v = 0, ⊕ = +

→ High order library function **foldr** (fold right) takes v and ⊕ as arguments

Ex:
```
sum     = foldr (+) 0
product = foldr (*) 1
```
0 → Base case; ↳ Recursion

→ Think of foldr non-recursively

Ex:
```
sum [1,2,3]
= foldr (+) 0 [1,2,3]
= foldr (+) 0 (1:(2:(3:[])))
```
Replace (:) with **operator** and [] with **base case**

$= 1 + (2 + (3 + 0))$

$= 6$

→ Some functions on lists, such as sum are simpler to define using foldr.

→ Advanced programming optimization can be simpler if foldr is used instead of explicit recursion