Asynchronous Code

Video Transcript

Video 1 – Introduction to Asynchronous Code

Performing operations in a sequential manner is mostly what we have done. That is you take one step then another. You have a sequential set of operations which at times is called imperative programming or procedural programming. However, in real life, many things happen at the same time. That is, you might have things running in parallel. They might be asynchronous, meaning that a set of computation starts off and you're not sure when it's going to come back.

So, learning some of the basics as to how to handle this environment is very important and it's central to working in a web environment every single time you make a call to a server, you're not sure if that server is going to respond, it could be that it's lightning quick. It could be that it's slow, it could be that it's down.

And so, learning to work with failure, learning to handle the variant responses in time is something that will make your code much stronger and much better. And as a general idea, it's something that you want to carry forward with you this understanding that many other things might be happening at the same time and having some understanding of the basic constructs in the language that allow you to deal with these situations.

Video 2 – What Is Asynchronous Code?

Let's talk about asynchronous code. And by this, we mean code that runs outside of the main program flow. Up until now, we have written procedural, sequential, sometimes you'll hear it described as imperative code, where you have instruction "a" followed by "b", followed by "c". In the code that you see before you, you have three very simple functions, function a( ), then b( ), then c( ), all of them returning simply the name of the function 'a', then 'b', then 'c'. And so let's go ahead and call each one of these. We're simply going to write a 'console.log( )', Inside of it we're going to call the function '(a( ))'. And then we're going to repeat this call three times, calling 'b( )', then calling 'c( )'. Now, I'm going to go ahead and load this into the browser.

So, you can see it here. And then I'm going to go ahead and open up my Developer Tools. And as you can see there, we execute the instructions in sequence a, then b, then c. So, that was an example of sequential execution. Now let's consider an example of asynchronous code. We're still going to have a, then b, then c. But in parallel to that, we're going to have an instruction, a timer that is not within the main thread. Now let's go ahead and add that timer to our code. I'm

going to go ahead and do it in the first function, I'm going to enter a 'setTimeout( )' inside of it I'm going to add a '(function( ))' which is a callback. And I left an n out there and then enter the body of the function. And inside of it, I'm going to do a 'console.log( )', and inside of that message I'm going to write '('a - in timer');'

So, as you can see there I have added a timer that has a call back that when the time runs to its finality. And in this case, I'm going to set it at two seconds. That 'console.log( )' will execute. So, let's go ahead and reload our page. And as you can see there, we immediately get a, b, c, and then in two seconds, we get a - in timer. So, it is running through all of our procedural instruction, sequential instructions. And something is being put to the side that takes two seconds to come back. So, this is an example of an asynchronous instruction, that is off the main thread while the main thread continues to run. And just in case you didn't catch it the first time, let's go ahead and reload and will get a, b, c, then a pause, then a - in timer.

Video 3 – Promises

An asynchronous operation is one that happens outside of the program's main loop. And in this case, the example that we did was a timer. You can think of it as the main thread, as being the event loop. And those asynchronous operations registering callbacks then most likely performing an expensive operation, like writing to the file system or a database and then coming back to the main thread. Within the language, we can use Promises to handle the eventual completion or failure of an async operation. And you can see here the workflow, at this part here of the workflow, we can execute the asynchronous actions or handle the error, come back to that Promise wrapper and then handle that result with a .then or .catch.

So, let's take a look at an example. As you can see here, I have once again 'function a( )', 'b( )', and 'c( )'. And inside of each one of these, now I have a 'new Promise'. That new Promise is wrapping a timer, each of which executes at three seconds after that it's fired. Now in addition to that, I'm going to go ahead and execute all of those functions. And I will start by creating a 'const list' and inside of here, I'm going to create an array with the functions, function '[a, b, c];' Now I'm going to write a 'for( )' loop in which I will walk through every '(fn of list)'. And I'm going to write the body of that for loop. And the very first thing that I'm going to do is that I'm going to write the function to the console just to see what is being considered in this loop iteration.

And then I'm going to go ahead and execute that function and handle the Promise with '.then'. Inside of it, I'm going to write my 'function', my callback, that will handle the output of the function. I'm going to use the word '(done)'. And then I'm going to write the body, and inside of it, I'm going to write a 'console.log' and then write '(done);', which is what comes back from the Promise once it's resolved. And let's take a look at what we see on the console. I'm going to go ahead and save the file. And before I run it, let me just note that what we choose to see very first is that we should see the function being written to the console.

And then after that, given that each of these Promises is being delayed by three seconds, we should have a pause of three seconds and then all three functions should come back at once. Let me go ahead and load that into the browser. And I'm going to go ahead and open up the Console And as you can see here, I have an error and you may have already spotted it, but I just saw it myself. It's here where I added an extra letter. I'm going to go ahead and save the file and reload. Let's go ahead and see what we see. And you can see there all three functions at a pause of three seconds, and then all three functions executed once. The reason for that is because each one of these is being queued in parallel. That is, the time elapse for each of those functions is all starting at the same point. And so, they all come back at the same time. Let me go ahead and reload the page so you can see it once again and in a second, they all come back.

