And then we have this one. Just different links. You could go to this in the Rubies homepage and if you go here. All of these links you can follow. There's lots of tutorials and more documentation and the book that I included is on here and the text form and also have some of these links. This is a. Website is kind of like a. Web-based ebook for Ruby. Lots of nice stuff. I could go here. You know what? Comments? It'll take you straight to that section. I have some of those links already. Here. So these are just links to different parts of that book, right? Variables, variables, scope methods, etc. So if you need that. Again, I'll update. The. Somehow I'll get you access to those files. But from the assignment what I wanted to do today. Let's start with a really quick. Explanation of the instructions. And then. Move on to the Ruby lecture. When I actually talk about Ruby as a language, you know that the. How to use it that is? And then if we have any time left, go back to the instruction that spend some time looking at the source code and to take a deeper dive at the instructions. But for now. Let's start with the assignment instructions.

So here is the grammar. This is a grammar that all of our assignments are going to be based off of. At least not all our assignments. All of the interpreter series assignments are going to be based off it's very small grammar, fairly straightforward to implement. And for this very first assignment, we want to build a Lexus. So remember, we spent the last class. One of the things we did was talk about one elector was we we want to look through our source code file one character at a time, and try to identify all exes at the final identify and label them. So the way we're going to do that with our lexer, I have given you a mostly complete token class. I think you almost all the code you need. You only need to modify just a little bit, but the token class. Is the object that you use to encapsulate that information. Or save it. So basically as you scan through the file a character at a time, every time you think you identify Alexi. We need to make another token object and you need to save the leg scene. I saved the label. That's what open object is for our Alexa. And that's all your class needs to be able to do, yes? What's the label? So if you wanna make a label called ID then you would save it as Heidi. If you wanna make a label called add operator you save it as add operator. And there are some examples of the source code file as well. When I when I tried to do with these assignments. Everything that you need to do to complete the assignment I've done it somewhere in the file for you, so if you have an example of every type of thing you need to do so you should be able to. With the documentation plus my explanation today plus wasn't the source code file, it should be more than enough help to complete the assignment of plenty examples you could make sure work off of. Oh. So here just an example of. Literally example, let's suppose that we're scanning a file and we we have this word in the file doc, and that's a. Maybe we choose. That token category is ID. That's the category that we chose for that. It could be variable, could be identifier, whatever you wanna call. Let's just pretend that's the label we chose to apply. Then what we do, we'd make a new token object and we would save this. There's a type variable that's for the token. And there's a text variable left in the left seat we would say ID into type. And into text we would save the word dog I literally capital B and capital Capital G and we return that object, that object. I would encourage you. To test your program as you go, that's just a general computer programming advice, but. I would encourage you not to ride the entire program and then Friday love with hope. You have no idea where the problem is. Well, I would encourage you to do is do. A little bit, for example if you know you have different. Tokens different categories are lexemes. Maybe just try to detect one of them right the code so you can successfully detect only one and then test it. See if it works. Give it a source code file that only had that type of code in or maybe has a mix and match. If that is, can you correctly identify every single time? Great, move on to the next one. Yeah, great. Deal on from that. Then. If you do make a mistake, if something does happen, you know exactly where it is. It worked until I got here. Here's my problem. You know exactly where to go in the cold to find the issue and troubleshoot. So that's my suggestion. Your leg, Sir. The file name is tiny scanner. What you're last is going to be in here are the things you need to do. I've done some of them. You'll have to do some of them. And I try to tell you what I've done here. So the constructor for your life, Sir. Should we passed a filename? That file name should be the source code where your program at? Where's the program I'm part or Alexi in this case. So that's what that should mean to be the file name. And then what it should do is open that file and scan the first character in the file and save it in your look ahead buffer. So we have a buffer and instance variable. Vaccine. That's a that we use to store 1 character at a time. So as we scanned the file was scanned, that's character put it in their skin that care to put it in there? It should only ever have one character add time. Now I already did that, so when I've written already opened file and put the first character into that very. What what you need to do? Right now, if I give you a file name that doesn't exist, the program will just crash. No good. You need to fix that. Like that should not happen. If I give you a file name and that file doesn't exist, you should gracefully terminate the program. Maybe print some message. Hey, I'm going away. Sorry, that doesn't exist. And then close it nicely. So the program should not crash. That's one of the things you want to try to do for this assignment. There's another method called next