1 Fold & composition

1.1 First Folds

Given below is a version of `foldr`, simplified to only range over list of ints.

```
foldr' :: (Int -> a -> a) -> a -> [Int] -> a
foldr' f z []       = z
foldr' f z (x : xs) = f x (foldr' f z xs)
```

Using this definition, implement the functions below. Note they have been suffixed with a `'` to not cause name clashes with their (more general) alternatives in the prelude.

- Write a function `length' :: [Int] -> Int` which calulates the length of a given list of integers.
- Write a function `any' :: (Int -> Bool) -> [Int] -> Bool` which takes a predicate in the form of a function returning `Bool`, and returns wether the predicate holds for *any* element in the passed list.
- Write a function `all' :: (Int -> Bool) -> [Int] -> Bool` similar to `any'`, but which only returns `True` when the predicate holds on *all* integers in the list.
- Write a function `map' :: (Int -> Int) -> [Int] -> [Int]` applies the passed function to each of the integers in the passed list.
- Write a function `filter' :: [Int] -> (Int -> Bool) -> [Int]` which filters a list of integers based on the passed predicate. The predicate is represented as a function. All integers for which this predicate returns false should be filtered out.

Examples

```
Main> length' []
0

Main> length' [1]
1

Main> length' [1..10]
10

Main> any' (>1) []
False

Main> any' (>1) [1]
False
```

```
Main> any' (>1) [1, 2]
True

Main> all' (>1) []
True

Main> all' (>1) [2]
True

Main> all' (>1) [1, 2]
False

Main> map' (+1) []
[]

Main> map' (+1) [1, 2, 3]
[2, 3, 4]

Main> filter' (>1) []
[]

Main> filter' even [1..10]
[2, 4, 6, 8, 10]
```

1.2 Beginning composer

For this exercise, use only the definitions from the previous exercise "First Folds", function composition, and the following helper functions:

```
even' :: Int -> Bool
not' :: Bool -> Bool
absolute' :: Int -> Int
greaterThanFive :: Int -> Bool
```

With these building blocks *only*, define the functions below.

- A function `amountEven :: [Int] -> Int` which calulates the amount of even integers in a given list.
- A function `onlyOdd :: [Int] -> [Int]` which returns only the odd integers from a given list.
- A function `absGtFive :: [Int] -> Int` which returns the amount of integers with an absolute value greater than 5 in a given list.
- A function `anyEvenGtFive :: [Int] -> Bool` that returns `True` if there exists an even integer greater than five in the input.

- *Challenge:* A function `anyEvenGtFive' :: [Int] -> Bool` that returns the same as the non-primed version `anyEvenGtFive`, but does not use `any`.

Examples

```
Main> amountEven []
0

Main> amountEven [1..10]
5

Main> onlyOdd []
[]

Main> onlyOdd [1..10]
[1,3,5,7,9]

Main> absGtFive [10]
1

Main> absGtFive [-10]
1

Main> absGtFive [-10, 3, -3, 10]
2

Main> anyEvenGtFive [9]
False

Main> anyEvenGtFive [9, 10]
True

Main> anyEvenGtFive' [9]
False

Main> anyEvenGtFive' [9, 10]
True
```