So, if we create a diagram, the diagram would be like the following. All of them fire off at the exact same time. They are running in parallel, so they all finish at the same time as well. Now let's take a look at a very similar example, except in this case we have different timings. As you can see here, function a( ) comes back and three seconds, function b( ) comes back in half a sec, and function c( ) comes back at 1.5 seconds. I've changed the syntax slightly as well. I'm writing now just the function name to the console.

And as you can see here, I'm using the fat arrows notation as opposed to writing the full syntax of the function. So, let's go ahead and reload that. As you can see, the functions come back right away, the function names. And then we get b back, which is the one that executes and half a second then that function, then the output is followed by c, which is the one that executes and 1.5 seconds. And then finally a, which is the one that takes the longest. I'll go ahead and reload so you can see it again. So, once again, if we present this diagram form, you can see there that b comes back the fastest, followed by c, followed by a. All of them executing in parallel.

Video 4 – Async And Await

Promises are great. However, the .then notation and the chains derive from it can turn out to be pretty messy. To address this language has provided cleaner syntax and that is asynchronous functions, async functions, and the keyword await. As you can see here, async functions return a promise without having to create a promise inside of the function and await operator is used to wait for a promise. So, let's go ahead and take a look at an example. As you can see here, I have a very simple 'function' called 'resolveAfter2Seconds( )'. And inside of it, I have a 'Promise'. And inside of the Promise, I have a 'setTimeout( )' that will fire in two seconds. I'm now going to consume that with an 'async function'.

This is part of the keywords that I just mentioned. And I'm going to call the function 'asyncCall( )' because it's going to call the function with the Promise and inside of it, I'm simply going to write to the console the following. I'm going to call '('calling');'. And then I'm going to catch the result in a 'const' called 'result'. And I'm going to wait using the keyword 'await'. And then I'm going to call the function called 'resolveAfter2Seconds( )'. I'm then going to write to the console the '(result);'.

Now the last thing I need to do is to invoke that function, the async function. And now let's go ahead and load that code in our browser. So, you can see it's calling, pausing, and then it is resolved.

So, the async function first writes to the console that is calling. It then waits for the result and the resolution of the Promise, which in this case resolves in two seconds. And then the message that is sent back I am writing to the console. Let me go ahead and reload that once again. You can see it's calling and now it's resolved. Now let's go ahead and take a look at an example that is more involved as you can see here, I have code very similar to one I've used before. I have three functions. Each of those functions has a Promise, and inside of that Promise, there is a timer. Now, in this case, note that each of them will execute and three seconds, and in this case, because we're using, or we will be using async and await, we will be executing in series, that is one after the other.

So, let's go ahead and write our code. I'm going to go ahead and scroll a little bit down. I'm going to create a 'const list' where I will hold my functions, it's going to be function '[a, b, c];'. And then below it, I'm going to write the 'for( )' loop where I will step through each of those '(fn of list)'. Then I'm going to write my for loop body, the 'const result' that will be coming back once I call the function. However, in this case, I'm going to use the keyword 'await'. And then I'm going to write my '(result);' to the console. Now because we're using the keyword await, we must use it inside of an async function and so I'm going to create a wrapper to do that. And I'm going to put the code that I've written inside of it. Let me clean up my code here a little.

And as you can see there, the little red underline is because I haven't given it a name. But in this case, we're simply using it as a wrapper. So, I'm going to wrap all of it in a parentheses and then execute it as soon as this code is reached. So, let's go ahead and take a look and see what this looks like in code in the browser. Let's go ahead and reload that page. And we are now waiting to see if we can see the code and you can see there a, b, then c. And note that in between each of those appearing on the screen, we get that three-second delay. First a, then next will come b, and then the last one will be c. So, as you can see, these are sequential instructions that are taking place. As opposed to the previous example where we were executing in parallel.

Video 5 – Async In the Modern Browser

The differences between asynchronous code and synchronous sequential code may seem academic. However, they have a tremendous impact when it comes to performance, and especially when it comes to user experience on the browser. Let's take a look at a modern web page and make some observations. As you can see here, I am on a blank browser page and I am at the Console. I'm going to go ahead and move to the Network because we want to monitor the amount of communication that this window is going to carry out as we navigate to a site. I'm going to navigate to the New York Times, a popular newspaper in the US.

And I want you to pay attention to the amount of communications that are made from this page to external servers and that number will appear on the lower left-hand side. So, let's go ahead and navigate to the times. And as you can see there, we have over 300 requests that are made to external servers from this page. And as I scroll through the side, you'll see that number increase and very quickly, we are over 400 requests and if we keep on going, you can see there that we have already exceeded 500. And a request to a server is something that is unbounded. That means you don't know if that server is up.

