So now that we've learned a little bit about what application programming interfaces are, let's go ahead and interact with a real one and get some data back from Open Weather Map.

So in order to fetch data from the Internet, there's a really handy cookbook document on how to do this.

And it takes you step by step through how to use the http package.

And this is a Flutter package that has been created by the Dart team.

So you know it's good and it's really simple to use as well.

So we're going to be using this http library to make our requests to Open Weather Map to grab the current weather data.

Let's take a look at how we would use this library.

So as always we're going to add it as a dependency to our project.

So we're going to open up our pubspec.yaml and we're going to add another dependency under geolocator and then we're going to hit packages.get to incorporate all the code from the library into our project. And now all we have to do is import the library.

So let's go ahead and do that in our loading screen.

Right under where we've imported our location.dart, I'm gonna import the http library and it's the one that's http/http.dart which is what they say we should import here. And now I can start using some code from that library. In order to make a network request, all we have to do is to call the get method from the http library and then we in the URL that we want to fetch data from. So let's create a new method here and I'm gonna call it getData. Now inside this method getData, I'm going to use that get method that we now have access to because we imported our http.dart package.

So this gets method expects a URL as one of the inputs, and it will try to grab the data from that URL and it returns a future response.

So this is a asynchronous method because it can take any amount of time to go through all the wires and talk to Open Weather Map servers,so this method might take a while.

So that's why it's going to work asynchronously. The URL that I'm going to pass in as the input to this get method is going to come from Open Weather Map.

So if you scroll down on the API docs for current weather data, you can see that you can also fetch the weather data by geographic coordinates. So you can pass in the location of your interest by adding a lat value and a lon value.

And here's an example of an API that they've already structured for us.

So if you click on this example API call, you can see that it's tapping into something called samples.openweathermap.org and it's passing in a latitude of 35 and a longitude of 139.

And it's also got some sort of app ID in here.

Now we're going to register our own app ID a little bit later on as well so that we can make requests to any latitude and longitude so that we don't have to use the samples subdomain.

But for now all we want is to be able to make sure that our app can actually make this request to this URL so that we can get back the corresponding data in the same way that our browser is able to hit up that URL and make a get request for this data.

So all we're going to do is we're going to copy this URL from the URL bar and we're going to paste it into our getMethod as a string,so with some single quotes around it.

Now this get method remember returns a future response right?

So that means if we wanted to hold on to the data that we get back, we have to create a new variable that is of type response and we can call it anything we want. We can call it the response if you want.

And now notice that this future response can't be assigned to type response.

So because this method is going to work asynchronously, it might resolve at any time in the future.

So if we wanted to use this response somewhere, say for example printing it into the console, then we have to turn or get data into an async method just as we have done in other places. And then also add that await keyword in front of the get method so that we wait for this to resolve before we try to print the response. Now at this stage if we go ahead and trigger our code getData, we can either added into the initState, getData, or if you don't want to have to hot restart your app every time you can also put it inside the build method which we know gets called every single time we hit save or when we hit hot reload.

So either way is fine.

I'm going to put it in here because we're going to be testing it quite a lot.

And this is quite an easy way of re-triggering this method, getData.

So now if we run our app and we take a look inside the console,you can see that we're getting an instance of response printed.

Now what exactly is a response object?

Let's take a look at the http package docs to see what that object actually looks like.

So if we go into the About section you can see there's an API reference. And this takes us to the documentation for the http package.

Now here let's go ahead and search for our response class that we're using.

And you can see that this is an http response where the entire response body is known in advance.

So this is the response that we're getting back. When we create a new response object, it has a number of properties. It has a body which is the body of the response as a string,and it's also got things such as headers or request or status code. The part that we're actually interested in, the part that contains the data, is actually held in the body of the response.

So let's print response.body instead.

So now let's hit run again.

And you can see that we're getting back exactly the same data as we're seeing over here.

So here is the response body.