character. Really close to the methods we saw in the book. We're talking about the Lexa, right? So next character and what next character does. Simply takes the next character in the file, puts it in your character buffer. So whatever was in there is gone. Which this one in this. If you need to save stuff, you got to put it somewhere else. You need an accumulator, you need another variable, you can say stuff. I already did that, so once you get to the end of the file instead of a character, it'll actually put EOF in that very. Next token. This is the the bread and butter of new program. This is the most important method that you need to complete. I've done a little bit of it. I give you an example in here, but this is where you identify all your totals. This method should be able to identify every single type of token in your grammar. And what it should do? As it identifies the token. It should create an instance of the token object, populate those two variables we talked about earlier appropriately, and then return that's open object. That's what this method does. It identifies one, so every time you call this, wherever you left off in the file, they'll start scanning right there and it should return the very next token it sees. I didn't do a test or file Ruby script that will test your program and all it does is call that over and over again until there's nothing left until the reason end of the file. So you have to complete that. Contiguous whitespace, so whitespace immediately after it, whether it's

Or tab or space. Any sequential whitespace characters should be combined and saved as a single token. I read it for you, and as a matter of fact, that's the example that I use to try to help show how to do it. I've done that before and then exercise it will throw that away, but still, for now we'll just save it as a token and return that token. When you reach the end of the file, you should create another token object and we'll just call it an end to file token. I've done that for you already as well. Finally, at the bottom of the file, I have a couple of helper methods that you can use to help you identify the different types of lexemes you might see. So I have a. 3. Methods at the bottom that use regular expressions that if you give me a character, I'll tell you if it's a number or a letter or a white space character, so you can use this to help you identify. Characters as you're trying to put tokens together. Kind of. Character. What is it made? A number of characters? A letter character? You can use my methods if you want to, or you can write your own. But I do have these available that you can use already. As I mentioned earlier, I I did include a test script, a Ruby script that you can use to test your code if you want to. The time being, what it does is it creates an instance of your lexer. And then it calls your next token method over and over again until it reaches the end of the file. And every time it identifies a token, it saves that in a file, so you won't see it. It doesn't actually create anything. If you run it. You have to look for that file and then it is called tokens. You can just open that file and you can see everything I print in there. That's the output from your file. That's what you recognize. If you want to see what your program is doing, house behavior, just open that file and you can see what it discovered. You could also, if you wanted to, you could modify it so it could print stuff, but your file should have something like this in it. Next open is ID, next, just like with the book has just list, I list all the pairs at times totally lexingto, Calexico, Calexico. Alright. Any questions? About the instructions. In that case, let us move on, Ruby.

So I'm gonna leave this open 'cause what I wanna do. I wanna go back and forth to the terminal. So I'm gonna leave it like this so I can easily go back and forth. Initially. And I thought this past for several years now, and the way I initially made this PowerPoint, it was supposed to be. Like us going through some cereal class, right? Let's pretend we want to make a printer object. What do we need? We variably we need measurement, and that was just terrible. Nobody ever liked it, so we won't do that in here. But I I I do keep this 'cause. It does cover most of the important things we need for the assignment. So we are going to talk about these things, but we don't really care about the example too much. And to start with, let me, let me. Go here so. Point number one, first of all, you can do everything you need from locations. You don't need to do anything locally, but you don't want to. Everything I'm going to do here you can do exactly the same thing Invoker, and we've been modifying the file directly, or Valkyrie, and even has a shell. If you want to work in the shell you can do that too. And it's got her interactive movie editor, just like I'm going to show you today, point number one. But if you want to do it on your own, a lot of people prefer to work on their local machines. More than welcome to do that if you have. If you have a Linux machine, so this is a Linux, have Linux installed here, and you need the Ruby interpreter, all your Ruby stuff. You would simply type in sudo. App install. This is for uh, a bunch. So this is for a bunch. If you have Sentosa, Red Hat or something else, the commands a little bit different. But if you have those, you don't have to install stuff soon. I have to install Ruby Dash. Cool. So Ruby fool is the name of the pack you need. I already have it. Open my camera timeout. And you see I already have it, but this is the name of the package you want to install. That will install everything you need. That will install everything you need to do. The Ruby is on it now if you have a Windows machine. I would encourage you. To enable WSL. If you don't know what it is, Google it. I promise you, it's not that hard to do enable WSL. Open in a bunch of terminal. And then do exactly what I just did. Same thing if you're on a