You don't know if that server is slow. You don't know when that server is going to respond and because of it, you would be in trouble if you had to wait sequentially for each of those connections, you'd be much better off doing this in parallel and refreshing your UI for those components that have come back. So, let's take a look at some code that makes one of these connections to see the pattern more closely. On the left-hand side here in the editor I have some code, just some boilerplate in a second I will write the code that will make the request. But let me make a small comment here that this code would only work with servers that allow CORS and CORS stands for Cross Origin Resource Sharing.

Most of the private servers will not allow it, but many of the big services from Google and others do allow these type of connections. So, I will start out by setting the URL that we're going to navigate to. And as you can see there that is a URL to Google fonts. Next, I'm going to make the request. And I'm going to catch the response in a variable call 'res' for response and we're going to wait for that answer coming back from the server and the parameter that the 'fetch( )' function takes is a '(url);' and it returns a promise, and we wait for that promise to be fulfilled using await. Then we're going to extract the data.

And we're once again going to use 'await'. The first one makes the connection. The second one waits for the text within the page. As you can see there I missed out an s, and then on the last step, we're going to write to the console. Now let's go ahead and reload our page. And as you can see there, we have brought back that font from Google. Now, as you can tell, that was very fast. We're hitting one of those global services that have high performance. However, let's take look next at an example where we're making many connections to many servers, and some of the questions that we'll address are what happens when things come back out-of-order?

How do you address different performances? And such type of things. As you can see on the left-hand side, I have a long list of addresses to MIT's course catalog. On the right-hand side, I have a terminal window and in this case, we will be using JavaScript, very much like what we wrote for the page, but now we will be writing it or running it at the Console. If you do not have the configuration for this, that is okay. I simply wanted to illustrate the asynchronous nature of these request and what we can expect when we make those connections from the server and I wanted to do it at the Console because we can do it in a more transparent way.

So, let's go ahead and write some code. As mentioned, we have here over 45 addresses that we want to reach out and pull down. And then we have a couple of function stubs, as you can see

here one is called 'makeRequest', and the other one is going to loop through all of those addresses and then apply the function that we will write. But let's get started with 'makeRequest'. We're going to go ahead and catch a response. And let's go ahead and use 'const' here and we will 'await' the 'fetch( )' of that page. And in this case, it's going to be a 'url' that we're passing in, and each call of that function is going to take two parameters.

The first one is the 'url' and the second one is a 'counter' and that's simply to keep track of which of that long list we're making a request for. On the next line, we will catch the data coming back and we will once again wait for the 'response.text( );' in this case. We will then 'return' to the calling function with the following text string. I will enter here ' 'Done:' + counter;' number. Once again, to know which requests we are working with. Inside of the loop of the addresses, the function that we will use to apply to each of those addresses is the following 'makeRequest( )'. And we will pass the 'url,' plus the counter.

The url is going to be the item in the list above and 'i' is simply the counter of that long list. Then because this is an asynchronous function, we will be making. We will be handling that as a Promise. and so, we will use '.then( )' and we will handle the resolution of that with a callback. And so, I will enter here '(function(result))'. Then the body of the function. I will write the '(result);' to the console. Now let's clean up our code and remove some of the empty spaces and now let's go ahead and run that code. The JavaScript runtime we're using is node and the filename is fetch.js. So, let's go ahead and execute that and as you can see there, you can see the asynchronous nature of the resolution of each of these requests.

The first one to finish is 1, but then the next one is 16, the one after that is 11, then 23. And so, you can see there that we have very different and unpredictable results if we were to run it again. Let me go ahead and run one more time. You can see there that now the first one to finish was 27 then the following one was 14, and so on. So, as you can see, one of the central patterns of the modern browser is to have constant communication with servers in the hundreds in asynchronous events and being able to manage, handle those to have performance, to have a good experience for the user is central to understanding how to work with asynchronous code.

Video 6 – Performance Budget

If this is the first time you've come across the notion of synchronous versus asynchronous code and delays on the browser, you might be tempted to think, big deal if the user has to wait for an extra half a second, or a second. Well, as it turns out when it comes to user experience, there is a high sensitivity to delays within the user interface. And as you can see here, Amazon discovered that even a tenth of a second difference made a significant impact when it came to sales. That in this case, it led to a drop of one percent when it came to one-tenth of one-second difference inload time.

And so, as it comes to the user experience which ultimately determines the success of many digital services on the market. Sensitivity to time is one of the least tolerated when it comes to the user. And so, understanding the performance budget, the time constraints that you have, and being able to work with asynchronous code to provide a better user experience is key to successful products today. If you want to learn more about the topic, do a search for user experience and performance budgets and you will see there a combination of great engineering and user experience considerations.

