Welcome to module 2-- developing SuiteScripts. In this module, we're going to look at how we can start developing our scripts, starting with the traditional Hello World script. What we're going to cover here would **be** **how** you can set up your Eclipse IDE. And that includes how you can use the SuiteCloud IDE plugin to automate some of the actions that you will be doing in SuiteScript. We also want to learn about the SuiteScript 2.0 architecture, and we are going to create our first script and the traditional Hello World script.

Before we begin, a few things about the account that we'll be using. So here we are going to pretend that we are employees of the company called SuiteDreams, which is a global custom furniture manufacturing company. So what the company does is that it creates furniture, either a custom furniture that was requested by customers or just some furniture designs that the company has created. So here we will be automating mainly the customer and employee records, but we are going to also automate some of the transaction records such as, maybe, sales orders.

Now, regarding the NetSuite account that you are using, your account would automatically expire after 30 days. After this course, feel free to play around with the account, but just do remember that it would automatically expire.

Now let's start with our traditional Hello World script. What we're going to do here would be six parts. So first, we want to make sure that we prepared our IDE, which we will be using Eclipse. Again, you can use any IDE that you prefer, but do know that there are a lot of customizations that are built into the Eclipse IDE using the SuiteCloud IDE plugin that would help you develop SuiteScripts.

We're also going to look at how you can create a SuiteScript project, how to create a script, how to upload your script to the file cabinet automatically using the Eclipse IDE. But I am also going to show you how to do that manually, again, just in case you want to use your own IDE. We are going to create a script record-- so how can you create a script and also deploy that script? And, of course, how we can execute that script. So that's what we're going to do.

So this is an example of a SuiteScript 2.0 script. Again, SuiteScript is something that you would use if you want to automate or extend the capabilities of NetSuite. So SuiteScript is based on JavaScript. You need to be familiar with JavaScript if you are going to use SuiteScript.

If you're not too familiar with JavaScript, that's fine as long as you have experience in other object-oriented programming languages. Now, the syntax will be different for JavaScript, because each programming language has its own syntax. But JavaScript is really not hard to learn.

Now, in your student guide in Appendix A, you should see some syntax guides on how to use JavaScript. But if you go online and try to search for information on JavaScript, you would see a lot of free tutorials that would get you up to speed on JavaScript programming.

Now, there are certain parts of your SuiteScript 2.0 script. The first part is the required annotation. So these are the two things that you need to make sure you add to all of your SuiteScript 2.0 scripts, otherwise that would not be recognized by the system and would not be triggered automatically. Now, a script that does not have this annotation, the IDE would still allow you to upload that script. But if you do forget to add the script, that would be treated as a library or a SuiteScript 1.0 script.

Now, aside from the required annotation, you need to make sure you have your define statement. So this define statement is the statement that allows you to create your own module. So this is called an entry point module, because this would be triggered via the system at the particular entry point.

Now, in this case, you can see that one entry point that we have here is the beforeLoad entry point. And this would be triggered as soon as the user does something to the record-- either open the record, print the record, edit the record, do anything to the record. Now, we're going to talk more about entry points once we get to the entry point module.

Now, there are several entry points depending on the script. For example, a user event script would have three script types. A client side script would have, I believe, 11 script types. Some script types only have one entry point. So it really depends on the script. So this is just an example. Again, we're going to talk about entry points more once we get to that module.

And once you have that entry point function, then you have your statement. So this is an example of a logging statement. There are four different kinds of logs. So this is an example of a debug log. So we're just going to start simple.

Now we're going to talk about the details of how you can create the script. But just note that this is a sample of how you would create a simple SuiteScript 2.0 script. So in this walkthrough, what we're going to do is that we're going to build a Hello World script.

What we want to learn here is that we want to learn how we can set up Eclipse, which should already have the SuiteCloud IDE plugin. If you have not installed Eclipse or the SuiteCloud IDE plugin, please refer to the preclass setup guide so that you can install those two things. Also, we're going to learn how you can create a SuiteScript project, and how you can associate your NetSuite account so that you can automatically upload your files or your scripts to the file cabinet on the associated account.

