**Welcome** to module 5-- SuiteScript modules. In this module, we're going to look at expanding our script further by using built-in SuiteScript API modules. SuiteScript API modules would allow us to extend the capabilities of our module by adding functionalities to it.

So what we're going to cover here is that we're going to start with a few SuiteScript API modules. So we're going to attach that to our entry point module so that we can, again, extend the capability of our module; going to add a few dependencies. So also, we're going to look at specifically the record module so that we can load or create records. And very important here, especially if you have a larger team, if you have a separate NetSuite administrator from SuiteScript developers, there can potentially be conflicts there. So we're going to look at where that would happen.

And then also, we're going to look at new functionality in SuiteScript 2.0 called promises. Now, promises is not actually new in JavaScript. That has been out for a while. But this is kind of new in SuiteScript. And we're going to look at how that applies to our development.

So there you go. So what are modules? So modules are basically libraries that you can add to your script or to your module to extend the capabilities. Now, you can think of modules as libraries. If you are a Java developer or a C# developer, you can think of that as a package that you import to your script as a dependency so that you can add more functionalities to your application. So it's basically that.

Now, what modules allow us to do is that, instead of creating a very long script, we can actually break our script down into smaller modules and then reuse those modules in different scripts. In SuiteScript 1.0, if you have been developing SuiteScript 1.0, what happens is that all of the functions are actually thrown into the global scope, which means that you have access to everything once you open the script. But this is not really efficient, because it's unnecessarily using resources for functions or capabilities that you're not really going to use.

For example, on a script if you're not going to search, then why would you have like the search functionalities using up memory? So that's something that you don't want to happen. But by using modules, you can make your script slimmer, because you would only put in things that you're actually going to need. If you're not going to need that, then you can just park that away and then get that when you are about to use it.

So that's it. So again, you can think of SuiteScript modules as libraries or dependencies to add functionalities to your script, OK? Now, how would you add a dependency to your script? Now, this is the syntax that we have been using so far. So we have the define statement to create our entry point module. Then we have our callback function. And then our callback function would be returning our statement.

So here in this example, we have a beforeSubmit function, which is just getting a new record. Now, if I want to load a record, I need to add in the record module. So I can do that by adding another parameter on my define statement. So here in the script that we have been creating, what we're doing is that we're not really adding anything in the parameter. We only have one parameter, which is our callback function.

But in your define statement where you have a dependency, you have two parameters. So the first parameter would be the modules that you want to include on your script. And then the second parameter would be your callback function. Now. two things that you have to remember here if you are going to define a SuiteScript API module, or any kind of module that you want to attach to your script.

The first thing is that the first parameter is actually an array. So you have to use the square bracket to make sure that you define an array. And this would need for you to put in the name of the module that you want to add.

So for example, if I want to add, say, an email module, I'm going to put an N/email, or in this case, since I'm adding the record module I'm going to put N/record. Now, where would you find a list of the different modules and the capabilities of those modules? If you go to the Help Center, there's a section there on modules where all of the different modules that are available will be listed.

Now, since this is an array, that also means that you can add in as many modules as you want. So for example, if you need like five different modules, you can just add that in the array. So you can add a comma, and then the name of the next module, and then another comma, and then the name of the next module, and so on. So you can add, again, as many modules as you want.

Now, once you've added the definition of your module or the name of the dependency, the next thing is that you need to have a place to store that dependency. Now, that would go inside the parameter of your callback function. So if you look at how we were doing it before, you have your callback function here, but it is empty so there's nothing there.

But now you have a parameter here, and that would represent the dependency that you have here. So here, since I'm loading the record module, I'm naming my variable as record.

Now typically, the naming convention for the SuiteScript API is that whatever the name of the module, that's going to be the name that you're going to use as a variable. But again, similar to your context, it's up to you in which name you want to use. So in the context, you can use like scriptContext, you can use context, you can use ctx.

But here it's also the same. For example, if you don't want to use "record" because you're using the word "record" in your script, maybe you want to change that to recordModule or maybe recMod, or something like that. So it's really up to you which name you want to use.

But again, common naming convention would be the name of the function without the "N/." Because if you see "N/," that means that's a NetSuite or a SuiteScript API function. Because you can create your own module. So that's just an indicator that the "N/" means something that NetSuite has created. So again, that's how you create a module.

Let's look at one of the most common modules that we will be using in the SuiteScript API module list. And that will be the N/record or the record module. Here you can see some of the functionalities that you can add using the record module.

So you can use the record module to load a record. Instead of just referring to whatever the user is currently manipulating, you can just open any kind of record. It can also be the same type or a different record type. You can also use the record module to create another record. And you can use the record module to transform a record.

