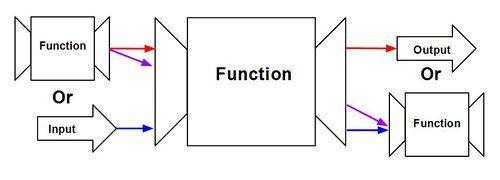
**higher-order**



take 5[1..]

1

A function is called higher-order if it takes a function as an argument or returns a function as a result.

1. **twice :: (a** → **a)** → **a** → **a**

**twice f x = f (f x)**

**Explanation:**

**twice (+3) 10**

**twice (\x->x+3) 10**

**Anonymous function**

An anonymous function is a [function](https://wiki.haskell.org/Function) without a name. It is a [Lambda abstraction](https://wiki.haskell.org/Lambda_abstraction) and might look like this: \x **->** x + 1.

Syntax

(\<args> -> <exp>)

Example (\x -> x + 1)

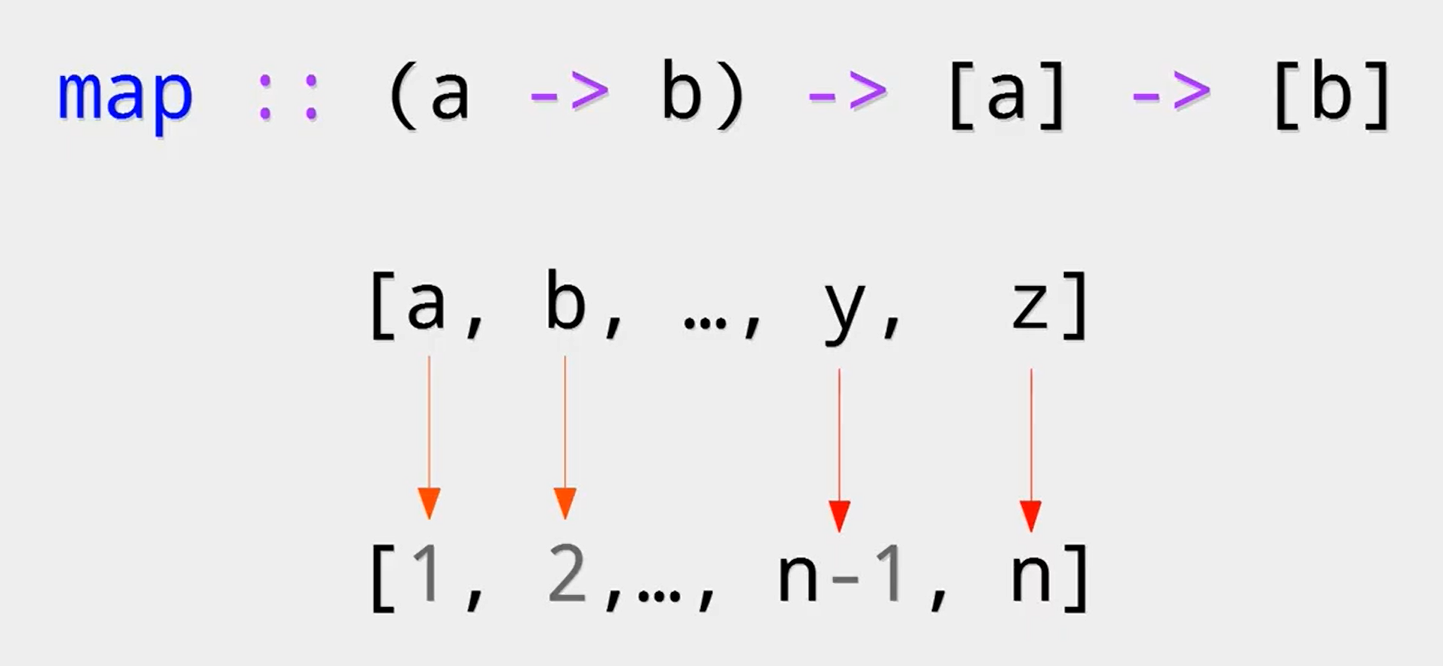
(\x y z -> x+y+z)

1. **Map** :The higher-order library function called map applies a function to every element of a list.

map **::** (*a* **->** *b*) **->** [*a*] **->** [*b*]

map f [] **=** []

map f (x**:**xs) **=** f x**:**map f xs



map (+1) [1,3,5,7]

map(+1)[1,3,5,7]

x:xs x=1 xs=3,5,7

1+1=2

\x->X+1

map (\x-> x+1) [1,3,5,7]

map (replicate 3) [3,4]

3,3,3 ,4,4,4

map(\(x,y)-> x+y ) [(1,2),(2,3),(5,7)]

1. **filter:** The higher-order library function filter selects every element from a list that satisfies a predicate**.**

filter **::** (*a* **->** **Bool**) **->** [*a*] **->** [*a*]

filter p [] **=** []

filter p (x**:**xs) **|** p x **=** x**:**filter p xs

**|** otherwise **=** filter p xs

**filter :: (a** → **Bool)** → **[a]** → **[a]**

**filter even [1..10]**

**[2,4,6,8,10]**

**filter (\x -> x>2)[1,2,3,4,5]**

**[3,4,5]**

**3)** **zipWith-** It takes a function and two lists as parameters and then joins the two lists by applying the function between corresponding elements.

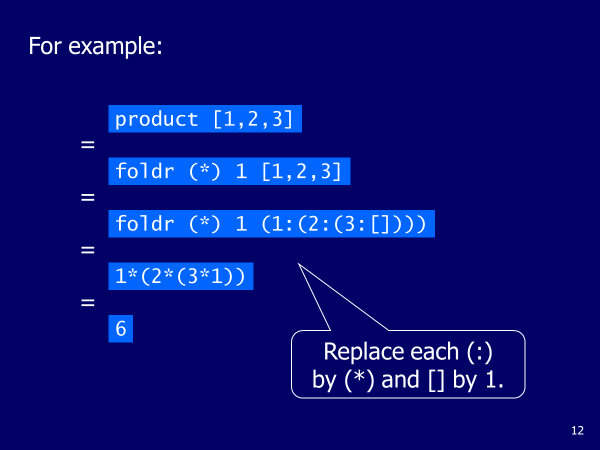
**zipWith(+) [4,2,5,6] [2,6,2,3]**

**[6,8,7,9]**

1. **The Foldr Function**- A number of functions on lists can be defined using the following simple pattern of recursion:

foldr (+) 0 [1,2,3]

foldr (\*) 1 [1,2,3]

****

**Exercise:1)** **Print all value which is divisible by 2 using filter fiunction**

**2)** **Use zipwith function with 2 list, Use anonymous function ,its definition is 2x+y**

**3)** **Use map function and generate square of all value of list**

**4)**

**4)**Define a function that counts how many times words occur in a text and displays each word with its count.

**xs = [1,2,3,4]**

**• ys = [5,6,7,8]**

**• zs = xs ++ ys**

**map odd xs**

**• 6 `elem` [3,4,5,6]**

* **filter odd xs**
* **filter even xs**
* **filter (\xs -> xs `mod` 2 == 0) xs**
* **filter (\xs -> sqrt xs >= 2) xs**

**zipWith' (zipWith' (\*)) [[1,2,3],[3,5,6],[2,3,4]] [[3,2,2],[3,4,5],[5,4,3]]**

**[[3,4,6],[9,20,30],[10,12,12]]**