We're also going to build a SuiteScript, upload the file to the file cabinet, and learn how we can create a script record and also a script deployment. And of course, lastly we want to execute our script so we know that it actually works. So let's go ahead and go to our walkthrough.

OK. So the first thing that we are going to do here is that we're going to open the Eclipse IDE. Now, you know that you have properly installed the SuiteCloud IDE plugin if you see this start page that says, "Welcome to the SuiteCloud IDE." You should already be in the NetSuite perspective, and you should also have the NetSuite Navigator here open.

Now, once you have the Eclipse IDE open, the next thing that you're going to do is that you're going to nominate a master password. The master password is important, because you can potentially have multiple projects that would link to multiple accounts. So if you are, say, a partner, or if you are developing projects for your own NetSuite customers, then it's important for you to be able to manage all of those different projects and have them linked to different accounts.

Now, if you're only going to develop one account, then that's still fine. You still need to also have that extra layer of security by setting the master password. You can do that by going to the NetSuite menu on the menu bar in Eclipse, then go to Master Password, and then use this to nominate your master password.

Now for me, I've already nominated the master password. So all I need to do now is just type my master password in. After typing in my master password-- so I've set up the security for my IDE-- the next thing that I want to do is that I want to make sure that the environment for this account is properly set.

For this training, what we need to do is that we need to make sure that we add the data center for our account. Now, all training accounts are located in this particular data center. So the URL of NA Northwest data center is system.na2.netsuite.com. Now, if you want to know the data center where your account is at, all you need to do is just go to your NetSuite account, then on your NetSuite account login, go to the URL on the URL bar.

So here you can see I have a system.na2.netsuite.com, letting me know that, OK, I'm at the na2 or on the NA Northwest data center. So if you want to set that up in Eclipse, you have to go to the NetSuite menu, go to Manage Environments, and then on Manage Environments click on New, and then just go ahead and add the data center that you're going to use. For training, we're going to use NA Northwest.

But it doesn't necessarily mean that all the training accounts would be at NA Northwest. It could be on one of the different data centers. Just make sure that you're connecting to the right one. If you're not connecting to the right data center, then you will get an error message once you start uploading your accounts to the server, OK? So again, that's something that you need to make sure you add if you don't already have that on your account.

So that's it. So those two configurations are things that you would set up as a configuration for your IDE. So that's a one-time setup. And once you've set that up, you don't really need to make some changes.

Now, once you have your IDE prepared, the next thing that you want to do is you want to create your project. Now, for creating your project, there are several ways of doing that. First, you can right-click on your NetSuite Navigator here, go to New, and then click on NetSuite Project. You can also accomplish the same thing by going to File, go to New, and then click on NetSuite Project.

Alternatively, you can also click on this NetSuite button on your toolbar. So if you click on the down arrow, just click on NetSuite Project, and then just go ahead and start building your project.

Now, for this project I'm going to call this "SuiteScript Course." OK. And that's the only thing that you really need to provide here. Just make sure that you use a name that would reflect the project that you're building. Also, for the SuiteScript version make sure that you are selecting the right version. If you are going to build a new project, I do recommend that you build that using SuiteScript 2.0.

If you are going to maintain or augment an older project running on an older API, then make sure you also select that. Say I'm going to just add a few scripts for an old SuiteScript 1.0 script. So I'd need to also select that, OK?

Now, there's also an option here at the bottom that would ask me, OK, what kind of project am I building? Am I building a SuiteScript project, or am I building a SuiteCommerce project? So if I have an SSP application project, I can also select that and use the same IDE.

So lastly here, I'll just click on Finish so that I can build or I can start creating my project. So here, I've already set up my IDE and I've already created my project. Now, the next thing that I want to do is that I want to link this particular project to my NetSuite account.

So to do that, what I need to do is that I need to right-click on this project, and then go to NetSuite, and then here just click on Change Project Settings. So again, that will be right-clicking the project, go to the NetSuite option on the menu, and then click on Change Project Settings.

