Now in the last lesson we implemented the geolocator package to be able to fetch our current location both for Android and iOS.

And the code is super simple.

It's basically a single line that fetches us the current location of the phone depending on which location accuracy we want.

However there are two key words that we're using in this code that comes from the documentation for the package that we've never really seen before.

So in order to understand this, we have to first learn about asynchronous programming and why it's useful.

Now when we're writing code, there are lines of code which take a very short amount of time to execute.

For example if we wanted to run a print statement, it's a single line statement prints the word hello world into the console.

And this takes fractions of a second to complete.

But sometimes there are things that might take more time.

For example if we wanted to load a large image from the Internet. Even if we're doing it in a browser and if your internet is slightly slower, you might see this painful loading of the image where it takes absolutely forever.

And this is because there's a lot of data that we're trying to transfer through cables under the sea literally trying to get data. And the larger the amount of data the longer the time it takes for it to load. Now consider if we made this function. It's a function that returns nothing. So it returns void but, it has a body that is composed of three steps or three things that it tries to do.

And the first thing is to simply print 'Hello moon' to the console.

This is pretty quick to execute.

So we see the result pretty much immediately.

Now the next thing is to load an image from NASA of the moon. And I don't know how often you've done this but they're huge and that can take a long time.

But because our function goes from top to bottom, our third Step can't execute until the second step is done.

So it's only once this image is loaded that the computer will actually run the very last statement, which is to print the word 'Hello Jupiter' into the console.

Now this behavior is what we would call synchronous.

Everything happens in synchrony.

Step 1 happens, then step 2 happens and only when step 2 has concluded or has finished and we've gotten the image back does step 3 actually execute.

Now what would it look like if this was asynchronous?

Well step 1 would execute as usual no changes there. But step 2, let's say that we're now loading our image from NASA asynchronously.

Well in this case while the image is loading and we're trying to wait on that data to come back, we can already execute step 3. And we don't have to wait for this to finish before we continue in our code.

Now why is this actually useful?

I want you to imagine that come Monday you're going to work and your boss tells you to get me 100 passport numbers.

Get them to me right now.

So you have to call up all of the customers one by one and ask them what is your passport number. So you have this checklist and you're going from the top to the bottom one by one calling up each customer.

Now it's not so straightforward right?

Very few people remember their passport numbers off the top of their head.

So you have to start the call ask them for their phone number and then they have to rummage around their house and to actually find their passport to be able to give you that passport number.

And this can take anywhere from a couple of seconds to a couple of minutes depending on how well they've stashed away their passport.

So what are you doing while all of that is happening? Let's represent this process using a graph. On the horizontal axis is time and we have a couple of units of time, you can imagine each square as a second.

Now let's say that it only takes you maybe a second to ask that customer, 'Hey can you give me your passport number?'

So your job is kind of done now.

The next step is waiting for the user to look for their passport and give you that number. And all of that time you're just sitting there twiddling their thumbs on hold on the call. And you're not really being productive right now right?

But once they've given you their passport number you can finally spring into action again and you can go and call the next customer, and then the same process repeats on and on again until you've gotten to the end of that list and you've reached 100 customers. And nobody likes being on hold waiting on a call, right?

And if that is the majority of your day then it's not going to feel great.

Now you might have realized that there's a different way of doing this.

You could in fact have just drafted individual e-mails, say you wanted to keep it personal to be able to persuade your customers to give you the passport numbers, and you could have simply sent these e-mails off to all of your customers one by one. Now afterwards this leaves you free to do any other tasks that you need to do.

And as the replies come in, you can respond to them. So you no longer have to be there on the phone waiting for their response.

Now if we were to represent this in the same graphical style, then your day might look more like this.

You send each customer an individual email but it doesn't really matter how long it takes them to respond because you're able to continue onto the next task,send the next e-mail or do the next thing and they can take their time finding their passport and only once they have their passport number do they respond to you and you get that email response back. And it's in that moment that you can actually act on it and you don't have to be on hold waiting for them to fetch the information you need. So now your day looks more like this.

You're using every chunk of the day doing something productive and you're responding to the e-mails that you've sent out as and when the responses come in.

So compare these two side by side where in one case you're doing something, waiting, doing something, waiting,doing the next thing and waiting.

Well this is synchronous programming. Whereas in the other case you're basically initiating a task and only once that task is complete do you get notified and act on it.

So this means that we're able to use only five units of time instead of nine units of time to achieve the same thing.