Video 7 – Map Hello World

Maps are a great example of bringing in resources onto our page and mixing them up with our UI. We can bring in the libraries, we can bring in the styles, and we can lay out great resources in this way. In this case, we're going to bring in maps. And so, we're going to start off by bringing in the library. And that is mapbox.com's library, as you can see there. And we're using this one because it allows you over 15,000, over 50,000 actually views per month for free. This is the library. Now I'm going to go ahead and bring in the styles. This brings in the styles for the mapping library. And we can view either one of these by navigating directly in the browser.

So, in this case, we're looking at the mapping library and you can see there how big it is even though it's minimized. And I could do the exact same thing for the styles by accessing them directly. So, by doing those includes in the HTML document, we're loading all of those resources and they will now be available for us to use within our page. Next, I'm going to add some styles. I'm going to set some styles for the body because I will be laying out a map. I want my 'margins:' and my 'padding:' to be '0;' to make sure I have no spacing around the map, then the styles for the map element are going to be the following.

I'm going to set my 'position:' to be 'absolute;'. My 'top:' at '0;', 'bottom:' at '0;', and my 'width:' to be '100%;'. Now, within the body of the document, I'm going to create the element that will hold the map and I'm going to use a '< div >'for this, and the 'id=' of the div is going to be ' "map" '. When we write our code in our script block, we will reference that div. We're now going to use a function within the mapping library to create a new map. And so, I'm going to use 'new mapboxgl.Map()'. And this creates, this function, creates a new map, and that function takes an object that has a certain number of properties.

So, the first one is 'container:'. And here is where we reference that < div id="map" >. And so, I'm going to enter that here. Next, I'm going to set the 'style:' and this is defined by the library and the styles makers, which in this case is 'mapbox'. So, I'm going to go ahead and paste that in here. Next, we're going to set the starting location, which is going to be the longitude and latitude and this is one thing to watch. Depending on which library you're using you could get latitude first or longitude first. And then the last thing that we're going to set is the initial 'zoom:' on the page. So, in this case, we're going to set the initial 'zoom:' at '10'.

And we will change these in a moment once we load the map but there's one more thing that we need. The additional thing that we need is that we need an access token and here I will refer you to account.mapbox.com for you to get your own access token I will not share mind because since these videos are viewed by many, they would stop working soon after the 50,000 limit per month is reached. So, I'm going to go ahead and paste my token here and now, let's go ahead and load up the map. So, I'm going to go ahead and reload this page and as you can see there, we are viewing the city of Boston. If I was to zoom in, let's say we use 15 here.

I'm going to go ahead and reload the page. Then you could see that we are zoomed in to Cambridge and we are on MIT's campus. I could change the styles as well. Let's say we go with a dark style instead of the default one that I'm using here and let me go ahead and remove all of it and paste the new style. And now, I'm going to go ahead and reload the page and you'll see there that now the styles are very different. So, this is an example of basic mapping. If you use the Google Maps API, you would do something similar as I mentioned as well, Bing would do it as well, and many others.

You can see here some of the elements are bringing in the library, bringing in the styles, setting, some of the styles around the element that you're going to set. We talked about margin and padding and other things and then creating the function or using the function that is going to create the map, giving it the element within the page, in this case, the < div id="map" >, that where you're going to position your map. And then some of the initial settings as well, which are the level of zoom, the location and latitude and longitude, the style that you're going to use, and in this case, the container. So, give it a try. This is something that is very useful in many different contexts and as you can see, you can control this programmatically.

Video 8 – Map Markers

As in our previous example, we have loaded the JavaScript libraries for mapbox. We have done the styles as well and then we have set some styling for the page. The body doesn't have any margin or padding. And then we have said the following styles for the element that is going to hold a map. And you can see the element here, which is < div id="map" >. We then set our access token. And once again here you would have to get that access token yourself. And you can do so by navigating to mapbox and getting your own personal access token, we then create a new map.

And you can see here the parameters that we're passing in and the object which are the container: 'map', the tag that is going to be held, in this case, the '< div id="map" >', then the styles, then the 'center:' of the map, which is, as you can see here. The last thing is the 'zoom:' the initial zoom. And so, that is what we did before, you can see the map here on the right-hand side in the browser. And now what we're going to do is that we're going to set up, or we're going to add to this map a 'marker'. And we're going to use it, we're going to do it using a function from the mapbox library. And so, we will enter here 'mapboxgl.Marker( )' that's the function.

