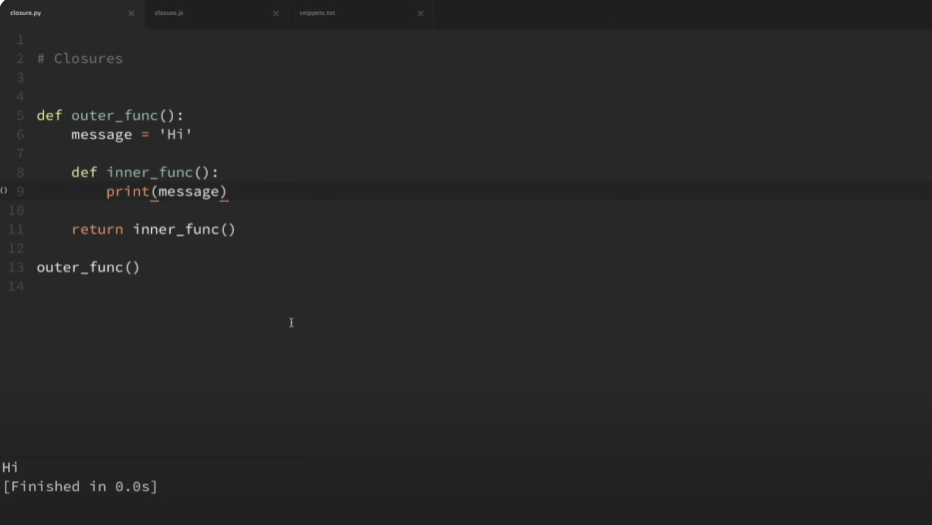
Hey everybody, how's it going? In this programming terms video, we're going to be going over the term closure. We're going to look at closures in both Python and in JavaScript so we can see this concept in two different languages, and that may help the term sink in a little bit. So don't worry if you're not familiar with one of these languages. What's important is to take away the concept of a closure and not the specific syntax of any one language. So first of all, to better understand closures, you may want to watch my video on first class functions because understanding those will help you better understand why closures work the way that they do. So, basically to quickly summarize that video, first class functions allow us to treat functions like any other object.

So, for example, we can pass functions as an argument to another function, we can return functions, and we can assign functions to variables. So that's gonna be important because we'll see this a lot when working with closures.

Okay, so what exactly is a closure? If we look at the Wikipedia definition, then it defines a closure as, “A closure is a record storing a function together with an environment: a mapping associating each free variable of the function with the value or storage location to which the name was bound when the closure was created. A closure, unlike plain functions, allows the function to access those captured variables through the closure’s reference to them, even when the function is invoked outside their scope.”



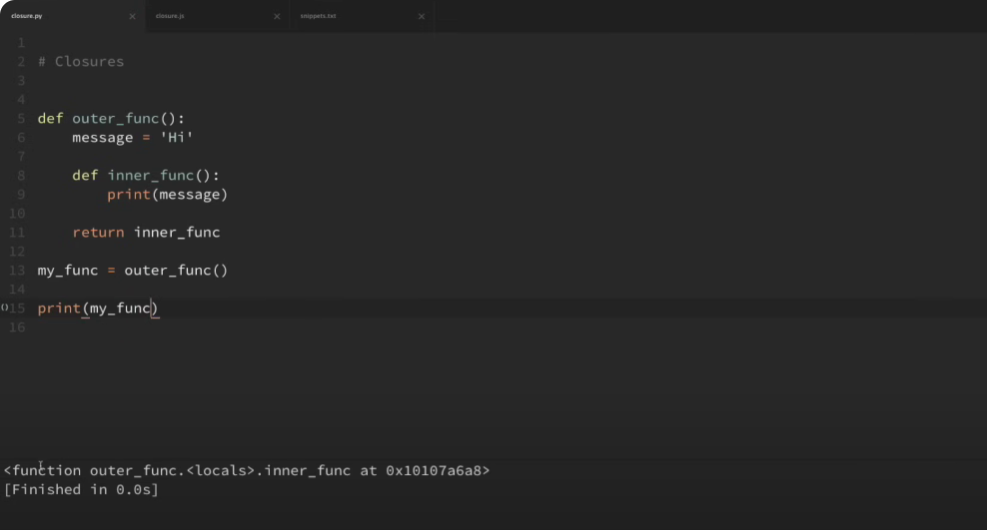
So, I know that may be kind of a difficult definition to understand, but let's take a look at a few code examples to really understand this.