And this is a very convenient function to use, especially if you have a record that you want to convert into another record. And instead of copying values from one record to another, you can just use transform to automatically pick those values up and then move that to the new record.

Now, if you are going to use the record module, there are some parameters that you need to set. And that would depend on the kind of function or method that you're using. If you go through the record.load method, for example, you can see here that you have two parameters that you are going to send. Now, regarding the parameters in SuiteScript 2.0, it's not like the parameter list that you would commonly see in normal JavaScript or in SuiteScript 1.0, where you have a list of different values separated by a comma.

What you will do, if you are going to pass a value to a function in SuiteScript 2.0, is that you're going to use a payload object. A payload object is simply an anonymous object with properties and values that relate to the parameters that you want to set. Again, looking at the record.load method, you can see I have an anonymous object here with two properties. One property would be the type property, and the other property would be the ID-- type property referring to the record that you want to load-- in this case, I'm trying to load employee record-- and then the ID would be the record that you want to load, which is ID 15.

Also something to take note of is that, if you are going to use that type property-- and this is common throughout all of the different API modules in NetSuite. So type would typically be set as an enum value. So in this case, you can see that the enum value is record.Type with a capital T-- you know that that's an enum value-- and then dot employee, so employee would be the actual value that you are going to set it to.

Now, if you are an experienced JavaScript developer, you know that there's really no such thing as an enum value in JavaScript. But we're kind of pretending that this is an enum value so that we don't have to remember all of the different values. All we need to do is just get that enum value and then set our type value using that enum value. But in reality, actually, it's just a string value.

So you can still use the string if you want to. For example, employee dot Type with a capital T dot employee is simply just the employee string. We don't recommend that you use that, though, because that can potentially change in the future. So it can work now, but moving forward if it changes, then your script would break. And that's something that we want to avoid. By using that enum value, you're guaranteed that even if the value in the background changes, it's still going to use that specific enum [? list. ?]

So that would be our module. Again, if you want to create the record, you can see here record.create, where you have an option to say, OK, this is the kind of record that I'm creating. You also have the isDynamic value that you can set. If you want to know more about the different properties or parameters that you can use when you're passing a value on a module, you can go to that module definition.

Before we go to our walkthrough, there's something important that we need to discuss. And that would be the business logic layer. Now, let's take a look at this phone call record. And in this phone call record, we can see that there are a lot of values that are mandatory. And you know that a value is mandatory in NetSuite by looking at the yellow asterisk or the gold star.

So we can see here that we have the subject, status, organizer, and also the date, those are the fields that are mandatory on our phone call form. Now, aside from fields being mandatory we can see that there are also some values that are automatically set. Status, organizer, and date are values that the system has automatically populated. If you look at the reminder type, we also have some values there, which is none.

Now, why is this important for us? Whenever you input anything on NetSuite, remember that you have that business logic layer that normalizes the behavior regardless of the input channel. So if you're creating a record or editing a record or doing something to the record, the behavior would be the same regardless if you are inputting data from the user interface, pushing data from a third-party application using SuiteTalk, creating a record or editing a record using SuiteScript or SuiteFlow, or pushing data in using CSV import.

So this means that if there is any behavior that we see on the system, you can be guaranteed that that behavior would be the same regardless of the input channel. For example, if there are any fields that are mandatory on the form, then you can be guaranteed that those are also mandatory on your SuiteScript. So that's something that you can use as a guide, because as we know, the user interface is very visual, and we can already see, OK, these fields are actually mandatory. So when I use the createRecord function or createRecord method in NetSuite, I know that these fields would also be mandatory.

And that also means that if there's anything that is defaulted on the form-- so for example, if our phone call field, such as the date or the organizer are pre-populated by the system-- then if I create a phone call record on my script, those fields would also be pre-populated. So even if they're mandatory, I don't need to set them, because the system has set those values.

Of course, I can change the values if I want to. But I don't need to do that on my script, especially if I don't really need those values. And again, it's not just SuiteScript, but it would be the same behavior regardless of the input channel.

In this walkthrough, what we're going to do is that we want to create a phone call record whenever the user saves a new employee record. So what we want to do is that we want to make sure that the employee calls HR so that they would know about the benefits. So that's what we want to automate.

Also, we want to make sure that the phone call is assigned to the employee. So the employee would be the organizer, because the default behavior is that whenever you create a phone call record, it would be assigned to the currently logged in user. So if I run this script, the phone call record would be assigned to me. And I don't want that. I want it to be assigned to that new employee that I'm creating.

What we want to learn here is that we want to learn how we can load SuiteScript modules. We also want to learn how we can create a record for user event scripts. We also want to learn how we can define the relationship between custom forms and scripts. And this is something that's important and also related with our business logic layer. We're going to see how this can affect the execution of our script, and how we can force the script to use a specific form on our record.