So here it's going to ask me, OK, what account do you want to use to link this particular project? Now, since I don't have that account yet, what I need to do is that I need to add that account to my IDE. Now, if you've already added another account, for example if you have created multiple projects and you've already imported that project or this account to the project, then you can just go ahead and select that same account. But if you haven't linked the IDE with any account, you can add that by clicking on these three dots or the ellipsis button.

So if you click on that, it'll ask you, OK, what account do you want to use? For us what's important is this account. So we can click on Add here. And then from here, we're just going to type in our email address and password. So let me just type in my email address and password.

And also, one thing that you need to make sure that you remember is that remember the environment that you've added? So here, we've added NA Northwest as our environment. So just make sure you add that or select the right environment before clicking on Next. Again, using the wrong environment for your configuration will give you an error message once you start uploading your scripts to the server.

So here, I'll just click on Next. Then after clicking on Next, the IDE will fetch the account information from the server and would give you all of the accounts that are associated with this email address and password combination. Now, if you have multiple accounts listed or associated with this email address, then you would see all of the accounts listed here.

If you are going to start developing your project, make sure that you select only the account that you will be using. Don't select all of the accounts, but just the one account that you are going to use. So in this case, that's not a problem for me because in this email address and password combination, I only have one account associated.

So I'll just select that, click on Finish. And then, as you can see, it would be listed inside the Accounts section of my Accounts list. I'll just select that project, click on Select and Close. And then after clicking on Select and Close, it'll give me the information here, such as the account name, the role that I'm going to use.

Typically for the role, you would be using the administrator role to develop your project. SuiteScript version would be whatever version that you used when you created your account. And then the file cabinet folder would be the folder that the system would be using when you upload the file to the file cabinet.

Now, you can change the name of the file cabinet folder. For example, if you want to change that from, say, SuiteScript Course to another name, that's something that you can do. It's still going to be associated with this particular project. But that's not something that we recommend, because if you are going to manage several projects, it would be hard for you to determine which project actually relates to which folder in the file cabinet.

By keeping the names the same, it will be easier for you to find the files or the folders that you're looking for. So here, I can just go to the file cabinet and look for SuiteScript Course. I don't have to look for another name.

So after that, let me click on the OK button. And we've already configured our project. So, again, these are all one-time configurations. Once you've set up your project, those are things that are not going to change.

Next thing that we're going to build now would be our script. So here to build our script, similar to building our project, we can right-click on the project on the NetSuite Navigator. You can go to New and then click on SuiteScript File. You can also do the same thing here in File, so File, New SuiteScript File. Or you can also click on this NetSuite toolbar button, and then here you can select a SuiteScript file, OK? So either of those three options would be pretty much the same.

So on the SuiteScript File, it's going to ask what kind of script do you want to build. Now, for us what I'm going to do here is that I'm going to select a blank script. Typically, you would not be using a blank script. You would be using the templates that are available for us. For example, if you're building a client side script, then you're going to use Client Script. Or if you're going to build like a schedule script, you can choose Schedule Script.

But for this particular walkthrough, we want to build a script from the ground up so that we can add in all the elements that we need for this script. In future walkthroughs, we are going to use the templates to make our development faster.

So moving on here, next thing that I need to select is the script filename. Now, we do have a file naming convention for building scripts. The first part of your file names should reflect the company that you are building the script for. For example, in our case we are pretending to be employees or developers of SuiteDreams, which is a global furniture company.

Now here, to represent SuiteDreams what I'm going to do is that I'll just type in "sdr." So that is SuiteDreams. Add in like a two- or three-letter abbreviation for your company. For example, if I'm developing this for NetSuite, I would probably use "ns" or a project number that we would be using here at NetSuite, OK? So it really depends on your company.

And you want this so that you would prevent collision, just in case you install scripts from other companies-- like if you purchased an app from another company, or if you're a partner, if you're developing scripts or projects for your own NetSuite customers-- you want to make sure that you don't want to cause conflicts. So next thing here after the company abbreviation, you add in an underscore, and then put in the kind of script or an identifier on the kind of script that you're building.