Mac. You can just open your shot. I think if you on a Mac you already have Ruby installed. You don't have to do anything special. It doesn't matter what version you use. We don't do anything really fancy with Ruby so. That gives us all the tools we need to do our homework. Now let's start here with variables. So variables in Ruby, Ruby is. Ruby is. A dynamically typed language. The data type for variables in Ruby is inferred, so you never have to declare again, never have to say what data type you want to variable to be. So unlike Java, we have to index for example. You don't have to do that Ruby, you say X. Equals and then Ruby will look at the right hand side of that assignment statement, whatever that is. Ruby will use that to determine what data type that variable should have. So this is not the same as saying that variables don't have data types. They do. But you don't have to say what they are. Ruby declares that data type for you. And since a dynamically typed data type for variables can change. So on line one, and you know, why don't we look at example X? I'm gonna open up my eyes so I RB is the program for the interactive Ruby editor. I really encourage you to use this to test and troubleshoot your code. Some of you may want to use fancier tools you can you're more than welcome to. Root Eclipse has a really nice. Ruby set of tools if you still need the interpreter, but if you have the interpreter. Eclipse has a really nice set of tools for working with Ruby, but in my opinion I think it's too much. I think it shows you too much. It kind of distracts you from what you're trying to do. I think less is more in this case because you don't get distracted by all the fancy stuff. So here's my interactive Ruby editor and I can do something like let's make a variable. X = 10 great typing X. Play the value of X. Now they're

I can say X equals Dom. Dom is a string. It's not a number. It's OK, Ruby is OK with that. On the very

I can say X equals. 1234554 whatever. It's OK now. It's a list, so it can be whatever I want to get in, changing every line if I want to, and Ruby every time I see an = Ruby will evaluate the right hand side of it and potentially change data type if it needs to. Dynamically typed language. It'll change it on the fly. It could be anything it wants to be. That's a good and a bad thing. Constance and Ruby. Begin with a capital letter, so the first letter in the variable name. And it's a capital letter that indicates that the constant and Ruby. Now the the custom is to make everything capital letters. So you're going to declare a constant. Most people will make everything capital. But it's not a requirement, right? You can just as long as the first letter is capitalized as all you have to do. Global variables in Ruby taking with the dollar sign. So a dollar sign. While it is a special character, it is also part of the variable name. So we're going to have things like$ Zach symbols. These are actually part of the variable name Ruby and they mean something. So the first one or two characters in a variable name and Ruby has some special significance sometimes if it's special symbol, so here. The first character is$ That's again it's part of the variable. Then you can't exclude this if you're referring to the variable, it needs to global variables. And finally these two. Someone told me in the last class that Java calls at the same thing, so maybe ignore what I said. But in Ruby? I'm. And Ruby. Instance variables. And instance variable and Ruby. Is equivalent to that in Java. Right, so if I were to create a couple of. If I were to create a couple of. Objects my Java class objects. I would have multiple copies of these variables in memory, so that's what an instance variable is in Ruby. No, I thought. I thought they called them class variables in Java, but apparently they don't, so. Disregard that and Ruby. It looks similar and we'll go over this in a moment, but do something my class. You know, my Ruby class. And a Ruby would do it like this. So I don't need the data type. Um. And I need to append the at symbol. To indicate that this is an instance variable. So every my Ruby class object that I make is going to have its own copy of X and name. That's what an instance variable is in Ruby and Ruby. The very first character. Of that, the area has to be an at symbol, and it's part of the variable name. So if I'm referring to and I can't exclude it, it's actually part of the name of the variable. A class variable, on the other hand, and rudely using being with two axles and the class variable is similar to the eye. Idea of a static variable in Java. It's not exactly the same thing, but it's similar to the idea in that there's only one copy per class, period, regardless of how many objects you have. So in Java if I do this. Uh, let's say stacking count, right? So maybe someone in my constructor? Public my job class. Now I have account plus plus right? Maybe that's something I do? So every time I create an instance of this object it increase increments that that variable, but I only have one copy. In Ruby? We do it like this. So Ruby. That's how we do it. Two app symbol hey, what did you just do? Two act symbols. Indicates that I want this to be a class vehicle, another version of this example. So if we were to do. My class C1 equals. Move. Right. So they got one. I can make two or three of these, right? 2. So I do something like this. Then what I have is every. One of these has named C1 name. Right, C2 has his own name. And its own value. And then C3 also has his own name. And it's all values but. All of these jointly share. Napa. There's only one count. Every time they reference count, they're actually referencing that one. They don't have their own copy of the option that's gonna class there is. And Ruby. Equivalent. But we look almost like this, so that's the idea. OK, so instance variable here, one accent. Class variable here. All of the objects there. That one single variable 2 axles. Any questions about variables?

