Final Script

Hi everyone, my name is Preston Dotsy. I am a senior and I'm just about to graduate this term so I'm very excited right now. My partner is Pankaj Mehra. We made this project together, but unfortunately his voice is kind of giving out on him right now, so I'm taking over the narration duties for this. So between the two options we had for doing the mini project we were thinking about doing the exploring of like rewriting some code to like, still incorporate concepts we've been learning about this term, but actually looked at the other option of looking into an existing framework that uses a lot of these concepts as well, and a few of them like really caught my eye. The one that caught my eye the most of those was the lift framework, so I decided to dig in a little bit deeper into that and explore how it actually uses a lot of the concepts we were talking about in this class out in industry.

So on the agenda I'm going to talk a little bit about the background of list and like how it got started and what it's. Trying to accomplish. The three main points I'm going to go over are abstract data types, high order functions, and functions as values. The lift frameworks shows these three concepts throughout most of their code, so it's actually a great example to use for this so. I'm excited to show you a little bit about it.

OK, so First off with background, what is lift? So lift is a free and open source web framework that's designed for the Scala programming language. The whole thing is entirely written in Scala, and that was designed so. That it could be. Portable to any Java virtual machine and run with any hardware other existing frameworks. It's meant to be like a. Plug and play kind of thing. It was originally created by David Pollack, who is dissatisfied with certain aspects of the. Ruby on Rails framework. I know that like through my reading of it, he definitely drew a lot of inspiration from. Yeah, but there was some things that he thought could be done better. And so he decided to go. in a different direction from it. Lift was launched as an open source project on the 26th of February 2007, and it was actually. Licensed on the Apache license. An example of the framework commercial use is Foursquare, which I don't know if you know what Foursquare is, but it's a cloud based location tag that basically lets customers and businesses. Become aware of each other, like through their phone. It uses like geospatial data to tell you like which customers are close to you, what businesses are close to you, and it's become quite successful, but that that was their first major commercial applications list. And so as a framework, it is specifically designed to write web applications. Like I was saying earlier, it's very similar to Ruby and Rails. The reason why it's similar is that it favors convention over configuration. But the main difference between them is that. Lift is designed with the view first architectural pattern and Ruby on Rails. Is the model view controller pattern that a lot of other frameworks use, and so it's a little bit of a deviation from that. The use of view first is intended to make the development more designer friendly, which is inspired by the wicket framework. So what is Lyft actually meant to do? So according to the website and looking into it, he had five core tenants that he really wanted to incorporate into lists, so he wanted to be designer friendly. He wanted to be really fast and efficient. It wanted to be secure, especially against like online attacks from other malicious users, easily scalable to handle lots of web traffic, and easily extractable and so it has to do with security as well. He really wanted to abstract while. The data away from the user and said rely on functions in the background which communicate with the front end. And so how? Do they actually accomplish these goals, and how does it relate to our class that we're currently taking? So the lift framework is its ability to accomplish its goals through the corporation of several key concepts which we have learned about over the course of the. Semester in 30. 155, namely, abstraction and data types, which relates to security. Use of higher order functions, which as you'll see in a little bit, are everywhere in web applications and treating functions as values, which is something that we have heard over and over and over again, but actually through exploring this it helped really sync that idea and look deeper.