For us, we are going to build a user event script. Now, don't worry about the different types of scripts for now, because we're going to talk about that in a future module. But just do remember that if you are going to create a user event script, use "ue" as your identifier-- "ue" for user event.

And then after that, just add in an underscore, then add in a description of what you're trying to build. So this is part of a project, you might add like a component number if that's something that you do, or a module number. Again, it really depends on your development environment. If this is a smaller project, maybe you can just give that a more descriptive name. Now, for us in this course I'm going to use this user event script as a catch-all script for all user event scripts deployed to an employee record.

I'm going to give this a more generic name. So I'll just name this "employee." Now, we also have an option here to add different modules. So modules would be additional functionalities or dependencies that you can add to your script. Now, for this particular script we're not going to do anything fancy yet, because we just want to display a Hello World message on our account.

OK. Let me click on Finish. And then that's that. So I've already built my script. Now, to make sure that your script would be recognized by the server, there are a few things that you need to add to your script. Now, this define statement would be one of the things that needs to be there for the system to recognize this script. The define statement is used to define or create a module.

Now, in NetSuite, like what we've mentioned previously, everything would be created as modules. So these modules are either modules that would be automatically executed by the system, or they could be library modules that you can call to add functionalities to your script. So there are prebuilt or existing library modules that you can use, or you can create your own if that's something that you want to do. By creating your own modules, you can have a reusable set of functions that you can attach to any of the scripts that you're creating.

Now, for the script that we're creating, we are creating an entry point module. So entry point modules are scripts or modules that are automatically executed by the system if the trigger was met by a user or by the system. Now, again, at this particular point in time you don't have to worry about entry points, because we're going to talk about that in a future module. But just know that anything that would be executed by the system when a certain action happens, that would be considered as an entry point module, OK?

Now, whenever you are developing an entry point module, the first thing that you need to have on your script would be JavaScript doc tags or JSDoc tags. To create a JSDoc dot tag, what you need to do is add in a slash and then two asterisks. Press Enter here.

And remember that you have to use slash and two asterisks, OK? It's very important that you used two asterisks, because that would be recognized by the system as a JSDoc dot tag. If you only use one asterisk, that will be considered as a normal JavaScript comment, not a JSDoc dot tag.

Also, you would see here that if I use a slash and an asterisk, it would turn the color of the comment to green. But what you're looking for is that it would be color blue. So you know that you're actually using a JSDoc dot tag.

Now, the two annotations that you need for this JSDoc dot tag, so the first one would be the version of SuiteScript that you want to use. So to determine that, you can use the NAPIVersion. So NAPI here, you're going to define-- for example, in this case I want to use my SuiteScript 2.0 script type. If you don't put in the NAPI version, then this would be considered as a library file, or this might be considered as a SuiteScript 1.0 script. So if it is considered a SuiteScript 1.0 script, then this might not get executed at all, because you have a module definition, which is not really recognized by SuiteScript 1.0, OK?

So another annotation that you need to make sure that you add would be the script type. So here, that annotation would be NScriptType. Now, for the script type, that would be similar to kind of what we defined in the file name. So for the script type, this would be a user event script. So that would be the script type.

Again, at this particular point in time you don't really need to worry about the different event types or the different entry points, because we're going to talk about that in a future module. Just do remember that these two annotations are required. Just make sure that whenever you're creating an entry point module similar to what we're doing, you have to add these annotations.

Then after adding those annotations, the next thing that we're going to do here is that we're going to add a function. Now, we already have a function. Now, this function is part of your define statement. But we need to create a function inside this function call. So this would be your callback function for your define statement. And then inside that, we can add in a function right over here, or we can add in a function inside the return statement.

So if you want to add a function inside the return statement, you have to name that based on the event that you want the trigger for your script type, OK? So for this event, I'm going to use the afterSubmit event or afterSubmit entry point.

So the afterSubmit entry point, what that does is that it gets executed whenever the user saves the record. So for example, if I edited the record and then I click on the Save button, once it gets in the server it would be executed. So that would be what afterSubmit is for.

