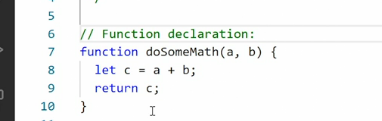
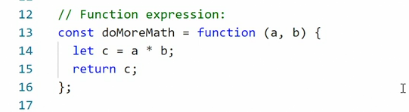
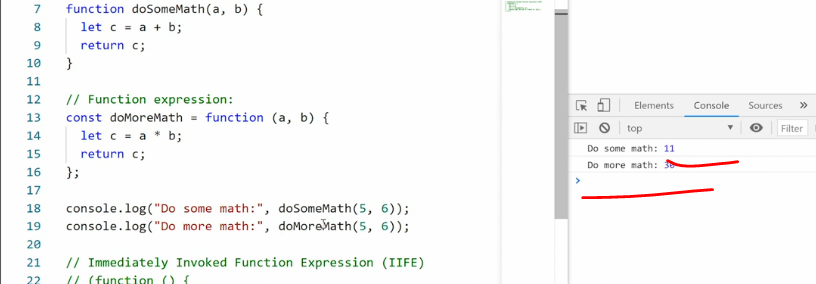
* - [Narrator] So far in the course, we've mainly looked at data and how to handle it through objects and variables and arrays.
* But the main use of JavaScript is as a scripting language, meaning we use it to write programs.
* And that's what functions are for.
* A word of note here.
* The term function is ***typically used to refer to as a function that sits on itself*** whereas a **method is a function that sits inside an object and acts on that object.**
* They're effectively the same thing, they just appear in different contexts.
* Okay, a function can be created in a couple of different ways and how we create a function has an impact on how it can be used.
* **The classic function is a function declaration** and it looks like what you see here.



* We start with a *keyword function*, then we *give the function a**name, we pass parameters inside a parenthesis* and this can either be nothing.
* So it can be just the parenthesis or it can have parameters.
* And then we have *curly brackets wrapping* around the function body.
* Inside the function body, we perform ***actions on the data.***
* And then we can, if we want to return that data using the ***return keyword****.*
* That data gets then sent back to wherever we declared the function originally.
* Now, we don't have to do that, we can also have the function just output data directly.
* So that depends on what you want to do with the function.
* If we use a function declaration like this, ***the function is hoisted to the global scope, meaning it becomes available everywhere.***
* So even if we declare this function inside a local scope, it'll still be available everywhere else in the script.
* This function is also *really an advanced variable and it can be re-declared.*
* So just like a VAR, if you create a new function and use the same name, do some math, that new function will take precedence further down in the code.
* So you can accidentally override an existing function if you're using function declarations.
* Function declarations are easy to understand even at a glance because they quite literally say, I am a function and I have a name and these are my parameters.
* So the clarity of intent is clear any time you're using a function declaration.
* And finally*, function declarations are parsed in the order they appear in the code.*
* So if you have a function B that relies on function A, you have to make sure function A is declared before function B in the code.
* Otherwise, when you get to function B, function A is not declared yet and the script cannot run.
* So the order in which these functions are declared matters for JavaScript to work properly.
* **Another way of declaring a function is through a function expression**, and that's what you see here.
* In a function expression, *we set up a variable.*
* In this case, *a constant*, we give that constant a name and then ***we set the value of that constant equal to an anonymous function.***
* So this is a function that ***doesn't have its own name.***
* This anonymous function, again, ***has parentheses and those parentheses may contain parameters.***
* And then we have *a curly bracket wrapping around the function body.*
* So the function functions the same way as the function declaration, except it doesn't in itself have a name.
* Instead, we place it inside a variable and then we call the variable.
* This pattern of using a function expression has become the *preferred way of declaring functions* in JavaScript frameworks like react.



* And the current best practice is to place function expressions inside a const.
* That makes sense if you think about it because the function expression is sitting inside a variable and ***it will then have the same scope as that variable type.***
* So if we're using a const, then it is ***locally scoped or block scoped automatically.***
* So if you ***place the function in a const, it also cannot be re-declared.***
* So you're never in danger of accidentally overriding or destroying your function.
* *Finally, function expressions are not hoisted.*
* *They exist only in the scope they were created*, and this is also facilitated by the const being used.
* So in a way*, function expressions are a more advanced version of functions.*
* *To call and use either a function declaration or a function expression, we simply name it and add parentheses at the end.*
* And then if there are parameters, we pass those parameters along.
* You can see that down here.
* First I'm calling the do some math function.