So let's go ahead and build our script. So first off, we don't need our client side script anymore. Let's open our user event script. And remember, our script would only get triggered when the user is creating a new record. So for this one, we need to add in a filter. And we can do that by adding an if statement.

And if you remember the properties for your afterSubmit function, you have a property on your context object called type. And this type would tell you, OK, what is the user currently doing through this record? Is the user creating a new record, editing an existing record, viewing the record, approving a transaction, or anything like that?

Now, for this value, this is stored in an enum value. And that would be under context and user event type. So that's the name of our enum value. And then here, the enum value that we want to use is create.

If you remember on our script record and also in our script deployment record, if you go to Deployment subtab, you have a type property there. You can also set that type to create so that this would only be executed when the user is creating a record.

Problem with that, though, is that this script would be limited to when you're creating a record. But we have sections of the script that we want to execute every time. So our logging, we want to execute that every time. But for this section, we only want this to execute on create. So instead of setting that on the deployment record, it would be best to do that on the script, which is what we're doing.

Next thing that we want to do is that we want to create a phone call record. So to create a phone call record, of course, we need to load the record module. Now, to add any module to your script, what you can do is that you can go to your definition. So for this definition, as you can see, I don't have any modules loaded. And if I don't have any module loaded, I can also delete that array.

So for example, if I remove that and then type it like this, this would still work. So that's another format that you can use. Or you can just leave array definition here blank.

But if I want to add in any module, I'll just put in the name of the module inside this array. So that will be a string value. So for example, if I want to load the record module, I can just say N/record. If I want to load another module-- let's say I want to load my email module-- I can just add that in-- N/email.

To get the list of the different modules that you can add to your script or your entry point module, you can go to the Help Center. And then inside the Help Center you have an article here called the "SuiteScript 2.0 Modules." If you expand that, you would see a list of all of the different modules that you can load.

So for example, you have your configuration module here. An error module, so if you want to create your own error. You have your log module which we have used, runtime module, you also have your record module.

So if I expand that record module, I can see here that I have several functions or methods that I can use, including creating a new record. So if I open the definition for my record.create, I can see here that I have three properties. The only required property is my type property. So I'm going to use that.

Then for my type property, this would be using an enum, so record.Type. So we're going to set that record type later. But anyway, so going back here, so I have my record module.

And with my record module, once I define the name of that dependency, the next thing I need to do is I need to store that into a separate variable. And then we're going to access the functions that is attached to that variable.

Now, I can do that by adding the variable name inside my function definition. So for example here, I'm going to use the name record as my variable name. Typically, the name that you would be using would be the same as the name as your API module. So here, since I'm loading the record module, then the name that I'm going to use is record. But you can use any name that you want.

For example, if I use the name recordModule, so that would work. Or if I want it shorter, maybe I can use recMod. So that also works. So it really is up to you and how you want to do that. But, again, normal convention is just to use the name of the module.

Now, another thing that you can do whenever you are defining a module is that you can add in JSDoc tags. JSDoc tags would be helpful if you want to get autocomplete on your IDE. So for example, here I'm going to add some JSDoc. The annotation that I need is the @param annotation.

So for this @param, I'm going to say, OK, I'm going to load my record module. So this first parameter inside a brace or a curly bracket would be the function that I'm loading here. So here I'm loading record. And then the next parameter that I need to use here would be the name I used in my function definition. So here in my function I used record, and that's why I'm using record here.

Now, by adding this annotation I will get autocomplete on my function. So that's something that you want to do. So here, let me go ahead and create my phone call record of our phone call. And then I can use my record module. And if I press dot then Control-Space, as you can see here I have my create property or my create method. For my create method, I have these different options.

So here, create. And then I'll add in my payload object. And then for this one, as you remember like what we've seen in the Help Center, property that we need to set is the type. So since we're creating a phone call record-- let me just get that from my record-- I have my Type enum, Type with a capital T, then phone call-- just set it like that.

So that's it. So that's how you would create a phone call record. Now, for this particular automation what we want is that we want to set the phone call as a reminder to our new employee to call HR so that they can discover what their benefits are. And also, we want this to be assigned to the new employee.

Because what's going to happen if I go to my phone call record is that whoever is currently logged into the account will be the owner of my phone call record. So in this case, I am logged in so I will be the organizer. So we want to change that.

So we also want to set all of the mandatory fields. So status is already set. We can ignore that. Date is mandatory, but it's already set. But here, title is not set. So I need to make sure I set that title or the subject.

We also want to set the organizers. If I open the organizer, the field ID or the script ID is assigned. So now that we know those two script IDs, let's go ahead and set the value. So that's simply going to be my setValue method. I'll set the title first.