Again, at this particular point you don't have to worry about that. Just try to remember the structure of our script. And that's really the main point of this particular walkthrough. Just remember the structure and how we create our script. But the details of what the different elements of that script are, then we're going to talk about that in future modules, OK? So again, we're not going to skip over the details. We're just going to have that discussed in another module.

So anyway, this is kind of different if you are not familiar with this format for creating your functions. If you want to create an entry point function, you have to put in the name of your entry point in the return statement-- so here, the name of my entry point would be afterSubmit-- and then adding a colon, and then use the "function" keyword. And then after the "function" keyword, you have your parameter list.

For SuiteScript 2.0 entry point modules, you always have to have that function inside the return statement. You can either define the function directly inside the return statement, or you can create your function before the return statement and then just call that on your afterSubmit, OK?

So for example, here if I create my function, let's say I have a function here, I will call that myAfterSubmit. And then for myAfterSubmit function, I would have an object that would be passed on to the server. So this would be my script context or my context object. And then I can have some functions here. I'll just say do something. And on my return statement, I can just call that. So I can just say myAfterSubmit.

Again, you can define your function inside your return statement directly or define that before the return statement and then just call that function at your return statement. Now, what is the difference between the two? Regarding entry point functions, they are pretty much the same. You can define your function before the return statement, or you can define that in the return statement. It doesn't really matter, and it's really up to you.

But if you are going to create another function that's not really attached to an entry point, but maybe you want to create a reusable function, then you want to define that function before your return statement. Because anything that you define before your return statement can be called inside your return statement. So for example, if I have a function here that sets a certain value, here I'll create a function, and then I'll create a setFieldValue function, then maybe I have a parameter here, or maybe fieldName.

Then what I can do is that if I have a function here, function-- so I have context there-- I can actually call that setFieldValue, so setFieldValue, and then maybe I have a value here. So this would work. And if I have another function-- let's say I create beforeLoad function-- so same thing here, I can also call that setFieldValue.

So putting your functions before the return statement allows you to call that anywhere from your entry point functions. Typically for entry point functions, you don't really need to reuse that. It's just a one-time thing that you would call depending on the user's action. So putting that inside, or putting your entry point function inside your return statement or outside your return statement would not really matter.

Anyway, so for us, I'm just going to remove this. And since I already have my afterSubmit function here defined, I'll just reuse that. So I'll remove this because, we don't have that setFieldValue function anymore.

So anyway, so we've already defined our script. So we created our script, we added the required annotations, and we've also added our entry point function. The next thing that we're going to do here is that we're just going to add a log that would say "Hello World." That's it.

So here I'm going to use log.debug. Then for my debug log, I'll just say "Hello World." That's it. OK? That's it. That's your basic script.

So you need to make sure that you have your define statement. Inside your define statement, you have a function. And then your function should be returning an anonymous object that has the name of your function defined with the name of your entry point defined as a function. So here you have your function that would just log the words "Hello World." Again, more details about that once we start expanding our script.

Next thing that I want to do here is that I want to upload that to the file cabinet. There are several ways of uploading that to the file cabinet. You can right-click on the editor window, go to NetSuite, and then click on Upload File in Editor. You can also press Control-U if you are using Windows. If you're using a Mac, I believe it's either Option-U or Command-U, OK? But you can just go look at the context menu and try to look for this option. It should also list the keyboard shortcut.

You can also right-click on the file here. So you can upload multiple files. Just select all of the files that you want to upload. Now, in this case I only have one file that I want to upload. So I can right-click on that and then go to NetSuite, and then just choose Upload Selected Files.

So in this case, I only have one file. So I'll just choose that. And as you can see here, it's already uploading my file. So we can just wait for a few seconds for this to upload. And once that completely uploads to the server, then we should be able to see that on our account.

So let me go to my NetSuite account here. So on my NetSuite account, if I hover over Documents and then just go to Files and then SuiteScript-- again, that's Documents, Files, and then SuiteScript. If I open that, so I can see here that I have a folder named SuiteScript Course. So the system or the IDE automatically created a SuiteScript Course folder.