This is the difference between synchronous and asynchronous programming.

Now notice how this is all being done by one worker, you. It's not that you have 10 people trying to get 100 passport numbers from 100 customers. It's actually still one person doing that task but it's just done more efficiently because you're letting the customer take their time finding those numbers and only getting your attention once they have the results.

So let's build out a real example and take a look at what it would look like if we were to write it in code. Inside your clima project, you're going to right click and you're going to create a new Dart file. And we're going to call this file simply just a scratch file.

So it's going to be scratch.dart and it's going to be created inside your main project right here.

Now inside this file, this is completely separate from our project, and it lets us test out some Dart code.

But unlike DartPad, we're able to tap in to all of the available Flutter and Dart libraries.

Now if you head over to this link which will be in the course resources for this module, then you can see that there's a little bit of code that I've already written as some starting code for you.

So go ahead and copy all of it and paste it into your scratch.dart.

So all that we have here are three functions, task 1, task 2 and task 3. And they're called in sequence inside a function called performTasks. And then in the main function which is our entry point to this particular file, we call performTasks.

So right now, you can run this scratch.dart without running any of the rest of our code by simply right clicking on the file and clicking on run scratch.dart. And you should see your console pop up and you can see that this is an isolated file that you can run without having to take all the time to load up the UI and build it onto an app.

So this is where we're going to experiment with our Dart code.

And you can see that as expected, all that happens is we print T'ask 1 is complete', 'Task 2 is complete',

Task 3 is complete.'

But the important thing to notice is that they happen in sequence because inside our method perform tasks we said to do 1 then 2 then 3, in this order.

These functions are getting called sequentially.

Now let's say that task 2 was actually a really time consuming task, like downloading a picture of the moon off the NASA's website.

Well let's simulate something taking a long time.

We can create a new duration object called threeSeconds.

And this is going to be equal to a new duration object.

And here we can specify the amount of time so microseconds, milliseconds, seconds, minutes, hours, days.

I don't want you guys have to watch this video for days to be to see this example.

So we're going to keep it short to maybe let's say a delay of three seconds right?

So now to be able to enact this three second duration delay, we're going to use a function called sleep.

Now the important thing about sleep is that this is a synchronous operation.

So we will have to finish before the next line can be executed.

And it takes a single input which has a duration object and we can simply put our threeSeconds in here as the amount of time that we want to pause for.

So when we call task 2, it will set a duration of three seconds and it will sleep our program for three seconds. And then and only then, do we get to print 'Task 2 complete' into the console. And because task 3 comes after task 2, then theoretically it should also come after three seconds delay.

So now let's go ahead and click run again.

So notice scratch.dart is currently selected as the thing that we're going to run.

And now you can see that task 1 complete, 1 2 3, task 2 task three. So if you're watching this video on double speed or half speed might be a good time to actually put it back to normal speed because I'm going to try and show you what actually happens with these delays.

Let's just run that again and notice Task 1, 1 2 3, Task 2 Task 3.

So we know that each of these tasks printing a single string into the console basically takes a negligible amount of time. All that's causing the delay is this artificial delay that I've put in here using the sleep method which comes from dart:io. Now because there are a lot of cases where things can take an unpredictable but usually a long period of time to complete such as downloading something or reading something from a file or even doing complex computations,it all takes a large amount of time.

And what we don't want to happen is for everything else, every subsequent task, to be held off until this task is complete.

So what if we could say that, you know what? Let task 2 do its thing in the background but continue forward and do the things that you can do right now.

And once that long task is completed well, then we'll deal with the results once that actually happens.

In this case we would be using an asynchronous method instead of a synchronous one.

So let's simulate a asynchronous delay.

We can do this by instead of using sleep,

we can use a method called Future.delayed. So it has two inputs,

one is a duration and the second is a computation to run after the delay.

This is what we need.

So the first input is going to be a three second delay and then I'm going to add a callback to specify what should happen after those three seconds are up. And it's namely creating this result and printing that task 2 is complete.

So I'm going to cut that and paste it into that.

Now the way that our code is going to run is this is a asynchronous method. And we know it's an asynchronous method because it returns something called a future.

So whenever you see a future in the documentation, you'll know that the method is something that will happen asynchronously. And it will delay by three seconds but it will allow other lines of the code to run if they can.

Now once these three seconds are up, then it triggers the computation in the callback.

And in this case it's to create the result, task 2 data and the print task 2 complete.

Now if we run our code again, let's watch the console and see how it works this time.

