Segment 1: Wheats and Chaffs

```
// Two versions of function to determine if first
// number is strictly larger than the second
boolean isLarger(int num1, int num2) {
   return num1 > num2;
}


boolean isLarger(int num1, int num2) {
   if (num1 > num2) return true;
   else if (num1 == num2) return false;
   else return true   // <— oops!
}
```

Segment 2: Reviewing `filter` and `map`

- `public IFuncList<T> filter(Function<T,Boolean> pred); // select`
- `public <R> IFuncList<R> map(Function<T,R> transform); // transform`
- `public IFuncList<T> takeWhile(Function<T,Boolean> pred); //select prefix`
- `public FuncList<T> distinct(); // remove duplicates`

Function	Input Type	Output Type	Output Length	Notes
`filter`	type A	type A	no larger than input	output elements are from the input list, in order
`map`	type A	type B	same as input	type A and type B could be the same
`takeWhile`	type A	type A	no larger than input	output list is a prefix of the input list
`distinct`	type A	type A	no larger than input	output elements are from the input list, no duplicates

Capture the following using map, filter, or a combination of these operations

```
transformation 1:
input: [-5, 8, 10, 0, -3]
output: [8, 10]

transformation 2:
input: [5, 4, 2]
output: [25, 16, 4]

transformation 3:
input: [“cAt”, “dog”, “DInosaur”, “armadillo”, “BOA”]
output: [“dog”, “armadillo”]
```

Segment 3: Other List Functions

Consider the following four problems:

1. Compute the product of a list of numbers
2. Compute the total number of characters in a list of words
3. Check whether there are any words longer than 10 characters in a list of words
4. Insert the word “brown” before every use of “bear” in a list of words

Which can we do with our list operations? How do we know?

Here’s an additional constructor for your FuncList class (if you are working in code) – don’t worry about how it works.

```
// a short-hand constructor for creating FuncLists just from elements
// e.g., new FuncList<Integer>(5, 8, 2, 3)
public FuncList(Object... args) {
   this.theList = new LinkedList<>();
   for (int i = 0; i <= args.length - 1; i++) {
       this.theList.add((T)args[i]);
   }
}
```

Put this in your Lec1Test file

```
<E> FuncList<E> toFuncList(E... args) {
   return new FuncList<E>(args);
}
```

Writing Aggregation Functions on FuncLists

```
// write a method called product that takes in a FuncList of ints and
// returns the product of those numbers

public static Integer product(FuncList<Integer> lst) {









}

// begin by writing a series of Assert expressions for product
Assert.assertEquals(5 * 3 * 10 * 2, Lec03.product(toFuncList(5, 3, 10, 2)))
```

```
// write a method called anyLong that takes in a FuncList of Strings and
// returns a boolean indicating whether any words have more than 10 letters

public static boolean anyLong(FuncList<String> words) {









}

// begin by writing a series of Assert expressions for anyLong
```

```
// write a method called remFirst that takes in a FuncList of Strings and
// returns a FuncList of the same strings without the first occurrence of dropWord

public static FuncList<String> remFirst(String dropWord, FuncList<String> words) {









}

// begin by writing a series of Assert expressions for remFirst
```