So in Ruby I think I could be wrong. I think by default the the access level for variables in Ruby is public. So I think you can access them by default anyway unless you make them private in Java I think protect this isn't all access on left. If you want to just like in Java, you can make access or methods or or. Get and set methods for your variable. So I wanted to hear, you know, I couldn't make it get X and a set X, or I get Nyanda set name. And that's how you're supposed to use Java as well, right? You're supposed to make this private and you're supposed to make access or methods for us. So you can do that or Ruby or you can do something like this. You can create this attribute accessor label here. Followed by a colon, followed by the variable name you want to apply it to, and that that allows you to use dot notation. So instead of saying for example. If we had. If we had. Get main method. And it just returned. That variable for us. Same thing in Java, right? We had that. We haven't yet named method. To return that, and maybe we have a set name right? The phone set name. We could call that if we wanted to or. We could label the variables with these attribute access or labels. And then if I want to interact with them. If I do something like that. Something like that? Then when I'm interacting with this object so I could have X equals my routine. Then I could do something like this X dot hope. My reading class. Something like this my review class dot X so I don't need to call the method, I could just use dot notation equals 22 more. A = C dot reference that so that attribute accessor label allows you to interact with. Whatever variable you apply that to. Selection savings. And Ruby. You don't need. Parentheses or semicolons or colons. Sometimes you can use them, but they you don't use them for the same thing in Ruby that you would have row. So for example in Java you're required to put a; After every instruction at that indicates that's the end of the instruction, and Ruby it's optional. If you want to put one, you can. But you don't have to. Now, if you're trying to write a one liner, then you do need it, but that's like it behaves kind of like a delimiter and Ruby. But it's not wired so you can see right here in this example. I'll have to word if. Followed by Boolean expression and this word is optional. If you want to write it as a one liner. You can write the word then. This is what Ruby uses to separate your Boolean expression from the body of your hip statement. They need some way to separate that you would use the word them or you just hit enter a new line in another way to separate this one Java you have unique parentheses. Not quiet Ruby and Java that parentheses or how you delimit your expression from the body of your statement. Really you just use it either a new line or the word then or; then you have your instructions and your expression, whatever the the body of your statement is, and finally at the end. We don't use the curly brackets either, Ruby. So in Ruby, every block of code. Begins with whatever keyword you need to begin that type of block, and ends with the word end. So we don't need brackets and Ruby as soon as Ruby sees some type of keyboard that indicates a block level instruction. Look for the word and it'll just all those things are the body of that destruction until it sees the word in. This is wrong. It should be a; so let's take. Simple if X = 2. Why? What do you print? Yeah, so it's not going to print anything it does. X doesn't come even know what would his exo. Change in X2. Run that again. There we go so we can see what's true. It printed it if I wanted to write that as a one liner. I could use the word then. Very good. I could use a; So a; Remember we can use that as a delimiter. Had also worked for instructions. So here. Let's say I did that and then. Brent. Two minor. So it's going to look for that end tag. That's how it ends that block. Or. We could use a; As like a delimiter to indicate that's the end of 1 instruction and the beginning of another. Yeah, you can use it daily if you want to. Right here from one or two. So it's not required, but you can see it's not wrong to use it either. I can choose to use it. Let's suppose I wanted to do something like this. You can use a; If you want to. It's not wrong, but it doesn't exactly do what you think right and Java for example it's a requirement. You have to endure instruction with a; And Ruby is a delimiter so you want to use it to set right here instructions so that you can put all your instructions on one line. If you choose to use it anyway, that's OK, it's just a limiter. It's no different than a new mind. I'll just treat it that way. And even right compound statements. So as far as what year did statement can look like, it can look like any add statement you would write it off and that is greater than one. Or X is less than 22. Then print. Compound. Turn up sign right. All those things. Enough. Those things are true. They have a print that extra data structure. So here's what you have to be careful with. So what about the else clause? And else if clause. If you have those types of clauses then you have stated the elk laws is pretty straightforward. It behaves like a thing. It doesn't do right at the same way you think it does. The elsif clauses behaves like it should, but it's written differently. It's one word without the E and you have to be really careful here. So unlike, let's say Java for example, which is a compiled language, kind of compiled language, some of these types of mistakes you would catch before you'd ever run your program. You hit the button they're trying to compile your color like look, you got a mistake, you need to fix this. And Ruby. Ruby is not a language that you don't get to do that you kept. You catch the mistake. Your program crashes. So you don't, you don't get the run button, and your program tries to execute things and then it crashes. Or in this case it just completely ignores it. So this is a dangerous one, because if you write this incorrectly, you won't get an error message, it just won't ever happen. So when you're writing, that's one reason it's important to test a lot. You're running a program, you're running your test. It should be producing this. Why is it not doing that? Maybe have a look at your mistake. If you have health gift tags in there, it's possible. I've seen this. I've seen this happen every selection. This actually happens quite a bit with students. It's possible that you forgot your, it's just ignoring that tag and we can actually look at that and action list. Go back to our turn up here and so if. X = 1 it doesn't equal 1. Print one. And Elsa? I'm gonna write it. We write correctly first, right else if X = 2. What does equal 2? Else. And then we're done. So sure enough, it does equal to. That's great. Now let's rewrite this the wrong way. My dad. else run the wrong way and many.