Then for the title, I'm going to say "Call HR for benefits." And then I'll copy that and then change this one to my assigned.

And for this one, since this is a list or record field, I need to make sure I set this to an ID. I want to get the ID of my employee record. So I'm going to say employee.id. So if you want to get the internal ID of any record-- so in this case we referring to an employee record, but if you want to refer to like a customer record, if the user has access to that, then you can just say customer.id. So whatever that record is dot id.

That's it. And then lastly here, what we're going to do is that we want to persist this through the database. So here I'm going to use a phoneCall.save. Now, this is something that is very important if you are dealing with creating a record. Because if you create a record using record.create, this would only create the record object. It's not going to save that to the database.

If you forget to save, or if you forget to call that function, phoneCall.save, what's going to happen is that it's just going to create the object and then discard that object once the execution completes. So you need to make sure that you always use that in pairs-- create the record, then after that save the record.

So let's see if this works. I'm going to upload this and go to my employee record. Remember, this would only be executed if you are creating a new record. So let's create a new one.

For my employee, I'll just call this Test 2 User. There you go. And then for my subsidiary, I can just do whatever-- just use whatever subsidiary. Doesn't really matter. And yeah, that's about it. That's the only mandatory fields we have.

So let me click on Save. And let's see if this works. OK, it looks like we were able to properly save the record. But did we create that phone call record?

We can check by going through the list of phone call records. So here I'll click on List. And then on my filter, I want to make sure I display all of my phone call records, not just mine. And then as you can see here, up top we have a new record, "Call HR for benefits." So that's what we created.

And then as you can see, it is assigned to Test 2 User. And then you have the title, "Call HR for benefits." So as we can see, even if we are manipulating the employee record, you can create other records from this automation. So you're not limited to just creating a record of the same type. You can create whatever record you want to create. Just make sure that you specify the type inside your record.create call.

Also, make sure that you set all of the mandatory fields, because if you forget to set any of the mandatory fields, it will give you an error message. Now, everything is OK for our scripts. Everything is working. But let's just here.

So for example, OK, HR is happy with the automation, because they don't need to remind employees to call them regarding their benefits. But one of the things that HR is complaining about is they want to call the employee back, but the phone number is not populated.

If I open my phone called record, I can see that the phone number is not mandatory. So I don't have that gold star or that yellow asterisk there. So HR wants this to be mandatory.

Now, for this change HR would not really be going to the SuiteScript developer, but instead they would go to the NetSuite administrator. And on some teams or in some companies, they have a separate NetSuite administrator from the SuiteScript developer. So what they would do here is like, OK, I'll approach my NetSuite administrator and say, I'll modify this form so that I can make the phone number field mandatory.

If I'm already on the phone call record-- and by the way, making fields mandatory or changing the form, that would be part of SuiteBuilder. Something that you can also do in SuiteScript, but generally if you do that in the UI, that will be SuiteBuilder.

If you modify the form, you can go to Customize and then click on Customize Form. And that will give you options to modify your phone call record. But if you're not in the phone call form, what you can do is that you can go to Customization, Forms, and then go to Entry Forms. Again, Customization, Forms, Entry Forms.

And then inside your entry forms, just look for the standard form. So here we're looking for the standard phone call form. By the way, if you see any of the standard forms, and you would notice here that the standard forms would have a negative internal ID value, these forms cannot be modified. So the standard phone call form, standard payment form, standard whatever form, would always be intact.

If you try to modify a standard form, they'll ask you to save that with a different name so that the standard form would still remain the same. For example, if I tried to customize this, So if I click on Customize on my standard phone call form, and if you look at the resulting form, you would see here that, OK, instead of giving me the standard phone call form, it's giving me "Custom Phone Call Form." That means I'm giving-- or I'm creating a custom form, a separate form from my standard form.

Now, for this one, if you're going to create a custom form, as an administrator you want to make sure you assign this to a specific group of people. If this would be used specifically by HR, then I will call this my "HR Phone Call Form." If this would be used by the whole company, for example I'm working for SuiteDreams, I'm going to call this "SuiteDreams Phone Call Form."

Again, we want to make it easier for users to look at a customization and then modify that-- or just make sure the customizations that you use would be user-friendly. So anyway, for the ID, same ID naming convention. Start with an underscore, then sdr underscore-- I'll say, just call this phone call. Make it simple.

And then since we want this to be used by all people in the company, I'm going to put a check mark on Record is Preferred. That means that the SuiteDreams Phone Call Form, that will be your default form for all phone call records.

Now, lastly here we're going to go to the Field subtab. And this is where the actual customization would come in. We want to make the phone call-- or the phone number field mandatory. So under the Field subtab I'll go to Main.