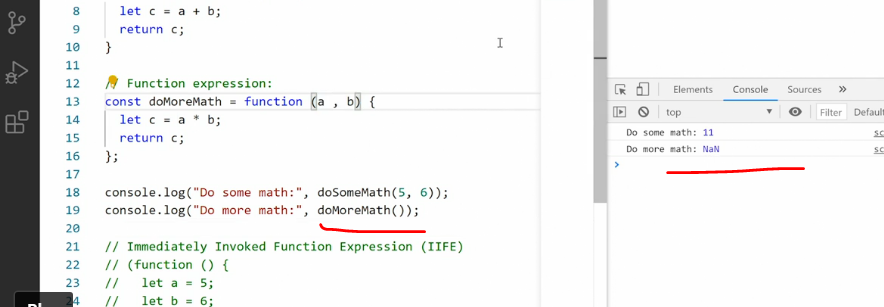


* That's the first one up here and I'm passing along my parameters, the number five and six.
* They then get populated as A and B, and A and B is used inside the function.
* Same thing below, here we call then do some math.
* You can see the calling of the function is exactly the same whether it's a function declaration or an expression.
* And again, we pass in the values.
* ***Interestingly, because we have a function expression, if we just say, do more math and save, what we get in return is the actual function instead,*** which looks weird but can be useful in some edge cases.

Graphical user interface, application

Description automatically generated

* So using a function expression gives you some added functionality that can be useful.
* One more thing, ***if we don't pass values to do more math***, even though it's expecting it, we'll get some weird output.



* Here we get NAN or not a number.
* ***We can solve that by setting default values for these parameters in the function expression itself.***
* So we can say A equals three and B equals two.

Graphical user interface, text, application

Description automatically generated

* So then even if we don't pass values, we get an output and it will be the default output.
* But if we pass in some values, five and six, then those values we're passing in take over and we get the correct output.

Graphical user interface, application, Word

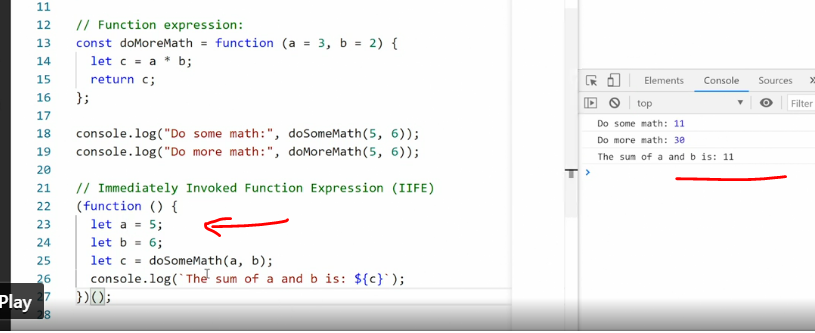
Description automatically generated

* In addition to a function declaration, a function expression and the anonymous function that sits inside here, there is one more type of function worth mentioning.
* It is the **immediately invoked function expression.**

Graphical user interface, text, application, chat or text message

Description automatically generated

* And you see it down here at the bottom.
* In an IIFE, ***we wrap an anonymous function or a name function inside parentheses.***
* You can see here there's a starting parentheses and an end parentheses.
* ***And then we put place another set of parentheses outside.***
* ***What happens now is this function is immediately invoked and run as soon as the browser encounters it.***
* ***So we are running this function right away without calling it.***
* *This particular pattern is a bit of an anti-pattern because you normally want to control when a function runs, but in some cases it can be handy because you want something to happen as soon as possible.*
* And you can see inside this function I'm actually using one of the other functions, do some math.



* So when I save this page now, you'll see we get an output here where we pass in values for A and B, and then we do some math and we get the output we want.
* And then we can change these values here and get a different output.