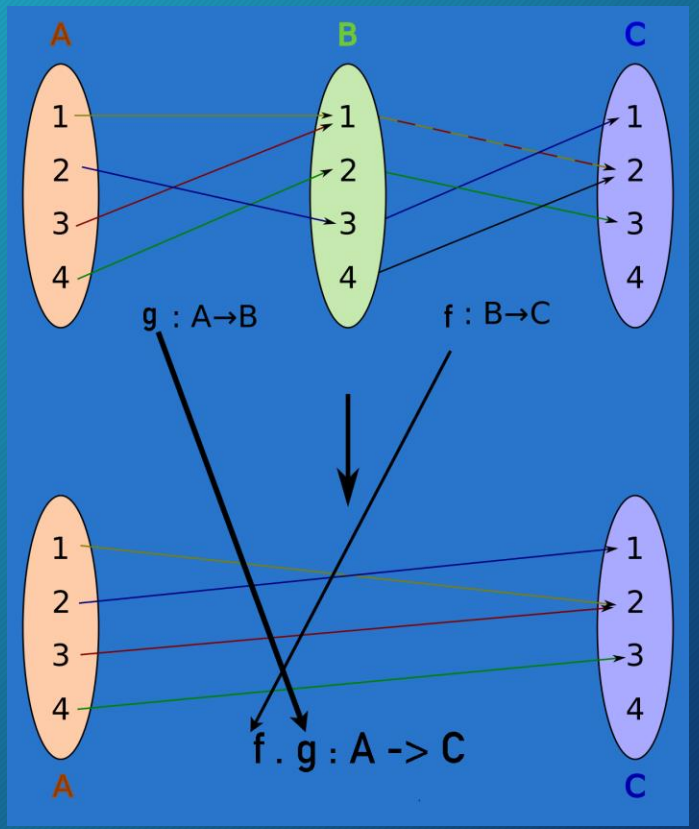
PROGRAMMING IN HASKELL

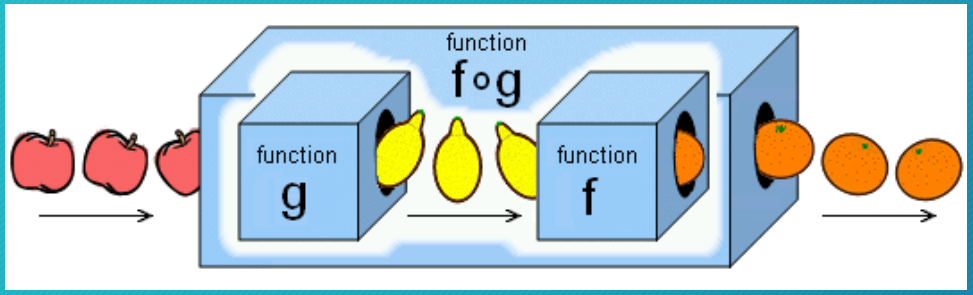
Chapter 8.4
Function Composition

Function Composition

We sometimes use one function after another and we can see these functions as one, composed together:

$(f \cdot g)(x) = f(g(x))$





Function Composition

Call g with some value, call f with the result

```
(.) :: (b -> c) -> (a -> b) -> a -> c
f . g = \x -> f (g x)
```

Input parameter of f **must** be the same as the return type of g

Function Composition – why

Often convenient to create functions on the fly
Could use lambda, but composition may be more concise

```
*Main> map (\x -> negate (abs x)) [5,-3, -2, 7]
[-5,-3,-2,-7]
```

```
*Main> map (negate . abs) [5, -3, -2, 7]
[-5,-3,-2,-7]
```

Function Composition – why


```
*Main> map (\xs -> negate (sum (tail xs)))
```

Apply to

[[1..5],[3..6],[1..7]]

returns [-14,-15,-27]

```
*Main> map (negate . sum . tail)
```

Apply to
[[1..5],[3..6],[1..7]]

returns [-14,-15,-27]

Eta conversion with Function Composition


```
f ( g ( h (i (j (k x) ) ) ) )
```

can be rewritten as

```
(f . g . h . i . j . k )   x
```

```
myfunc :: a -> b
myfunc x = (f . g . h . i . j . k )   x
```

can be rewritten as

```
myfunc :: a -> b
myfunc = (f . g . h . i . j . k )
```

Eta Conversion


```
answer :: [Int] -> Int
answer xs = sum (map cube (filter by7 xs))

cube :: Int -> Int
cube x = x * x * x

by7 :: Int -> Bool
by7 x = x `mod` 7 == 0
```

can be rewritten using the eta reduction

Steps to introduce function composition


```
answer :: [Int] -> Int
answer xs = sum (map cube (filter by7 xs))
```

Each function operates on the output of the previous function:

1. filter by7 xs → Filters numbers divisible by 7.
2. map cube (filter by7 xs) → Cubes the filtered numbers.
3. sum (map cube (filter by7 xs)) → Sums the cubed numbers.

Our goal is to express the function using functional composition and eliminate xs if possible.

Applying Function Composition

We use the composition operator (.) which allows us to rewrite:

```
f (g (h x))  ⟶  (f . g . h) x
```

Since xs is simply being passed through all the functions, we can remove it by introducing function composition:

```
answer = sum . map cube . filter by7
```

Break it down

- filter by7 produces a ***list***.
- map cube transforms that **list**.
- sum reduces the **list** to a single value.