And then I'll just look for phone-- so as you can see here, I have the phone field. And I'll just need to put a check mark on the Mandatory column to make it mandatory. Put a check mark on that. Click on Save.

And next time a user creates a phone call record-- so if I go to Activities, Scheduling, Phone Calls, New, and if I look at my phone number field, OK, it looks like my phone number field is now mandatory. So everything is working fine. Our HR personnel is happy, our NetSuite administrator is happy because it's an easy change.

But if you, remember our script was working. Let's see if it's still working. I'll go to an employee record. And I'll create a new record here.

So for this one, I'll call this user Test 3 User. OK. Again, doesn't really matter which values we use here. And let's click on Save. Let's see what happens.

OK. So it looks like I have an error message. And this error message was thrown because of a SuiteScript error. I know that this is a SuiteScript error, because it says it's Notice SuiteScript.

And what we see here is that we have a message that says, OK, please enter value for phone. Now, as a user I'm going to look at this and say, OK, it looks like I forgot the phone number. I didn't really-- I didn't realize that the phone number field was mandatory on my employee record, but, OK, let's go ahead and modify that.

Now, the record was already created because this is running on my user event script. But for the user it's just, OK, it looks like it was created, but wonder why we got an error there? And then here we have a phone number. It's like, oh, phone number is not mandatory. So what did we get an error message?

So here the user would be very confused and say, I should not have gotten that error. I'm going to just-- you know what? Doesn't really matter. I'll just put in that phone number value there, click on Save. OK.

And yeah, for this one, because the record has already been saved, we got a different error message. But as you can see, it's just-- it's still not working even if I have my phone number value. So for the user it would be very confusing, because it's not allowing the user to save that new employee record.

But it's not really with the employee record. But in fact, the validation is happening on my phone called record. And the error is happening here when we try to create our phone call record. Or in fact, it happens here when we try to save that phone call record into the system.

It got an error because the system was expecting you to put in a phone number, but you didn't on your script. But here you can see the phone number is mandatory. Remember that business logic layer. And that business logic layer gives you that unified behavior regardless of the input channels.

So whatever you see on the form, like if there's anything that's mandatory on the form, you can also bet that that would be mandatory on SuiteScript or other input channels-- SuiteFlow, SuiteTalks, CSV import, and whatnot. So that means that this would be causing us problems.

And if you think about it, it's not really a problem with our script, because we didn't modify our script. The script was working. It's just that the form was modified, and the script would be using the default form that is on the system. And the default form is the SuiteDreams Phone Call Form.

If we modify the form, though, if we use the standard phone call form-- remember, the standard phone call form or the standard anything form would never be changed. So here, I can see that if I use the standard phone call form, my phone number, that value is still not mandatory.

So what I can do is that, instead of being at the mercy of whatever form is currently selected or preferred for the record, I can just always use the standard form so that I can be guaranteed that this would never be mandatory. So my script would always work.

What I can do is that, going back to my script, I can set another property. So remember when we were looking at the documentation, I have another property called defaultValues. And in the defaultValues property, I can set here-- so this is an anonymous object. I can have several values that I can set. You can just go to the Help Center for more information.

But basically, here I can say, I have a property called customform. And then for that customform property, I can set whichever form this script would be using to create the record. Now, we already know that the standard phone call form can be used, and that's not mandatory.

So if I go to, again, Lists, Records, Forms, Entry Forms-- I'll just open this is in a new subtab here. So if I go down and look for my standard phone call form, I can see that the internal ID is minus 150. So if I copy that and paste that over here, I can say that, OK, when you create this phone call record, don't use the preferred form. Use the standard phone call form.

Now, let's see if this would work. So let's upload that-- let's upload our script. And let's go to my employee form. Now, for this one let's just create the new one.

So again, if I go to Lists, Employees, Employees, New, and let me change this to-- I already have Test 3 User, so I'll just use Test 4 User. Again, doesn't really matter what values we use here. And let's click on Save.

And after clicking on Save, as you can see, I was able to successfully save my employer record and successfully execute my script. So we can verify that by going to my phone call list. And you can see I have another entry for "Call HR for benefits."

So I have two entries here. So I believe it's this one. So this is the latest one. And you can see that this was assigned to Test 4 User, which is the user that we have created.

Now, if I edit this, since the preferred form is a SuiteDreams Phone Call Form, you can see that the phone number field is still mandatory. We have not changed the UI in any way, but we have just forced our script to use a different form. Now, sometimes that's something that you have to do to make sure that your customization will remain intact even if the user tries to change-- or the administrator tries to change the configuration of your form. So that's something that you have to consider.

So that's about it for the walkthrough. Just again, if you want to add in any module, make sure that you include that module definition, add in that function or that repository for your module. You can name it any name you want, but the common convention would be the same name as your API module. Add your parameters definition.