Now there's also this thing called status code which we can access as a property on the response object.

And this gives us the statusCode of the response.

So if we print that out instead, you can see that what we're getting back is the code 200.

So what are status codes?

Well, when we're interacting with an external server, they need a short and unified way of telling us what exactly happened when we interacted with them.

It's kind of like a code for a particular outcome.

Now we've all seen 404 pages right?

So there's some famous ones from Pixar or from Lego or from various designers on Dribbble. And essentially a 404 status code means that the resource that you tried to access from the external server doesn't actually exist.

So there's a whole bunch of different codes that can be sent back and anything that's in the 200 means that your request was successful. And specifically, the code

200 means OK, everything is a OK and here's your result.

Anything in the 404 means that something bad happened when you tried to make a request. And anything in the 500 means that there's actually an issue with their server.

So I'll include a link to this website where it talks in detail about what each of these codes mean.

And you can refer to it when you're trying to figure out what a statusCode that you're getting back actually is trying to tell you.

But there's actually this cheat sheet by Sander Hoogendoorn in one of the talks that he gave which I think is fantastic.

Where it tells you that any code that comes back that starts with a one means

hold on, wait, something's happening.

Anything that comes back with a two means here you go when successfully. Anything that starts with a three means go away, you are not authorized to view this particular resource.

Anything that starts with a four means that you screwed up. Anything that starts with a five means I screwed up, I being that external server who's sending you that message.

So for example let's try and fetch a resource that doesn't exist.

Instead of trying to get something like weather, let's you know try to get something like heather. And this will now refer to something that the API doesn't actually know how to handle.

So now if we hit save again and we rerun our get data, you'll see that what's printed is our classic 404 as the status code. So it's usually a good idea to check what the status code is before we start presuming that the response.body is what we think it is.

Because right now if I tried to tap into the response.body and do something with it, then you can see that what I'm actually getting is just nothing.

I'm not getting anything back as the body.

And if I tried to render that in my app to display it, then it's also going to be an unexpected error.

Let's instead go ahead and check to see if the response.statusCode is equal to 200 which means that everything went perfectly.

Then in this case, we're going to create a new string that's called data and it's going to be set to response.body.

But if their response status code was not 200, then else we're going to simply print out the response.statusCode just to see what actually went wrong.

Now at this point, there's one thing that you might realize which is that our get method looks very weird.

It doesn't look like it came from a package right?

Because usually when we're using an external package, we're having to tap into some sort of object that's created in the package.

But if we take a look at this get method and where it lives, which is in the http.dart file.

So let's hold down COMMAND and click on get, and we land on our http.dart file which we've incorporated into our loading screen right here.

Then you can see that here there's actually no classes that have been created.

There's no http class.

Instead, it's giving us access to a whole bunch of methods, get method or the head method or the post method.

And this makes our code look almost like as if we created a method called get somewhere and we're using our own. Which is kind of confusing if somebody else came along and they didn't realize that this actually comes from the http package.

So one of the most common ways of using this http.dart package in Flutter code is to add the keyword 'as' and give that package a name, usually it'll be http.

So now we can use anything that's inside this package by using that word http.

So now instead of just get, we can write http.get and instead of response, we can write http.response because all of these things came from that package.

So now the next time you look at this code or somebody else looks at this code, it's quite obvious where this get method comes from, it comes from that http package, and where this object response comes from,it also comes from the http package. Because if you didn't know that and you try to figure out what that response object is, had we just gone on to the Flutter docs and we try to search for this thing called response, it doesn't actually exist here right?

It's not the same as the documentation for that http package because here is where that response object comes from.

So now if we print out that data again and hit save to view it in the console, you can see that this is structured as a string but it's more than that. It's actually formatted in a way for you to be able to tap into certain things based on a key value pair. And this is what we would call a JSON formatted API response. And in the next lesson we're going to learn a bit more about what that actually means and how to use it.