And if I open that SuiteScript Course folder, you can see that I have my file here, sdr\_ue\_employee.js. Now, another thing that you can do, for example, if you want to manually upload your scripts to the file cabinet-- and you typically wouldn't be manually uploading your scripts to the file cabinet if you're using Eclipse with the SuiteCloud IDE plugin. But if you want to use your own IDE-- for example, I'm already used to using, say, Sublime for all of my JavaScript files, or Atom, or maybe Visual Studio, or whatever JavaScript IDE you want to use, and you want to keep using that because you're accustomed to that, and you don't want to have multiple IDEs in your system.

Then what you can do is that you can just click on Add File. You can create your folder, click on Add File, and then just choose the file that you want to add to the system. Now, for us we don't really need that. I just close the window. But that's something that you can do if you want to upload manually.

Also, if you are going to use another IDE aside from the Eclipse IDE, then what you can do is that you can download the API definitions. So on the upper right-hand corner of the page, you would see here that I have a SuiteScript 2.0 API and SuiteScript 1.0 API. So this would be the API definitions that would help you with content assist or autocomplete. So you can link this to your IDE, and this would give you all of the different functions that are available on either SuiteScript 2.0 or SuiteScript 1.0.

So this would just be for definitions. So it'll give you like Help Center information kind of things. But it would not give you the actual implementation of the API. So that's part of the things that are located on the server, and that's not something that we can view. So just the definition.

So anyway, so that's it. So we've created our project, set up our IDE, uploaded our file to the file cabinet. The next thing that we need to do is that we need to tell the system that we have a file that we want to execute. Because uploading a file to the file cabinet would not automatically execute the file, it'll just tell the system that, OK, there's a file in the file cabinet but I'm not going to touch that file.

Because for the system it's just a regular file. It doesn't really matter if it's a JavaScript file, a Word document, a text file, a picture, or whatever. It's just a file. So we need to tell the system that, OK, I need to execute this.

You can do this by going to Customization. And then under Customization, go to Scripting, go to Scripts, and then New. OK. So again, go to Customization, Scripting, Scripts, and then New. By clicking on New, it will give us a new page, and that page would have us create what we call a script record.

So a script record would just be a configuration file that would tell the system how we're going to execute this particular script. So here, we're going to select our script file. So as you can see, we have a lot of scripts here, but we're just interested in the file that we've created-- sdr\_ue\_employee. Click on Create Script Record, and it'll give us the script record page.

So in this script record page, as you can see, we have a lot of fields that we need to make sure we specify. So for the name field, so it really depends on your project. Again, you have to talk to your development team on your naming convention for your script record.

Now, for this course I'm going to use this kind of naming convention. I'm going to start with the name of my company, which is SuiteDreams-- kind of similar to what we did in our file name-- and then our script type, which is UE, and then lastly, just a description of what I'm trying to create. Again, so this is a catch-all for all employee customizations. I'll just call this Employee, OK?

Now, another thing that we have here would be the ID. Now, this field is very, very important for us developers, because if you are going to build a script, you would almost always refer to the ID of a customization instead of the name. So this ID would be static. Once you've created the IDE, that's about it.

You can change it if you want to, but I don't really recommend changing the IDs once it's set, because it might have been used in another customization, and that other customization might break if you change the ID. And that's why you need to make sure that you set the ID properly when you start building your customizations. It's not just for the script but any kind of customization that we have in NetSuite.

If you're an administrator, or if you have access to any kind of customization, make sure that you inform users to use the proper ID naming convention. So that convention would be start with an underscore. So the reason why we're starting with an underscore because the system would automatically give us a prefix of what we're trying to build.

Now, in this case we're trying to build a script record, so it will give us the prefix custscript. Then after that, the naming convention is similar to your file naming convention, so sdr underscore ue and then underscore employee. So again, it is very important for you to use the proper ID naming convention.

And not just for you. You have to make sure that you also educate the rest of the company who at least has access to any kind of customization if they're creating a custom search, or if they're creating a saved search, or a custom form or custom field, maybe-- so anything that you customize in NetSuite, if they have access to that, we do recommend that you educate them. Because even if you use a proper IDs but the rest of the company doesn't, it's still going to be hard for you.