And then after that, you can just go ahead and create your record, or load the record, or whatever you want to do. If you are going to create a record, make sure that you also call the save method so that you can persist that through the database.

If you want to create a record-- so if you use record.create, that would only create the object, or it will create that in memory. Now, if you don't save the record, or if you don't invoke task.create, what's going to happen is that it's not going to be saved on the database. So these two go hand in hand. Whenever you're creating a record, make sure that you save the record using task.save, OK?

Moving on with our discussion, let's talk about another functionality that we can add to our script called promises. Let's talk about promises. Now, what are promises? So promises are ways for you to get information or make a call to the server or call to something on a separate thread. So that's pretty much it.

So for example, this is a code that shows us how we can load a record using a promise. Now, promise is not a separate NetSuite API. But instead, promises are completely embedded within the NetSuite module.

Now, promises are not available on all modules. There are certain modules that use promises. And you can see that in the documentation. You know that a method or a module allows promises by looking at the definition.

For example, we've seen record.create and record.load. But here we have record.load.promise. That means that we are actually loading a record based on a promise call. So again, every time you perform a promise, that just means that you are triggering your script on a separate thread.

So in this example, we have a field change event. So on the field change event, we have-- let's say on the main process, so here I have a comment that says "execute mean process." Let's say that I have tons of lines here, so maybe dozens of lines that are heavy processing. And then here at the bottom, I have also lots of lines with heavy processing.

What happens is that, once your execution goes to your promise, this would be executed in a separate thread. So it's going to be thrown into a separate thread. And that would be executed there.

So once that is executed, then it's going to continue on with the process without waiting for this to finish. Now, why is that important? It is important because if you are going to create an automation, you have to think about the user's computer or the user's machine.

Because if you're going to create, say, a user event script, then you're guaranteed that your script would be running in one single environment, and that would be on the server resources. You don't have to worry about running your script on an older or slower computer, because it's always going to run on the server.

Now, on the client side, that can be problematic. Because on the client side, there is a potential that your user might be using an older machine with a slower internet connection. Now, if your user is just using your application at an office where you have a broadband connection and fast computers, that's not really an issue.

But what if you have a big company and you have, say, a manufacturing house or maybe a warehouse or something like that, where 20 users are sharing one machine, and that one machine is like 20 years old and has dial-up connection. That's kind of extreme, but just try to imagine if you have that kind of setup.

That's going to be problematic for your script, because the more processing that you put in on your client side script, the slower it would be. So again, it's not an issue if you have a fast computer with a fast internet connection. But if your user has a slow computer and a slow internet connection, that'll be very painful to use.

So by using promises, it allows you to just use multiple threads so that it would not affect the performance of your script. So your main process would be executed on one script. And then another thread would be executing your promise.

Now, for example in this case, since you're running this on the client side, if you're trying to load the record, you have to fetch the information from the server. So your script would be getting the information, sending your request to the server, and then the server would be getting a response back.

Now, you can see here that the syntax is a little bit different. So instead of returning the value, you have record.load.promise. So that's one thing that's different. The parameter list is the same. But you can see here that, OK, so what's happening? Where is the value going to go? Because it's not returning a value.

So here, what's going to happen is that once your value gets returned, it would be processed in a then statement. So here you have your record dot load and then dot then. And that result would be passed into a callback function, so similar to your define statement.

So if you look at it, it's a little bit-- the syntax is a little bit different if you're not used to this kind of syntax. But this is quite common for JavaScript. But here, so anyway you have a then statement. So you have a function.employee. And this employee object is the object that is being returned by your record.load.promise.

So again, it's not going to be returned immediately, but it would be returned once the process completes. Then once the process completes, then you can just do whatever you need to do for that promise call. So maybe if you want to get the value, or if you want to do something, just put that inside your callback function.

Now, if you encounter an error message, what would happen is that it will go to another section, which is a catch section on your script. So here you have catch, and then you have an exception. Then you can process that the same way you would process any exception. So you want to type in the name of the exception, the actual message, and also the stack trace.

Now, if you are going to use a promise, especially for error handling, one thing that you might notice is that the IDE would not recognize your catch statement. So the catch statement here would be identified by the IDE as the try-and-catch block. So you will get an error.

But you can just ignore that error, because this is not a catch block. This is a catch method. This is an actual function that is attached to your promise call. So again, if you see that error message, you can just ignore that. That still works.

What we're going to do this walkthrough is that we're going to just do something simple. We want to display an alert on the system whenever the user tries to edit a billed order. So for this one, aside from displaying to the user, OK, that order that you're editing has already been billed, we also want to notify the administrator so that we can track anybody who is trying to modify a bill.

What we want to learn in this walkthrough is that we want to learn how we can use the dialog module. The dialog module is a special module that is available in SuiteScript 2.0 that would allow you to display a modular window on your page.