Then we will, in addition to that, set the longitude and latitude. So, let's '.setLngLat( )' and then the location that we're going to set is going to be the following. Let me go ahead and enter that in. After that, we're going to add it to the map '.addTo(map);' and now let's go ahead and reload the page on the right-hand side, and we should get a marker around here. Let's go ahead and see. And yes, you can see there that we did in fact get a marker. And if we zoom in a bit on the location, you can see there that that is the MIT campus. So, to recap, we created a map like we had done before and in addition to that, we added a marker which indicates a location, in this case, the MIT campus.

Video 9 – Map Clustering

Often, you will have more data than you can present to the user without it being overwhelming. One of the ways you can do that is by using clustering. As you can see here, I have the beginnings of a map and in line 7 and 8, I'm bringing in the standard library and styles. In line 17, in line 15 through 18th, I'm bringing in additional resources to be able to do clustering. We will be using data from the City of Chicago, crime data, crime reports. Here is the top of that file and it has some description as to what's in it. And if we navigate all the way to the end, you can see that there are a lot of crime reports, well over 20,000. And each one of them has a lot of details including latitude and longitude.

And so, we will use that to be able to add points to our clustering approach on our map. Let's go ahead and take a look at how that looks on the page. I'm going to go ahead and open up the developer tools. And then I will enter here, 'chicago'. And as you can see there, we have data inside of 'chicago', over 25,000 crime reports in this case and I can get the data length as well. How many records do I have? Now, let's go ahead and go back to our code. I'm going to start off by creating a 'map', and you can see there what I have entered, I am referencing the '< div >' that has the identifier of 'map'. I am then setting the latitude and longitude as well as initial zoom, and I'm setting styles as well.

Next, I'm going to create a cluster group. Let me go ahead and align that well. You can see there that I am creating a 'MarkerClusterGroup( );'. I am then referencing the 'chicago.data;', as you can see there, and assigning it to a variable called 'data'. I'm then using the 'length;' property to be able to get the number of records that we have within data. I'm going to be using that on a for loop in a minute. Now, let's go ahead and write that for loop. We're going to start off by '(let i=0;)'. This will be our starting condition. Then we will execute, we will loop as long as 'i 00:02:41,780 and on each iteration, we're going to increase our counter, which is 'i++)'. Let's go ahead and scroll up so we can have a little bit more room.

And let's start off here by getting a handle on the record and we will call that record 'a' and so, that will be 'chicago.data' plus the '[i];' for this iteration. We're next going to access the 'title' and we will pull data from the 'data', in field '[13];'. You can review the data yourself to see if I am using the right field. And in this case, I just realized that I should be using 'a' and not 'data'. Next, I'm

going to be creating a 'marker'. And as you can see here, this marker is different than the markers we have done before. This is part of the clustering library. And you can see there that I start off by setting the 'L.LatLng', then I set an 'icon:', and as part of that, I add a 'title:'. Now, let's go ahead and finish the loop by setting.

Let me go ahead and get my alignment right here. By setting the 'marker', and we're '.bindPopup' to the '(title);' and then we're adding a layer to markers with the marker that we have just created. Let's go ahead and finish up that loop. And the last step that we're going to write is to add, to 'map.addLayer( )'. And in this layer, we're going to add the markers that we just built. So, this is '(markers)'. Remember, this is what we have been working on here, 'markers' and we created up here. Now, let's go ahead and close our developer tools and reload this page and see if we get our clustering. Now, on at first try, it did not work for me and as you can see here, the reason is that I had not added my 'access token'.

And as mentioned before, you will need to get your own and I have just added mine. So, let me go ahead and reload the page and as you can see here, we have clustering on the page. And you can see the clusters of crime within the city of Chicago. You can see here, for example, that this cluster has low crime in comparison to say this one that has over 900 crime reports and if we scroll down, you can see some here that have very high numbers. This one is at over a 1,000. So, as you can see, this is a lot easier to parse visually than if we just simply put a lot of dots on the page. That can communicate it as well, but you have to give it some more thought. And in this case, clustering gives us a pretty good sense of what is taking place and what the numbers are throughout the neighborhoods.

Video 10 – Heat Maps

Let's create a map, then construct some data. In this case, we're going to be building GeoJSON data, and then, let's add additional layers to the map. We'll start off with a very basic map, as you can see here, and then we will construct some GeoJSON. Let me go ahead and show you the structure for GeoJSON. As you might imagine, there are a multitude of ways in which you can represent geospatial data. GeoJSON is one of the most widely supported standards. As you can see here, you can define a ' "Feature" ' with some ' "geometry" ' and some ' "properties" '. And if you build this, as we will do, then you can add this to packages that know how to ingest GeoJSON. So, let's go ahead and start off by taking a look at what this map looks like.

And I'm going to drag and drop the file that we're working on. And as you can see there, we have a map of the city of Chicago with a dark style. Next, we're going to listen for an event on 'map.on( )'. The event that we're going to listen to is when the map has been loaded. So, we enter ' 'load' ', and then we have a callback, which is a 'function( )' and inside of that function, we are going to do a few things. As I mentioned at first, the very first thing that we're going to do is that we're going to build some data in the GeoJSON format. Then we're going to add that data to the map and then we will add some additional layers.