Because just imagine having n fields, and then the field names would be 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. And those are like different kinds of data. Like, one's the date, and then one's a total for something, then another one is an ID for something, and then another value would be, maybe, a comment. And it would be hard-- like, so what is 1? What is 2? What is 3? And you really wouldn't know.

So if you properly named the IDs, it'll help you understand what the value is just by looking at the ID. So again, you can just think of the IDs as a variable names that you would be using to access certain values on the system. So again, very, very important, and you would quickly realize how important this is once you start building your own scripts. And then you would see in your system if people are not properly naming IDs, OK?

Anyway, so that's the ID. Next thing that we have here would be the description. Similar to your ID, this is also important. This would help you get more information about a particular customization. Now, in this script, let's say this is a user event script for all employee customizations. OK. That's it.

Doesn't really matter if it's a long description or a short description. You want to make sure that you give enough information for whoever would be, say, picking up this project, just in case you're passing this project to somebody else. Or if an administrator wants to know what the script is about, then this description would be helpful for them. Or even for you. Like, if you create this customization, and then you move onto a different project, if you want to review all the projects that you've created, then this would help you.

Now, in this course I would be skipping the description field just for the sake of time. But once you start building your own script, try to avoid skipping the description field. So anyway, moving on here, going on to the Deployment subtab, so next thing that we have to set is that we need to build a system, what kind of record would trigger our particular script.

Like what I've mentioned earlier, the afterSubmit script would be triggered when a user saves a record. So we need to tell the system that, OK, not all records, just this particular record, would trigger our script. Now, by going to the Deployment subtab, this would actually create a different record called the deployment record.

So you have two records that you need if you want to execute your script. You have your script record-- so what we're building right now-- and then you also have your deployment record. So the deployment record, even if it's a separate record, you can create that record directly from your script record. So that makes it really convenient.

So for the first column here, so the first field that we want to set would be, OK, which record would trigger this? And this would be set to Employee. If you want that to be triggered on another record type, of course, you can just use another record type. You can have as many deployments as you want.

For example, if you want to reuse your script-- say I want to deploy this to a customer record, an employee record, and a partner record-- I can do that. So I can just add three deployment records on my script.

Anyway, so for us we want this to apply to an employee record. The next thing that I have here would be the ID. So for the ID, I'll just copy the ID from my script record. And that's kind of common, especially if you only have one deployment for your script record. Now, it's OK to have the same ID for the script record and the deployment record, because you will be given a different prefix.

So for your script record, the prefix would be custscript. And then for your deployment record, the prefix would be custdeploy. Technically, it's still not going to be exactly the same ID, because you have those two different prefixes.

You also have an option here for deploying or undeploying your script. And this is very convenient if you want to temporarily or even permanently disable a certain script. Instead of deleting that from the system, you can just go ahead and disable it.

Now, I don't recommend that you would delete files from the system, because you can still reuse those files in other projects. Even if this is not executing, maybe there are some pieces of codes here on the script that you can just copy and then put that on one of your other scripts. So instead of deleting that, you can just uncheck this, and that would not be executed by the system.

Now, also you have a status. So here for the status, you have two different status options. You have Testing, and you also have Released. Now, what is the difference between Testing and Released?

If you set your script to Testing-- and this is what you would use if you're still developing your script-- what will happen is that on your script-- when you set your status to Testing, what will happen is that only the owner of the script can execute this particular script type.

Now, in this case the owner is Larry Nelson. So I'm logged in as Larry Nelson. So that means that I am the owner of the script, and only I can trigger the script. If somebody else saves the employee record, then this script would not get triggered.

Now, after you are done with development-- let's say I'm really confident that the script is OK-- I can set this to Released. And once I set the status to Released, that means that everybody in the account who saves the employee record will trigger my script. So for now, I just want this to be set to Testing. And then once I'm ready, I can just change that to Released.

Now, other options that I have here, I have the event type. So these are the different events if I want to filter the execution of my script. For example, I only want to trigger the script when the user is creating an employee record. If they're editing an existing record, then don't trigger the script. Or maybe if I want to do the opposite. Like, I want to trigger the script only when I'm editing an existing employee record. So these are filters that you can add.