Now, this has been present in NetSuite for a while, but this has only been opened for developers on the recent SuiteScript 2.0 update. The dialog module is also promise-based. So the behavior that you're going to get them the dialog module will be very interesting. We also want to learn here how we can send email messages. And also, we also want to discover the different strategies that we can have using promise-based dialog messages.

For this automation, what we're going to do is that we want to create a client side script for the sales order record. Now, remember that the promise API can only work on client side scripts, at least for now. So that's why we want to create a client side script. So let me close this up.

And let me create a new script file. So this would be for my client side script. And for the name of my client side script, again, same naming convention. I'll just call this sdr\_cs\_order. And for this one, I'm going to click on Add/Remove Modules so that I can add the module that we're going to use.

Specifically, we're going to add the email module and also the N/ui/dialog module. So let me add those two modules. And when I click on Finish, as you can see, everything has been predefined for me. Of course, I need to change my version number, so 2.0. So as you can see here, my email module and dialog module, my parameters, and also the functions, have already been defined.

Now, for this one we want this to get triggered when the user tries to edit a record. So this means that we need to trigger that on pageInit. So on my list of functions here at the bottom, I'm going to comment everything out. And I'll just remove the comma over there. And let's go back to my pageInit definition.

If you remember, pageInit would get triggered as soon as the page completely loads. Now, for this one we want to get this from our order records of our order and then context.currentRecord. So we're using order, because, again, this will be deployed to a sales order record.

We also want to get the status. Remember, based on our requirement, we want to only display this message if the status of our transaction is billed. So if it's already billed, then we want to display that to the user. And we also want to do this when they are editing. If they're viewing a billed record, no problem. But if they're editing it, then we want to make sure that we inform our administrator.

So we want to get the status. So our status is equal to our order. So for this one, I'll just use getText. And I will use the orderstatus field. So that's the field that I want to get.

And then I'll add in an if statement here. So if context.mode. So mode is a property similar to type. So if you remember, in the previous walkthrough we used the type property to filter if they're creating a record. But for a client side script-- by the way, it's not scriptContext. I'm going to change this to context before I get an error message.

So here for the mode, we want to check if they are editing. So I'll just use edit. Now, for the mode value [INAUDIBLE] we don't have [INAUDIBLE] but we should be getting that enum value in the future. For now, let's just use the string "edit." So again, it would still work.

Then also, add in another condition here. And I want to say that, OK, do the script if the status is billed. So that's what we want to get. So that's it.

So for this one, what I want to do is that I want to display a message on the screen, And while the user is reading that message-- that's the point of promises, you want to execute things on a separate thread. So while the user is reading the message, you're already processing things on the background. And anything that is potentially slow, such as sending an email if you have a slow internet connection, or loading a record, or searching, or doing whatever, you can optimize that by using those pockets of time. Whereas, instead of just waiting for the user to finish before proceeding onto your script, you can do that side by side. Again, that would save you a lot of time if you're dealing with optimizing your client side script.

Now, for this one I'm going to use the dialog module. So I have dialog.alert. Now, there are a lot of functions that I can use for the dialog module. I can use alert, which is similar to your JavaScript alert. I have confirm, which also is similar to your JavaScript confirm. Or I can create my own. So those are the options that I have.

Now, for my alert module, so if you want to know more information about this, what you can do is that you can go to the Help Center and look for your N/ui/dialog. So for this one, I have two properties. So have title. So here. I'll just call this-- I'll just say "Edit Warning."

And then for the actual message, I'm going to say-- OK, so for this one, "This order has already been billed. Auditors will be notified of this action," something like that. OK. So again, this is just an example. You can put in any message that you want to put in.

Now, while the user is reading that, I want to send an email. So I have email.send. Again, if you want to know more information about this particular function, you can find that information in the Help Center. So here for the author-- and for the author and the recipient, just to make it easier, I'm going to assign this to my account.

And my account ID is minus 5. So that's the default user. Recipients, we'll also use minus 5 here. And then for the subject of my message, I'll say my subject would be "User has edited a billed order." And then for the body of my message-- maybe just align that, make that [? part ?] clear.

For the body of my message, I'll say, order-- and then maybe I'll just get the order number. So here, order.getValue, and then get a transaction ID so that the admin or the auditor would be able to load that record, and then just say, "has been recently"-- just that. Keep it short, I guess. There we go.

And then that's it. So let me upload this and go to my NetSuite account so that we can create-- let met just close these. We don't need those anymore. Let me create our script record.

Let's look for our CS order here. Create our script record in-- for the file name or the name of my script, I'll just use the same naming convention, sdr\_cs\_order, deploy this to my sales order or my order record. Choose Order there. We use the same ID. Click Add, click on Save. That's it.