So, let's go ahead and get started. And we will start by creating an array to hold the 'crimes' data. And we're working once again with the City of Chicago crimes data. This is the collection of over 20,000 crime reports that has latitude and longitude information in it. So, this is going to be our array of crimes. And then we're going to access the data from the City of Chicago, as mentioned. And we're going to use a '.forEach( )' to move through all of those crime reports. And inside of it, we're going to use the fat arrow syntax notation and we're going to be or each of those crime reports we're going to call a 'crime, ', singular and we're going to have a counter that is giving us the position within that array.

Then we will use the '=>' notation. And I will enter a return, and that's the body that will move through each of those items. And inside of it, I am going to push into the crimes. Now, 'crimes', plural. Array each of these new objects that we're going to create. So, 'crimes.push{ }', and inside of it, we're going to create an object and that object is going to have the following. A 'type: 'Feature', '. Then I'm going to have 'properties:'. And inside of those properties, I am going to have a 'dbh'. And this is a dbh that is meaningful for the map marks platform and relates to the intensity of the data when it comes to its severity in this case. And I'm not going to set that. I will set that for you.

So, for now, I will simply note that this is randomly selected intensity for this one feature. So, I'm going to enter here 'dbh' and then I'm going to use a random value. I have selected to have the intensity varying from 0 to '60'. And then I'm going to simply select a random value. So, I'm going to use the 'Math' library than '.random( )', and I'm going to call the 'random( )' function. And so that will return a random number between 0 and 1 and I'm going to multiply that times 60 to get a variation between 0 and 60. So, that will be the value of 'dbh:'. Next, I'm going to enter 'geometry:' and remember, all of this comes from the JSON standard. And I'm going to enter a couple of properties and an object.

The first one is going to be the 'type:'. And then, I'm going to specify ' "Point", ' here. Next, I'm going to enter the 'coordinates:'. And I'm going to provide values that are based on the latitude and longitude within the Chicago crime data. So, this is something we've done before and I will not go into as much detail as to what it is. But, it is the position within the crime report data for the latitude and longitude. So, let me go ahead and enter here the second value, and this will be '[26]]'. So, that looks pretty good. I am creating new objects that are compliant with the GeoJSON standard, and they're going to be added to the crimes array and let's go ahead and review here GeoJSON.

So, what I have there on the page looks fine. However, if you note, it is requiring double-quotes as supposed to single quotes. And in order to be compliant with the standard, I'm going to go ahead and update my property names, for example, that I had not quoted and I will do it now and match that format a 100%. So, I could do it this way, but I'm going to choose to be a 100% compliant with the standard and do double-quotes on everything including the property names. Okay, as you can see there, I have replaced all of the single quotes and I have made sure that all

of the property names have the double-quotes. And now, we're going to put all of that into a FeatureCollection.

So, you can see here that is part of the specification. And I'm going to go ahead and create that collection now. I'm going to call it 'geojson'. And the object is going to be of ' "type" = "FeatureCollection", '. And the ' "features": ' are going to be 'crimes', which is the big array that we have just built. And this should be, of course, quotes, I mean colon. We're now going to add all of this data to the map. And we do so by using 'map.addSource'. Then identifying the type of data that we have, in this case, it will be ' 'crime' ', and then entering an object with a ' 'type' :'. In this case, is ' 'geojson' '. And the ' 'data' :' that we are entering. The data will be our 'geojson' that we have just built.

So, to recap, at this point, we have constructed the data in the JSON format, and then we have added that data, that JSON data, that GeoJSON, to the map. Next, we're going to add a layer to the map, and within it, we're going to specify how to layout the data. This part tends to be pretty, pretty vendor-specific. So, if you look at Google, if you look at Mapbox, if you look at other internet mapping platforms, they will do it slightly different. So, we will now build it from scratch. Instead, I will paste it onto the code here, and I will make some comments about what it is, what is being done. As you can see here, the very first part, give it an identifier, then sets the type of map that we will be creating, which, in this case, is a ' 'heatmap' ,'.

You can see there the data source in some of the magnification settings for it. Now, there is some setting here for the type of weight that you will be showing within the ' 'heatmap' ,'. And here is a place where I will give to you as an exercise to try out on your own, to come up with a different way of setting the intensity of the map. As you will remember, when we were writing the code, we simply wrote a random value for the type of intensity of the crime. And you might come up with a different type of waiting. Say, for example, you thought that breaking and entering was the most important type of crime. You could decrease the value of the others, remember, it goes from 0 to 60.