All right so. First, abstract data types. So let's go over what abstract data type actually is. For the sake of brevity, I'll call it ADT from here on. Out so an ADT is a type or class for objects whose behavior is defined by a set of values and a set of operations. The definition of ADT only mentions what operations are to be performed, but not how these operations will be implemented, so it's very similar. I've been doing a lot of Java lately and abstract classes are used all the time in that. It does not specify how data will be organized into memory and what algorithms be used for implementing. The operations and. The reason why it's called abstract is because it gives an implementation independent view, so it's abstracted away from the actual guts of how it works. It just gives you a definition which you can use in other parts of your code. Right and then how do we use abstract data types in this? Class so ADTs are used heavily throughout the labs, as you probably have seen. ADTs are especially prevalent in the ASP dot Scala files. Which I found. In them. The abstract syntax trees use a ton of abstract classes within them. And yeah, so a good way of thinking about it is in the AST objects. They declare the classes, but they don't. They never actually show what they do because we ourselves are actually the ones implementing those. In our like lab dot Scala files. Instead of implementing the methods of the abstract classes and the classes themselves, we had implemented the class methods within the lab Scala files that I previously said. And objects and communicate with each other to. Carry out their. Functions by incorporating this abstraction into our lab files, we're able to increase the ease at which we make changes to methods. Hide private data, increase clarity within the code, and make our code more secure by hiding key details. Right, so, uh, as an example of that, I actually took a little bit of code out of. The St dot school file. From web 4. So in this, uh, you can see right here that it's a sealed abstract class UOP. And so with the and you can actually see down below there's the BP one too, but I used the, uh, just because it had a a more concise example. So in that we have neg and not and. Here we go. You could see. So in this right here we're defining it in abstract class, but we don't actually see what. It does yet. And so if you look into the. Lab 4 dot. Scala file I looked into the function type of. And you can see here we have both unary and neg. Actually implementing everything and they're laying out how the guts. Of it work. So in this we get an actual definition that we can use in other areas, but it pulls from this implementation and so by splitting it into different objects you're able to allow the objects to just communicate with each other instead of having it. Be like a monolithic, like huge class with everything all together and so those increased security. It makes things simpler. To debug honestly, and I've been through a lot of my classes lately, I've been realizing that if you're doing an object oriented language, creating lots of smaller objects that all talk to each other is so much simpler in the long run than creating these huge files all at once. And so lift also uses 80s as well. I'm realizing most web frameworks actually use a ton of all these. Concepts that we've talked. About but the most obvious use of ADTs with lift framework is how it abstracts with the HTTP requests. And so instead of having it all carried out on the front end, it actually abstracts. Way to back end servers that hold the implementations so it calls it using abstract data types, but the actual implementation carries it out on the backside, so there's no user facing implementation at all. And so I lifted a little excerpt away from their site, and so lift out score seeks to abstract the HTTP request response cycle rather than placing object wrappers around HTTP requests. By incorporating this abstraction into their lab files, they can increase the ease at which they make changes to methods hide private. Data increase clarity within the code. And make their code. You care about hiding key details. And so I actually have been referencing the Lift Handbook for a lot of the code examples on this. And so I have an example here of how it does that and so. As you can see, there is no actual implementation done, it just has these abstract classes that are called on the front end and then it will actually send a request to the server which carries out via the functionality of it.

And on to higher order functions. So first let's define what a higher order function is, and so in computer science also in mathematics as well. What we're talking computer science right now. A higher order function or an HOF Is that function that does at least one of the following so it takes one or more functions as arguments or parameters where. You want to call them. Or it also returns a function as result, so it doesn't have to both of them, but just needs to carry out one of those. And so how do we? Use this in our class. So high order functions are one of the most concepts we covered this semester with the use of higher order functions, we were able to incorporate another layer of abstraction into our code and choose the layer separation between our classes. Higher order functions also allow us to use the return values from the functions called within the HLS parameters and is streamlined and easy to parse. By incorporating HOS, we are also able to facilitate the function or sorry not the, but two facilitate function composition which is essentially composing new functions out of multiple other functions. This is possible to this act that Scala treats function as its first class values, so that's a very important point, and we'll we'll touch on that more in a little bit. So as an example, I pulled this from the lab 5 dot. Scala file and so it is the first map list that occurs in. There, and so as you can see, it only has one case within it case AWDWBS. So the first function in that call is actually flat map, but within the parameters we also have a map as well, so it's a function that's calling another function within it, and so instead of. Having everything like out laid out in the open, it's just calling another functionality from the library and then that value that's returned from it is used within the first function, so map returns a value or a series of values. And then flat map works on those afterwards. And then so let's look at how Lyft also uses high order functions, so Lyft heavily relies on HS as well as the entire framework is written. Install it and as such uses the same libraries that we. Have used. In 3155, so you'll see a lot of very similar code all throughout the list code base. As we've been using in 3155. Yeah, Scala is through all my research. It seems very good at scaling. So it's very popular in a lot of web frameworks because compatibility with Java and its ability to scale up very. Well and the building of web applications with lists, many hos are used to facilitate the functionality. The same applications common hos used in the lift framework include map, flat map and filter, which I think you're probably pretty familiar with those by now. Common uses of these. The lists are in the organizing and listing of user data and retrieving internal data from the applications. Back end. And so I also pulled this from the. Handbook as well. So this is another good example of a high order function. So as you can see, in the case class down at the bottom you have the S2. It calls function full, but then within that we also have programINFOS which is another function that it's calling on itself. So it's very similar to what. We're seeing in class. Code snippets that I saw within the book contained higher order functions.