OK. So let's go ahead and test if this works. So let me go to a list of orders. I'll go to Transaction Sales, Enter Orders. But I'm not going to click on Enter Orders, because that'll give me like a blank sales order form. I'll go to List. Open this in a new subtab.

And let's look for billed orders that we can test this on. Let's see. OK. So it looks like the first one is billed, very convenient for us. So as soon as I click on Edit, I should see a pop-up on my screen. Let's see here. OK.

And as you can see, I have a message on the screen saying that this order has already been billed, the auditors will be notified of this action. And this is what the dialog or the UI dialog looks like. As you can see, it has the same user interface as NetSuite so that it's not too jarring.

If you used the alert statement, it would look like it's out of place, because it really depends on your browser. It would look like one way if you're using-- like, say, Chrome it would be a pop-on on the top of the screen. If you're using like Firefox, it would be a pop-up in the middle of the screen. Or if you're using another browser, it might be different.

But here, if you use the dialog UI it's always going to be the same. And with the dialog UI, this is a promise-based messaging system. So this means that this would be executed on a separate thread. And if I look at my code, since this section is on a separate thread, this should have already executed.

So let's verify if that has been executed. Again, we're not going to click on OK, so that we can show that while the user is reading the message we're already doing a lot of things in the background. And again, that would save us a lot of time, because instead of waiting for a process to finish, we're just doing or we're continuing on with the process.

So let me go to my employee record and look for Larry Nelson's account here. Let me open that in a new subtab. And then here, if I scroll down at the Communications subtab, let's look for that email. So some of the emails here are future dated.

So let me see. Maybe it's in the second page. So it should be-- OK, not second page. Probably the third page. And we are looking for-- it should be-- there you go. So this email has already been sent.

So this was sent by Larry Nelson. It was sent to Larry Nelson, because we have the same author and recipient. And we can say, OK, we have a notification that the user has edited a billed order. If we look at the details, we can read here that, OK, order number 216 has been recently opened, which is the message that we have on our account.

So as you can see here again, this is a very good use case for using promises, and also a good use case for using the dialog module. So I hope this was something that you can--

So that's about it. So a few things to remember before we go to our exercise. Only load the modules that you will be using in your code. If you're not going to use any of the modules, don't load it. Or, for example, if you were using a module but you stopped using it, then just go ahead and remove it.

Because the more modules that you load in your script, the heavier it becomes. And that can potentially slow things down. It doesn't usually, but that's just really an inefficient use of resources. So make sure you avoid doing that, OK?

Also regarding forms, make sure that you check the NetSuite UI so that you can verify the business logic layer. Now, if you remember, we are going to use the NetSuite form so that we can check the behavior of the record. If a record is mandatory on the form, you can bet that it's going to be mandatory on your script.

Also, if a field is defaulted-- like, if it's mandatory but it has a default value-- then that means that it also will be defaulted on your script. So the behavior would be the same regardless of the input channel. Also consider using the customform property. So if you use the customform property, at least you know that even if the administrator changes the form, then it should be safe, OK?

But again, the best thing I think that you can do instead of just using the custom form is that you can actually have like an open communication with your NetSuite administrator. Because if they've set a value as mandatory, there's a very good business reason why they're doing that. So working around that might not exactly be a good idea, at least not always.

So the best thing that you can do, if you can, try to inform your NetSuite administrator to tell them that, OK, I am manipulating this particular record. If there are any changes to the form, please let me know. And also the same way, your administrator would also check up on you to say that, OK, I'm going to do something for this record. Is it OK, would that affect your script. So having that open communication channel would be a lot better.

Then also, if you are going to create a record or edit a record, make sure that you save your changes to the database. So every time you call record.create, make sure that you also have record.save. Because if you don't, you're going to lose everything that you have created. So there you go.

A few things to remember regarding promises. These are the modules that support promises, at least for now. So we have the email module. So you can use promises for that. You have the HTTP and HTTPS module for calling web services.

I think this would be the most useful part of promises if you're going to call web services. Because if you are, that typically takes a long time to process because of the amount of information that you usually send. So promises would be very useful for that.

You also have the record module, the search module, transaction module, and the UI dialog, which we just used in our walkthrough. Also, right now you can use native JavaScript promises. So if you want to create your own promise, so you can have like variables equals to new promise something. So if you are familiar with native JavaScript promises, you can use that.

If you're not, there are some documentations in the Help Center to guide you on how you can create your own custom promise. And like what I mentioned earlier, promises are client only, at least for now. We're looking into how we can have that apply to other script types. But right now, it is only available for a client side script.

Now it's your turn. Please complete the following activity. This concludes module 5-- SuiteScript modules.