As function composition applies functions from right to left:

```
answer = sum . map cube . filter by7
```

Is equivalent to

```
answer xs = sum (map cube (filter by7 xs))
```

Example Usage

If we run:

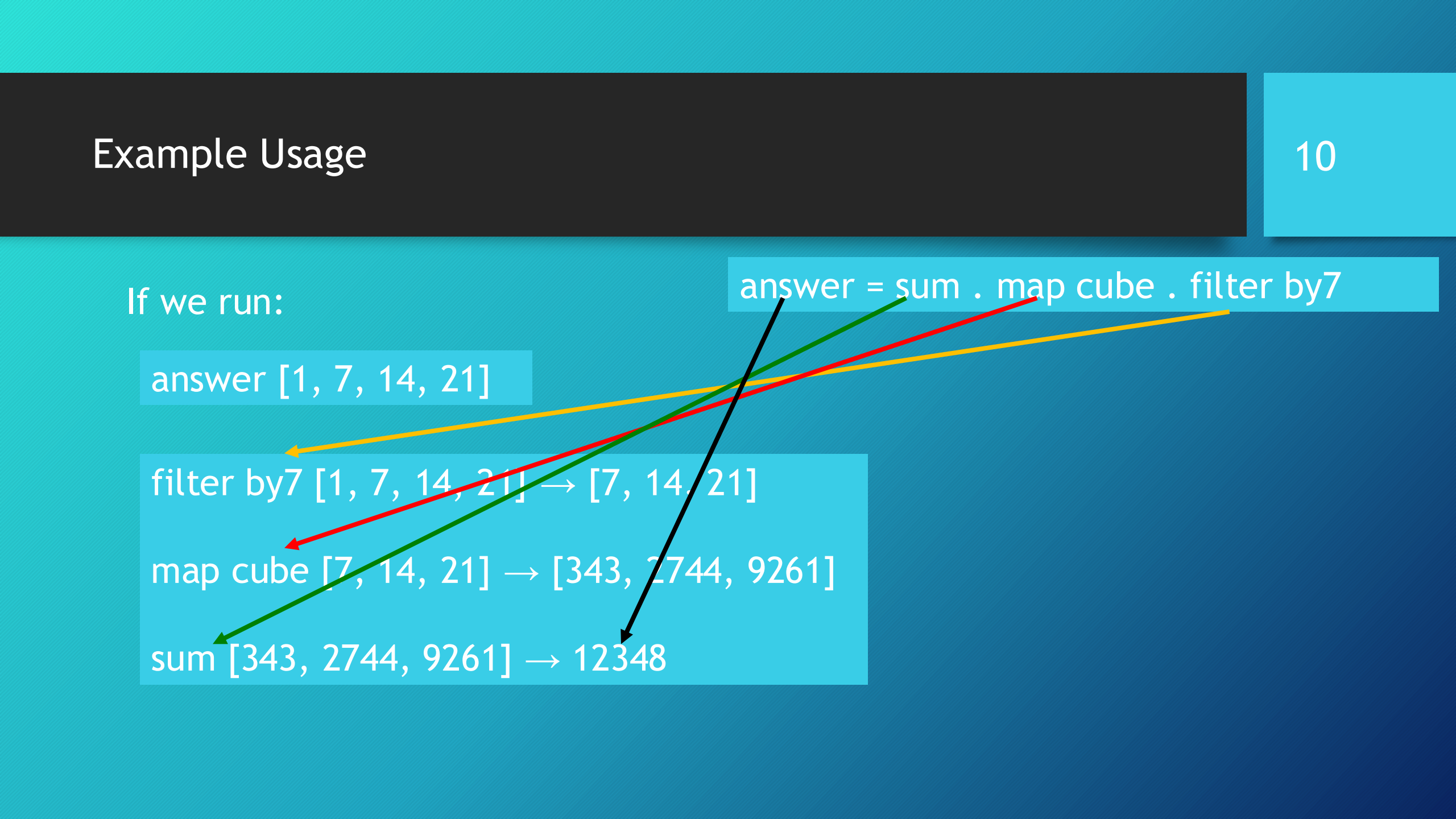
```
answer = sum . map cube . filter by7
```

```
answer [1, 7, 14, 21]
```

```
filter by7 [1, 7, 14, 21] → [7, 14, 21]

map cube [7, 14, 21] → [343, 2744, 9261]

sum [343, 2744, 9261] → 12348
```



More on Composition Operator


```
answer :: [Int] -> Int
answer xs = sum . map cube . filter by7 xs
```

Object-level definition - xs still named

can be rewritten , by removing xs when it is the rightmost term on each side of =

```
answer :: [Int] -> Int
answer = sum . map cube . filter by7
```

Function-level definition – xs argument is removed

Function Composition with multiple parameters


```
sumReplicatedMax :: Int -> Int -> Int
sumReplicatedMax x y = sum (replicate 5 (max x y))
```

To rewrite, note that :

- max 6 9 → Finds the maximum of 6 and 9.
  - Since 9 is larger, this simplifies to 9.
- replicate 5 9 → Creates a list with 5 copies of 9, producing [9,9,9,9,9].
- sum [9,9,9,9,9] → Sums the elements, resulting in 45.

The process

To rewrite a function with lots of parentheses using function composition

- first write out the innermost function and its parameters
- then put a $ before it
- compose all prior functions by omitting their last parameter (but not other parameters) and putting . between them

Function Composition with multiple parameters


```
sumReplicatedMax n x y = sum (replicate n (max x y))
```

Rewrite innermost function and put $ before it, Then compose all prior functions

```
sumReplicatedMax n x y = sum . replicate n $ max x y
```

As x and y appear on the right of both sides of the equals (=) sign, we can reduce by dropping them

```
sumReplicatedMax n  = sum . replicate n $ max
```

More on Composition Operator


```
fun xs = (filter odd . map square) xs
```

similarly can be rewritten

```
fun = filter odd . map square
```

Eta abstraction

Eta reduction

We try to eliminate the arguments as often as possible


ANY
QUESTIONS?