Okay, so my first example here is in Python. Now let's walk through the code and point out exactly what's going on here. So, you can see here when I run this outer function that it executes the function, and it doesn't take any parameters. Now the first thing that it does is it comes in and assigns this variable message with the value of high. And then we have an inner function here. Now this function doesn't take in any parameters either. And all it does is print the message variable. Now when our inner function accesses this message variable, this is actually what we call a free variable because it's not actually defined in the inner function, but we still have access to it within the inner function.

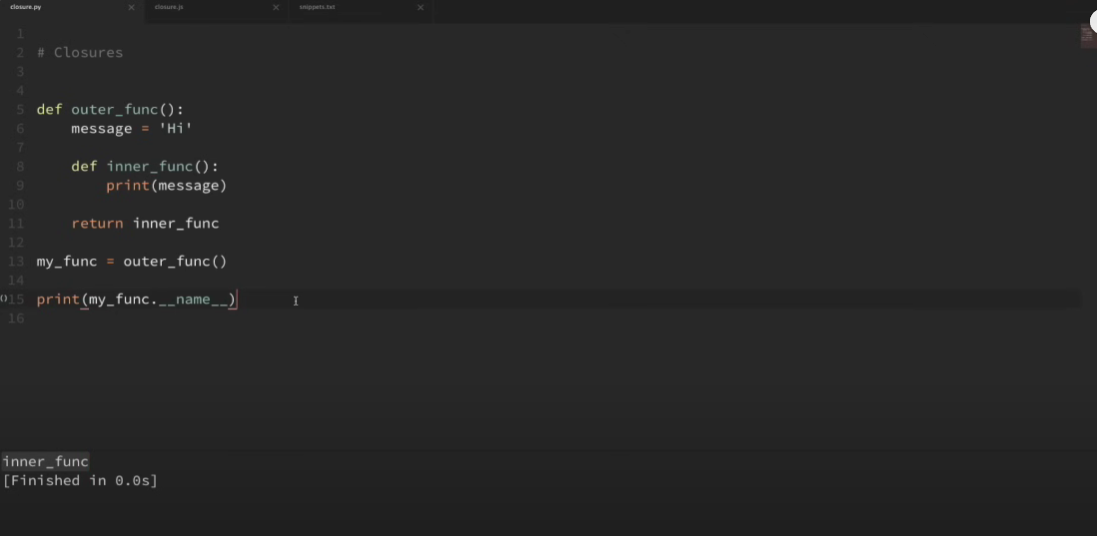
So then down here, we are executing this inner function and then we are returning that. Okay, so if I run this code, then you can see that it prints out the value high. So again, really fast, what it did is we executed this outer function, it came in here, assigned this message variable with this value, then we created this inner function from within our outer function, which just prints that message variable, which we call a free variable inside inner function. And now we're returning the inner function and executing that inner function also. So it should print that message, which it did.

Okay, so now let's do something different here. Instead of executing this inner function and then returning it, let's just return the function without executing it.