Let's make the rest of them one, say, and simply look at the, at this specific one and give it a high-intensity. And so, I will let you play with that part and see what type of analysis you could do. This is working with data, working with geospatial information, and working within the web browser to be able to have that type of UI. That could say, for example, support a police force to do better policing or to understand the types of crimes that are being committed within a certain jurisdiction for government officials. You can see here the rest of the representation for that layer, and you can read the documentation to get a better understanding. So, in this case, once we've added that information, let's go ahead and go back to the heatmap.

And let's go ahead and reload that map and as you can see here, we get now an overlay of a heatmap. And in this case, although the values are randomly distributed, you can see where there are more crime reports than other parts. If you go back to the example where there was clustering being made, you can compare and see how those overlay and the overlays are pretty close. So,

as you can see here, heatmaps can be a powerful way to convey information. There's a great deal of richness when it comes to this platform as to how you create that geometry and that visualization. The same thing is true for many of the other mapping platforms. So, as mentioned, I will leave it to you as an exercise to dig out the type of data that you want to visualize to provide the right way, weightings when it comes to that data, and to see how your map changes based on your decisions.

Video 11 – Map Animation

Tracking or showing an animation on a map turns out to be pretty useful to convey information to the user. In this case, we're going to add a button to the map. And then we're going to move a marker through a number of bus stops between MIT and Harvard. There is a bus that moves between these two institutions. So, let's go ahead and get started. As you can see here, I have the basic boilerplate for a map. But in this case, we're going to add a few more things. We're going to start off by creating a style called '.map-overlay', where we're going to set the 'position:' to be 'absolute;'. Then we're going to set 'left: 0;'. Then we're going to set our 'padding: 10px;' that is.

Next, we're going to write our '< div >' that will hold the button. And as you can see there, I have the '< div >' for the ' "map" ', but in this, this one is going to be for the button. So, let's go ahead and write our '< div >'. And inside of it, we're going to have a '< button >'. And inside of that button, we're going to show the following, 'Show stops between MIT and Harvard'. We'll go ahead and add the onclick event. And we're going to write a function afterwards, called ' "move()" ' to move that marker. We'll also set some styles here for the font, to have it nice and big. And I will set the ' "font-size: 2em;" '.

And lastly, I'm going to place the 'class' map overlay to the, to the '< div >', that's holding the button. So, this will be ' "map-overlay" '. And then, I'm also going to add the ' "top" ' class. This is part of the styles that Mapbox defines. And at this point, if we were to reload the page, we'd see that we have that big button that says, Show stops between MIT and Harvard. Next, I'm going to scroll up and start to work on the script tags. And inside of here, I'm going to set my access token as well as my map. So, let me go ahead and do that. And now, I'm going to go ahead and reload the page. And as you can see there, I have the map in Cambridge.

Here is MIT, Massachusetts Institute of Technology. And up here is where Harvard is. And the bus moves through Massachusetts Avenue that starts here and will move along this route all the way to the Harvard campus. Next, let's scroll up and add a marker to the map. I'm going to go ahead and add that code here. And as you can see there, we are adding a new marker, a latitude and longitude there, that is specified on line 40, and then we're adding that to the map. So, let's go ahead and reload the page. And as you can see there we have our marker there at the beginning of the MIT campus. And next, we're going to add a number of latitude and longitude coordinates that are, that correspond to each of the bus stuffs.

And simply to illustrate the point, if I was to scroll in pretty close. You'd see the one that is on the MIT campus. Here is a bus stop. Let me go ahead and see. Here is the 77 Massachusetts Avenue bus stop. And I have collected the coordinates for the rest of the bus stops along the way. And I'll go ahead and reload the page so we can see the full map. And now, I'm going to go ahead and paste those coordinates. And as you can see there, we have a number of bus stops that takes us all the way from one campus to the other. Now, it's time to write or move function. And we're going to start off by creating a variable for a counter.

And I'm going to initialize that at '0;'. And then I'm going to write the 'move()' function. 'move()', then the body. And inside of it, I'm going to have a timer. The timer is going to be 'setTimeout'. And then inside of it, I'm going to use the '=>' notation. And now, we're ready to start writing the body of that timer. We're going to start off here by checking to see where we are within that array of coordinates. And so, we will check the 'counter' to make sure that it's '>=' than the 'busStops.length'. And if it is, then that means we've gone through all of the coordinates, all of the positions, and we will return.

Otherwise, we will move the 'marker' to the new latitude and longitude. So, 'setLngLat', and then the 'busStops', and the specific '[counter]' position that we're in. Then we will increase, increase our counter. So, next time, this timer is called, you advance to the next position. And then we call ourselves again to have a repeating function that is executing. And the last thing that we want to do is that we want to set a time increment for the counter. So, let's go ahead and reload the page. Make sure we have the new code. And now, I'm going to click on the big button that says, Show stops.