Look. Let's start right here where I just typed in X, right? That is, that's an instruction. The instruction was just X. And what's it going to return? It's gonna return the value of X. This statement. That's an instruction. What does it return? Will it returns null? So Null is an object in Ruby by the way. It's not just data from some languages knowledge, just like zero knowledge, not zero and Ruby nulls and actual object that has its own methods and stuff. So keep that in mind. That will come up in another assignment, will have many students have issues with that thing as well. But this if statement returns something, right? Typically if you have a block of code like this if statement. What gets returned? Is the result. Of the last instruction that got executed. So last instruction was a print statement. So print statements were turned Nolan Ruchi. We could see if we let's rewrite our print statement. Else and will do. 25. Oh, that's not true. Right, so it actually returned. That's the last instruction I got executed. So that's just, that's the way Ruby works that notice nothing got printed, so anything that would have gotten printed that result of that. Construction happened on the left side here. The right side of his arrow is the thing that got turned. So again, I really like this interactive. Really, really nice. You could get the same thing from Eclipse or another fancy one, but in my opinion this is all you need. That's what it printed. This is what got returned. Everything is very clear, right? It's very clear what happened. It's very simple. There's not a lot of clutter to distract you. So anyhow, same thing with the case then, right? Whatever gets executed last. The result of that instruction is what gets returned, and you don't have to say that. I only put it here just so that you know you would be aware that that could happen, and you keep that in mind because everything in Ruby does, and that will affect. That will potentially affect your code later when you're working on the assignment, but just so you know that you're aware of that situation. It's not requirement, but in Ruby. I'm still having word switch. We used the word case. And just like the statement, you don't have to use. Parentheses are not required. If you do use them, it's not wrong. So for example if I go back here. If I wanted to if my if statement, I could put parentheses around here. It's not wrong to do that. It'll still behave normally. I don't have to do it. But it's not wrong to do it. Same thing for my case statement. If you wanna put parentheses around it, you can. It's not wrong to do it, but it's also not required. And then instead of the word case. So in Java you have the word case and Ruby you have the word then. This is the value you're trying to match it on, and then if you want to write a one liner you know it's just like a new state. Here's my one that then and then the rest of the instructions for you to hit the enter. Example. So this. Oh, before we do the example, let's do a comparison. So in the last class, students asked me to do this. So I think this is actually helpful. I think it's a good idea, so that's why I'm doing this here as well. So in in Java. We have something like this right variable 1. And then you have to do your curly brackets. And you have a case one. Do some stuff. Case to do some stuff. Case three, stop and then your. Default clause right here. So that's where those second Java. And in Java. These are not exclusive, so as soon as one of these things is true. I'll execute the instructions in that case and everyone below it until you get out of here. Until you do break statements, you break out of the the block. That's not true in Ruby, so Ruby they're exclusive. So Ruby, case wherever. One. I don't need the parentheses, don't need the curly brackets, and here I just say 11. Then we got that document. Again, you can use just I can statements; I can put a new line and put my new instructions here and I said you can write it just the same rules you apply for if statements apply here. And then Ruby. we just use the word L.

Well, you can import it. Load. Just type the file name or the path to it if it's in a different folder, wherever you like this one. So it actually executed that because I had that method, but now I want to. I can run that method as many times as I want or any other methods that are in here. So another nice thing that we can do this, why don't we? Why don't we make another method? Let's take this out of here. Let's make another method that has parameters. Times 2. And this one, we need the parentheses, so we'll just say parent 1/2. We'll just do two from now. i would have it