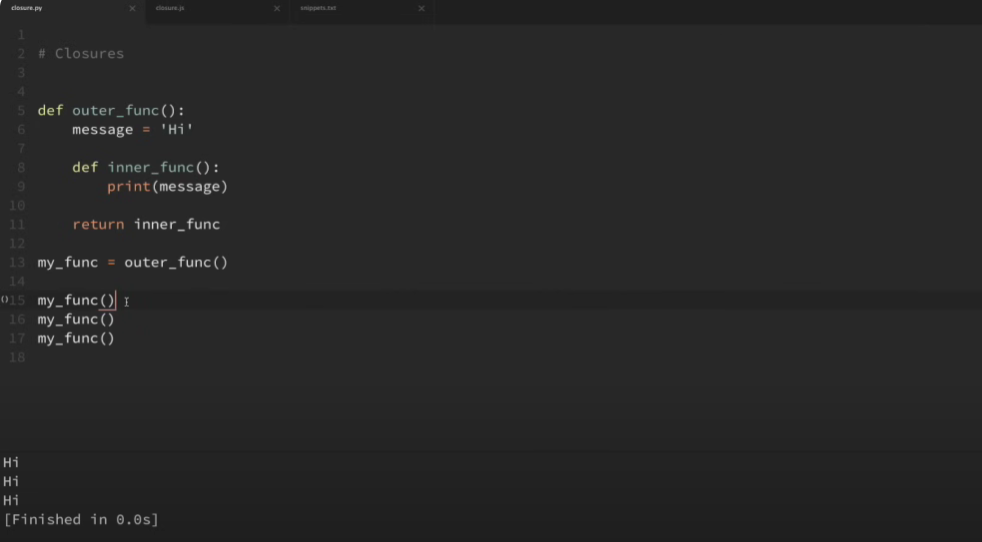
And we can do that by taking away the parentheses here and saving that.



So now if I run this, it doesn't look like anything happened. But what I'm also going to do is I'm going to assign this to a, variable called my func and then if I run that again, you can see that it still doesn't do anything. So let's walk through this really quick. When we execute this outer function, it comes in here and it does everything that it did before. But instead of executing the inner function, it just returns the inner function. So now my my func variable here is actually a function now. And it's equal to the inner func function because that was what we returned. and we can see this if we print my func and if I save that and run it, then you can see that it is a function and that it's equal to this inner function and I can actually print out just the name of that and if I run it, you can see that the name of the function that this variable is assigned to is inner func.



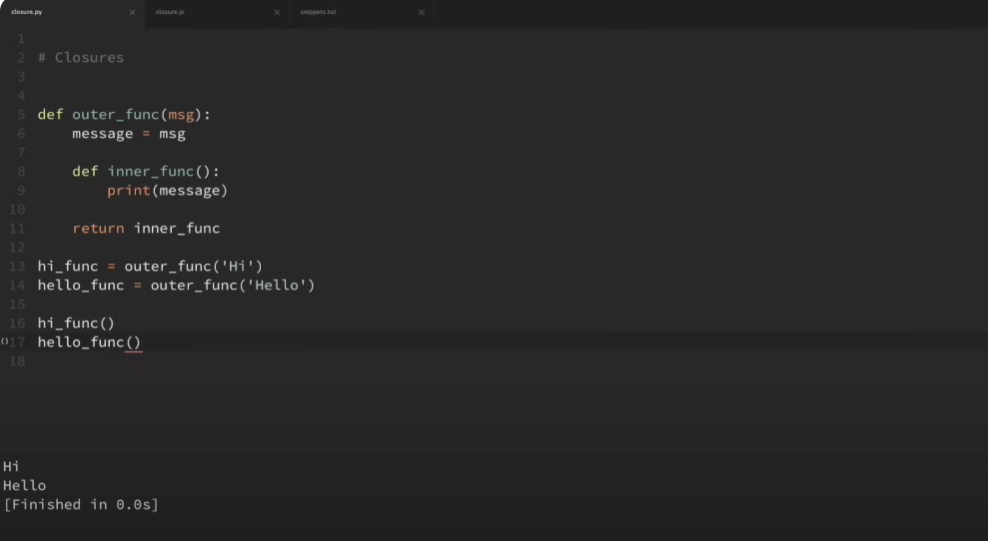
So now I can execute this variable just like any other function. So if I execute this a few times in a row here, and then save and run that, now you can see that it printed out our message of hi. Now this is kind of interesting because we're done with the execution of our outer function, but the inner function that we returned still has access to that message variable that it's printing out. So that's what a closure is. So, in simple terms, a closure is an inner function that remembers and has access to variables in the local scope in which it was created, even after the outer function has finished executing.



Now this gets more interesting when we add parameters for our functions.

Now our functions don't currently have parameters, but let's go ahead and add a couple really fast. So in our outer function here, I'm actually going to pass in a parameter of msg for message, and then I'm going to set this message variable equal to the value that we pass into our outer function. So now, whenever I set this variable here, so let me create two new variables. I'm going to call this high function, and I will call another one down here Something like hello function. And now I actually have to pass in this message parameter here. So I'm going to pass in the same one that we had before for that one, which is high and for this one, I'm going to do hello.