And if things are working well as they are, you can see there the marker moving and stopping at each of the bus stops in-between the two campus. So, you can imagine this being used to illustrate a certain route, but you could also be in continuous communication with a server, extracting those coordinates and updating a UI to show where a bus is. You could do that for many things as well. And if you've used modern applications that show tracking for shuttles or for buses or for trains. This is very much the mechanism that is being used. To practice find a source of data that has latitudes and longitudes along a path that you want to illustrate or look for live data where you could extract that information and show it on the map.

Video 12 – Real-Time Bus Tracker

In this exercise, you will be creating a bus tracker, a real-time bus tracker. The data will be from the MBTA, that's Massachusetts Bay Transportation Authority. And as you can see, this is their API portal. There are the steps to become a registered developer, which you might want to try for practice. Now, note that in order to experiment, you don't need an API key. And so, in this case, we will be looking at a very specific route, a bus called bus one, that travels between the MIT campus and harvests, Harvard's campus. The URL that we will use is the following. You can see

there we are hitting the mbta.com/vehicles, that is the source of the information. Then we're filtering for bus route 1, which is the bus I mentioned, the one that travels between the two schools.

And we are including the trip information as well. So, if we navigate to a browser and paste in that link, you will see the current at this specific second data for those buses. And we don't know how many buses there are. We don't know what they're doing if they're running or if they're stopped. But all of that data is captured within this file. So, if we were to look, say, for latitude and longitude. You can see there that there are a number of buses in the latitude and longitude is being captured by this document, which you can download and visualize within a mapping platform, which will be your task for this exercise.

I will get you started by writing the code that periodically queries the service to pull down the latest data. So, let's close these tabs and write some code. We're going to start off by writing the function that will reach out and pull down that data programmatically. We're going to use an 'async function', so we can use fetch. We're going to call this one 'getBusLocations()'. Then we're going to write the body of the function. Inside of it, we're going to have the URL that we just used. And so, let's go ahead and paste that there. The next thing that we're going to do is that we're going to make that call and catch that in a 'response'.

And we're going to wait for 'fetch' to complete. That is for all of that data to come back. And what we're passing to 'fetch' is the '(url);'. We're then going to extract the data that is within that response. And we are also going to wait for that to be extracted. So, we will write 'response.json();'. And then we will return this data to the calling function. So, that will be 'json.data;'. We're not going to write the function that we will call run simply to pull, 'getBusLocations()'. This will be 'async function' as well since we will wait for the response from the request to get to get the buses.

And inside of it, we're going to call the function that we just wrote. And I will set that and 'locations', and we will wait for it to complete. So, this is 'getBusLocations()'. And I see that I've missed out a letter here on 'async'. This is why I get the red underline. And now, I'm going to write to the console simply a date stamp. This is good practice when you're trying to track how frequently something is being used. And I will simply write the 'Date()'. Then I will write the '(locations);' to the screen. And next, we're going to create a 'timer'. That timer is simply going to be 'setTimeout()'. And we're going to call ourselves, our own function to be able to once again call 'getBusLocations'.

And I'm going to call this every '15' seconds. And this is important for you to note. If you had one of these services too frequently, you're going to get banned. And the reason for that is that you could hit it every fraction of a second, and millions of hits, you know, per seconds, potentially. And that would cause an overload on the system. And so typically, if you space it out about every 15 seconds, but it depends on the service. That should be good to show that you're not trying to take the service down. So, in this case, as you can see here, we've written two functions. The very first one reaches out to MBTA with the URL that we showed, which gets the bus data for route one.

And then we have another function that polls this, the 'getBusLocations' function every 15 seconds. You can see there the call. Then we're writing a 'Date()' stamp, then we write the 'locations'. And now, let's take a look and see if this is running. We'll go ahead and reload the page. Let me make my font a little bit bigger. And I'm going to go ahead and reload the page here, and nothing is happening. And that is because I have not called run, which I need to do to start this process. So, I'm going to go ahead and write 'run()' here. And then I'm going to go ahead and reload the page once more. And as you can see, they're very, very quickly, we get the first response.

And you can see here that that was at 15: 02: 38. And in 15 seconds we should get the next one. I'll go ahead and wait for it before I start looking. Here is the next one, at 15: 02: 53. And if I expand this, you can see there the different parts of the data that are coming back in this JSON file. And if I expand further and I look at 'attributes:', you can start to see here "IN_TRANSIT_TO" to the number of seats that are available. But more importantly, the latitude and longitude which you will need to extract in order to be able to place on the map.

So, in this exercise, you're going to bring everything together, your knowledge of maps, your knowledge of timers with the map, your knowledge of being able to animate and track data on the map. And you will do so with live data from the MBTA. If for some reason you happen to get banned, and you can't get back on. You can use data from another city. Many cities around the world are now sharing this information real-time with developers. And so, you can make use of that data to be able to create your bus tracker.