Makaela Fauber
CSCI 1335
12/18/24

Video Script

Slide 1
Toay, we are going to be reviewing higher order functions

Slide 2
To review, we know that functions are first class, meaning that functions are treated as values in scala, so they can be passed as arguments to other functions, or they can also be returned as values from other functions. When a function takes another function as an argument, we call that a higher order function.

Slide 3
The higher-order functions we'll be talking about are map, filter, and flatMap. It should be noted that the flatten method is also on here, but it is not a higher-order function. We will review it because it's helpful understanding how flatMap works, but it does not actually take another function as an argument, so it is just a method, not a higher order function.

Slide 4
To start, we're going to be reviewing the map method, which it iterates over a collection and then applies a function to each element within the collection. On the right here we have the implementation we were given in class for how map works on lists. We can see that it kind of recurs recursively operates on the list.

Move to Scala worksheet
Now I'll go through some examples and we'll see how how it works. First, we'll begin with lists like we did in class. Let's say we have a list of integers and we want to increment each element of the list by one. We can use map to do this. *show code* So now we have our original list and the new list that is the original incremented by one. It is important to note here that the first list has not changed. Lists are immutable, so this is actually creating a new list. It is not modifying the original. There are also different ways that we can also write map. We can use the binary operator format *show code* and we can also define functions separately and pass them to map *show code*. That's how math works on lists, but it also can work on other types of collections like maps and options. Let's start with maps. Here we have a map I have created where the keys are integers and the values are strings of pets. Now if we apply map in the same way as we did for lists *show code*, we get the function applied to the touple as a whole, which is not necessarily useful in this case. It'd be much more convenient to be able to access each element within the touple. We can do that in two different ways. First, we can pattern match with the key in value. *show code* The second way way that we can access each element within the touple, is with the shorthand '._' *show code*. These two did the exact same thing, just different methods of accessing the different elements. We can also see that map can also apply functions to options *show code*. Now, we've seen how map goes through the elements of a collection and then applies a function to it.

Slide 5
Now we'll look at the flatten method, which to reiterate it is not actually a higher-order function. It returns a single collection by merging child collections. It's important to know so that we can understand how flat map works here in a minute, but again it's not a higher order method.

Move to Scala worksheet
Let's see how it works on lists. Again, like I said, by flattening a collection of collections into just one collection, it's easier to work with. So let's say that we have a list of lists, which is hard to work with. We can flatten it so it is easier to work with. *show code*

Slide 6
Then we've got the filter method, which we pass a function that returns a boolean value in order to sort through the items of a collection. This function with a boolean value is called the predicate.

Move to Scala worksheet
Let's see how that works. We will also start with lists for this one. Let's say I only want the even numbers out of our original list. I can apply a function that checks if the elements are even to filter the list *show code*. So that's another way we can pass functions to filter to sort through our elements and only output the ones that we're interested in.

Slide 7
Lastly, we have the flatMap method, which maps the collection and then flattens it. Since we've already gone through map and flatten we should have a decent idea how this works, but let's look at it in action.

Move to Scala worksheet
Let's say that we want to add to the original list and increment by one. If we use the map method here, we create a list of lists. *show code* This is also not very useful. We can always flatten it again to get rid of the child lists *show code*. The flatMap method essentially does both of these steps in one, mapping and flattening and will output the same code *show code*.

Slide 8
So when do we use map versus flatMap? Basically they do very similar things, but if the collection results in many collections, you want to use flatMap to flatten all of those. Look at how you're modifying your collections and what the output is to decide whether you should usen map or flatMap.

Slide 9
Thanks

Slide 10
Credits/sources