So both of these variables are taking in a string argument and and then it sets our message variable to that string and returns our inner function ready to print that message. Now one thing we should notice here is that our inner function still doesn't take any arguments. So when we execute these functions, we only need to add an empty parentheses. So if I go ahead and copy both of these, then I can just paste these in here and let me put empty parentheses to execute both of these and run that. So when I print these out, you can see that each of these functions remembers the value of their own message variable. So I forgot exactly where I read this, but I saw that one way that you can remember this is that a closure closes over the free variables from their environment. And in this case, message would be that free variable.



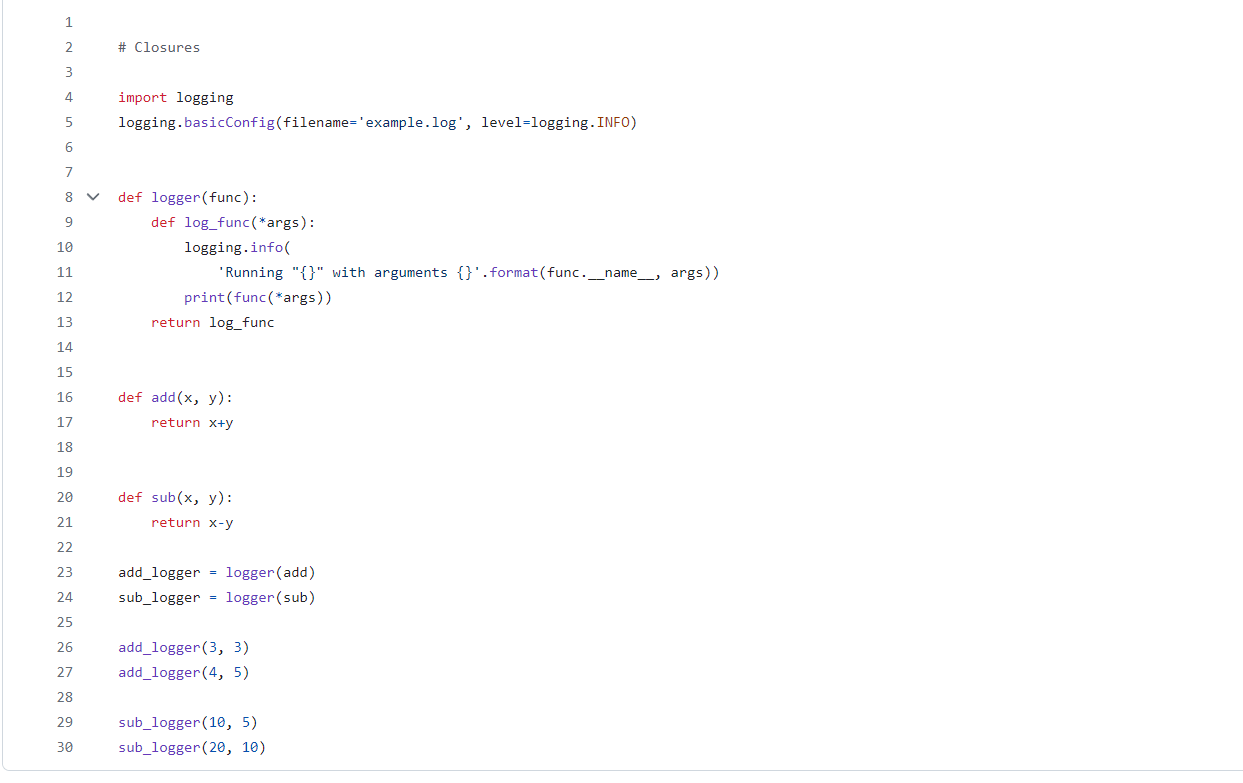
Okay, so now let's look at an example in JavaScript that is a little bit more complex. Now, my hope here is that by seeing this in two different languages, that it'll really knock home the concept rather than just the syntax of the language. So this is an example that I used in the first class functions video. Okay, so here I have a function called HTML tag which takes a tag parameter. Now this is our outer function. Then we have an inner wrap text function that takes a message parameter. Now all our inner wrap text function does is logs a message to the console which wraps our message within these tags. And then you can see down here that we are returning the wrap text function.