All right, last topic functions as values. So Professor Chang said this over and over again, but I'm going to just give it a quick recap of what he means by that when we say functions or values. This concept is one of the defining features of the functional programming language. All functional programming languages have this concept as a core tenet of how these languages execute. The idea behind functions or values is that. Functions are not separate from the values of the return, but rather are the values themselves. By treating the functions themselves as values, we can now use functions as parameters within higher order functions, so it loops back into higher order functions also as well. This allows us to create cleaner, more efficient code as well as increase abstraction and rely more heavily on communication between objects. Instead of making monolithic classes which are much harder to debug than a collection of smaller objects. That cooperate. And so, what? Like how does? Functions as values fit into Willy map 31. Obviously, presenting said this over and over again, she really had really tried to drill it into our heads, incorporating this concept into how we think about functional programming has been vital to creating solutions to our assignment system master, especially when writing code for the lab assignments and our labs. You'll be using many high order functions. The reason why we can use the parameter functions within higher order functions is that installer functions are treated themselves as first class. Values that means that when we pass the function to another function, we're not actually passing the functions details to the to the HLS, but rather passing the value that the function itself represents. And down the bottom just another reminder that the reason why this works is because function are values. So I pulled an example. From which one was the log? File and so I chose for this one instead. So it I thought this was a good example of it, because fold list actually incorporates another function loop and then in case node LDR calls loop. And which within loop calls loop and so it's a higher order function that is treating loop as a value within the higher order functions parameters and so it's this would not work if it did not treat functions as values because it wouldn't know what to make of it, so it's able to parse through the result. And actually see it as a value. And you see this over and over and over again in the class. It's been drilled into our heads because it's a thing that it's like. Fundamental to what makes functional programming actually work? And then let's go into list and how they use it. So just as we have used functions as values in our lab assignments in 3155, lift also follows as concept and its framework. Through its use. Of Scala higher order functions. Practically all web application frameworks make heavy use of HOFS. Which can only execute if the functions in the HOF parameters are treated as values. Well, this takes advantage of this concept to use. Functions as values within HOF parameters. To facilitate super code implementation and ensure code security when deployed to clients and a lot of developers use this, all libraries which make their framework run smoothly and so this was also pulled the handbook as well. I thought this is a great example of it because it actually has three high order functions in a row and so it's using it's converting. The like the return value of these higher order functions into a result and then performing more higher order functions as well. So it's basically like 3 nested high order functions with themselves, and the reason why they can do that is because it's actually treating them as values and not as functions.

Right so quick summary. I'm running out of time right now, but. The use of ATS aren't just a thought exercise that we used in class, but they're super like integral into part of building applications and other pieces of technology. Lift framework is no exception and uses this to their benefit as well. Higher order functions are also ubiquitous when working in functional programming languages like we've talked about them a lot in this class, but I didn't realize how much of a great degree they're actually going to be in industry like looking through all these frameworks, high order functions are everywhere, almost every like every little function that. Seen has been a high order function. If it's doing anything more than just like assigning a value and then the last point. Functions are values and the reason why this has been repeated so much in this course is that with if in functional programming languages, if it wasn't treated as a value the way that these languages work wouldn't work. Anymore like it it. If high order functions didn't exist, which they wouldn't without functions being treated as values, most other parts of the language become crippled. And so that is the very base layer of how everything works in this. All right, thank you for watching. I'm sorry I'm a couple minutes over, but I really appreciate it. So for me and Pankaj good luck with all of your finals and thank you for having a great class. Alright see you guys.