Now, typically these filters are added from the script. So instead of putting that on the deployment record, you can add in an if statement on your script that would either trigger the script when the user's creating a record or editing the record, or whatever option that you choose in your event type. So this is actually something that we're going to use in a future module. Just do know that you can set the event either from the script deployment or from your script.

And then lastly here, we have the log level. So we have four different log levels. We have Debug, Audit, Error, and Emergency. For now we're just going to set this to Debug. But we are going to talk about the different log levels in a future module.

So here, just click on Add and then click on Save. And then that is about it. So we're done. So we've set up our IDE, created our script, uploaded our script to the file cabinet, created a script deployment, and then we also created our script record. Now, the next thing that we're going to do is that we're going to test.

Remember, afterSubmit, like what I've been mentioning, is triggered whenever the user saves the record. Now, in this case we have to specifically save the employee record, because that's where we deployed our script. So we're going to do that. I'm going to go to List, Employees, and then Employees. I'll open this in a new subtab. Again, that's List, Employees, Employees.

And then here, I'm going to open-- it doesn't really matter which record we open. I'll just open Aubrey over here. And then I'll click on Save so that we can trigger our afterSubmit. Go ahead and click on Save. OK. And then it looks like our record was successfully saved. So I'm going to go back to my script deployment.

So every log that you create on the system would be stored inside your script record, specifically inside the Execution Log subtab. So if you want to know what happened to the script execution, either if you got an error message or if is successfully did what it had to do, you can go to the execution log. So if I open my execution log, as you can see here, we were able to properly log that information. So here we logged "Hello World," and that "Hello World" was logged by me.

OK. So that's how you would build your project. Again, at this particular point do remember the structure of your project and not just really the details, because we are going to talk about the details in a future module, OK? That is about it for this walkthrough.

Quick review again on our syntax. So you need to make sure you add in your annotation. Very important-- you have to put in a slash and two asterisks, not one. So two asterisks would be your JSDoc tag. You have your define statement here. So the syntax of define would be provided by your IDE. You don't have to worry about it. But if you want to know the syntax, basically that's the define statement, and then inside your define statement is a callback function. And then your callback function needs to return an object that has your entry point function. So in this case, our entry point function is beforeLoad.

So as you can see here, the syntax for our beforeLoad entry function would be beforeLoad colon and then function, and you don't have a name specified here, and then context. And then after that you have your statement. So in this case, it's just a simple log statement, log.debug. And that would log in the phrase "Hello World."

That is pretty much it. So again, with this syntax, like what I've mentioned, you can create your entry point function inside your return statement. So just remember to use the syntax. Or if you want to create your entry point function before your return statement, that's something that you can also do-- so beforeLoad here, which is the standard JavaScript function syntax, then you can call that inside your return statement. So beforeLoad would be the name of the entry point, and then beforeLoad here on the right side, that would be the name of the function I've created. So if I named this something else, like myBeforeLoad, this would look like beforeLoad, colon, and then myBeforeLoad.

So a few things to remember before we go to our exercise. JavaScript syntax is case-sensitive. So you have to be careful of that. If you use any variable names, you have to use it as it was used in the script. If you change the case, then that will be treated by the system as another variable.

Which is kind of weird with JavaScript, because unlike other compiled languages like Java or C#, it'll tell you if you have misspelled something. But in JavaScript, it'll just assume that you're using a different variable. So be careful.

Also, with the naming convention, make sure you use lowercase. So that is a common naming convention. Just use everything as lowercase. And the syntax would be the company abbreviation, an underscore, then the script type-- for example, a user event script would be ue-- another underscore, then the description dot js.

Now, the ID naming convention is very similar, except you're going to start with an underscore, because you will be given a prefix for that customization. So if you're creating a search, that will be custsearch, or if you're creating a script that will be custscript. And then syntax, again, is the same-- company abbreviation, underscore script type, underscore, description.

Now it's your turn. Please complete the following activity. This includes module 2-- developing SuiteScripts.