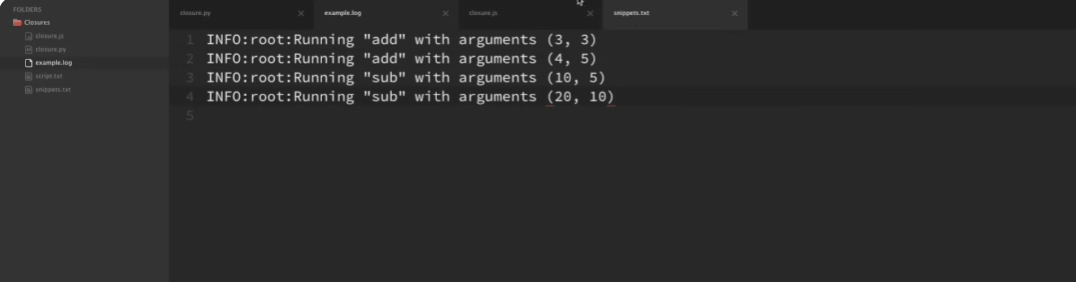
Okay, so here I am setting this print h1 variable to our outer HTML tag function with the argument of h1. Now what it's going to do is it's going to execute this HTML tag function, set this tag variable to h1, and then return our wrap text function ready to accept a message. So there's a difference from our last example. So here the inner function we are returning does take one parameter. So when we call this print function, we need to pass in a message argument. Okay, so now I can treat this print variable just like it is our wrap text function waiting for this message. So what I'm going to do is uncomment out this code here and save that and go ahead and run that.



So, you can see when I ran that, that it remembered that we passed in the H1 as our tag, and then printed out each individual message wrapped in that tag. And I also have an example down here with a paragraph tag, and if I run that, then you can see that that works also.







Okay, so lastly, let's take a look at one last example, and the last example will be in Python. Now this will be a more practical example, and also slightly more complex. So let me go ahead and grab this code from my snippets here, and I'm just going to go ahead and paste that in here. Okay, so let's quickly walk through what this code is doing here. So we have an outer function here called logger, which takes in a function as its parameter.

The inner function called logfunk takes in any number of arguments, and that's what the star does here. It's basically saying that we can pass in any number of arguments to this function. So then within the inner function here, we are logging that we are running the function that we passed in, and it also prints out the arguments that we used with that function. And then we actually execute the function here with the arguments, and then we print that to the console. And then you can see down here that we are returning this inner log function. Okay, now down here I have two very simple functions, add and subtract, which all they do is simply add and subtract two different numbers. Okay, now we have two variables here, one called addLogger and another one called subLogger, which we pass in our add function to our outerLogger function, and here we pass in our sub function to our outerLogger function.

So now we can use these two variables that we created as if they are our inner function. And just to remind you what the inner function looks like, it accepts any number of arguments, and then it comes in here and logs that we're running our function with the arguments, then runs that function and prints it to the console. Okay, so I'm just going to go ahead and run this really quick. Now we can see here that it printed out our results to the console. So we did our add logger of three and three, and it printed out six, four plus five is nine, and then 10 minus 5 is 5, 20 minus 10 is 10. So it printed out the correct values to the console.

Now we could have done that with our original add and subtract functions. But what creating a closure did for us is that we can see if we look into our files here, that it also created this example.log. And you can see that it should have logged this information here that we were running a specific function with specific arguments. So if I open up this log file, you can see that each time we ran a function that it came in and logged the function that we were running and the arguments that we passed in. So you can see how these closures can be extremely powerful and there's countless different ways in which we can use them. Now this specific example with logging in Python would probably be a better use case for a Python decorator, but that's going to be a topic for another video.

Okay, so I think that's going to do it for this video. I hope this helped you better understand closures and also gave you some ideas for how you can use them within your own applications. But if you do have any questions, just feel free to ask in the comment section below. Be sure to subscribe for future videos and thank you all for watching.