DM580 – EXERCISE SHEET #5

(1) Write a function `apply :: (a -> b) -> a -> b` that applies a given function to a given input.
(2) Show how the list comprehension `[ f x | x <- xs, p x ]` can be expressed using `map` and `filter`.
(3) Write a function `lengths :: [String] -> [Int]` that finds the lengths of all strings in a list.
(4) Use `foldl` to write a function `intersperse :: String -> [String] -> String` that intersperses the given string between every string in the list, e.g.,

```
intersperse ", " ["foo", "bar", "baz"] == "foo, bar, baz"
```

   (a) Would using `foldr` in your solution give the same result? Why or why not? Write down your answer before testing it. Did you get the result you expected?

(5) The functions below are found in the standard prelude in Haskell. Write them yourself (without consulting their definition in the prelude). A *predicate* on a type `t` is a function of type `t -> Bool`, and we say that a value `x` *satisfies* a predicate `p` iff `p x == True`.
   (a) Write a function `flip :: (a -> b -> c) -> b -> a -> c` that swaps the order of the arguments in a curried function.
   (b) Write a function `all :: (a -> Bool) -> [a] -> Bool` that returns `True` iff *all* elements of the given list satisfy the given predicate.
   (c) Write a function `any :: (a -> Bool) -> [a] -> Bool` that returns `True` iff *any* element in the given list satisfy the given predicate.
   (d) Write a function `dropWhile :: (a -> Bool) -> [a] -> [a]` that drops elements from a list until it reaches an element that does not satisfy the given predicate.
   (e) Write a function `takeWhile :: (a -> Bool) -> [a] -> [a]` that takes elements from a list until it reaches an element that does not satisfy the given predicate.
   (f) Write a function `concatMap :: (a -> [b]) -> [a] -> [b]` that returns the list of elements generated by applying the given function to each element and flattening the result, e.g.

```
concatMap (\x -> [0..x]) [0..3] == [0,0,1,0,1,2,0,1,2,3]
```

(6) Write a function `search :: (String -> Bool) -> String -> Bool` that returns `True` iff the given string contains a substring that satisfies the given predicate, e.g.,

```
search (\x -> x == "cat" || x == "dog") "I like cats"     == True
search (\x -> x == "cat" || x == "dog") "I like raccoons" == False
```

(7) Write a function `while :: (a -> Bool) -> (a -> a) -> a -> a` that takes a function, a predicate, and returns the value resulting from repeatedly applying the function to the initial value while the predicate is satisfied, e.g.,

```
while (\x -> x < 1000) (\x -> x * 2) 8  == 1024
```

(8) Write functions `removeFirst :: (a -> Bool) -> [a] -> [a]` and `removeLast :: (a -> Bool) -> [a] -> [a]` that remove only the first (respectively only the last element) of a list that satisfies the given predicate.