Remember that at the top here where we say perform tasks, I haven't changed that order at all.

It's still task 1 should perform, then task 2 then task 3 gets called.

But now let's see what happens down here.

Task 1 task 3 completed almost immediately, and then task 2 complete after a three second delay.

So this is asynchronous programming in action.

We saw that because task 2 took a long time.

We were waiting for it to happen but we continued. We plowed ahead with the things that we can do right now.

And once task 2 completed, it's then and only then, does it actually get executed and printed into the console now.

That's all very well and good for things like downloading images from the Internet or doing time intensive tasks.

But sometimes we need the results of the tasks to be able to continue to the next task.

So for example let's say that our task 3 actually required an input that comes from task 2.

So let's change task 3 to take a string input that's called task2Data. And then let's use that inside 3, so task 3 complete with and then we'll add all task 2 data inside this function.

Now task 3 relies on something that comes from task 2.

So that means we have to return something from task 2 to be able to do that.

So let's output a string from task 2 instead of nothing.

And we're going to output this result right here.

Let's create a new string up here called result.

And let's set it to nothing to begin with, but only after our asynchronous method has delayed by 3 seconds.

So let's say we're fetching some data from the Internet. Only once that's completed do we assign result to a value.

And finally at the very end do we actually return this result as the output of our task 2.

So now that task 2 has a output we can assign that to equal a new string called task2Result, going to equal to the output of task 2. And then we're going to use that inside task 3.

So task 2 result is going to be the input for task 3.

So the entire order of things is task 2 will execute asynchronously so it can take as long as it wants in the background.

Now once it's done though, it's going to assign some data to a string called result and then it's going to print that it's completed and also output result.

Now at this stage, we're going to take that result and bind it to a new string and then use that as the input for the next step which is inside task 3.

What do you think will happen given that we know that task 2 is asynchronous and it can complete at any time at once and our code will actually skip ahead to try and do the things that it can't right now?

So if we take a look at running our code, you can see that task 1 and task 3 complete almost immediately and task two is taking its three second delay to complete. But because task 3 relies on some data from task 2, so if it executes immediately then it won't actually have the information to hand.

And so that's why we're getting task 3 complete with null at this point because task2Data isn't available until the task 2 completes.

So how can we fix this?

Well we can make our code actually wait so we know that this is an asynchronous method and we allow other code to continue on and not have to wait for this to finish.

But in this case because task 3 relies on this second task to complete, what we can do is we can make our code wait for it to complete before we call the next task. To do this we can change this into a async method.

So this is called a modifier and we add it just before the curly brace.

By adding that async keyword, we can now have access to the keyword that's await.

And we're going to put it right in front of task 2.

So before we call task 2, we're going to wait for it to finish. Now inside task 2, we have to make some changes too. Instead of returning a string,we have to return something that's called a future.

Now a future is something that will exist in the future.

Right now it's nothing because our task hasn't completed. But once our task has completed then our future will be an actual thing like an actual string or an actual integer. Now we have to also mark this method as asynchronous to be able to say that we have to wait for this Future.delayed method to complete before we can output our results.

So now with this updated code with our async and await where we're waiting on the parts that need to complete in order to give an input to task 3,now if I hit play and run my code, you can see task 1 complete happens first. And then task 2 happens after a delay and then finally task 3 happens using the data that comes back after task 2 is complete.

Now we've talked a lot about futures when we were writing our code in the demo. What exactly are futures?

Well they're kind of similar to receipts.

Let me explain what I mean.

Let's say that you go to the coffee shop and you order a cup of coffee.

Now at this point you'll get a receipt with an order number. And you can go away and do something else or check Instagram or check Facebook while your coffee is made.

You don't want to stand there waiting for it to be done.

So then once the coffee is done, they'll shout for your order, 'Order number 1 please come and collect.'

And now you'll show up and you can actually claim your coffee based on your order receipt. That receipt is the same as a future in Dart.

It's not the actual thing that you want.

It's just a promise that you will get something in the future. If you come from JavaScript and you've heard of promises, then it's exactly the same.

It's something that doesn't exist right now but after it's completed its thing, after it's gotten the image or after it's downloaded the text, then your future will actually materialize into a real object like a real string or a real integer.

Now you can be more specific with your futures.

You can specify that this is going to be a future string or this is gonna be a future integer, and we do that by using an angle bracket.

So in this case we can change our date type from just a generic dynamic future to something that is a specific data type.

So it could be a future string.

So this means that we're expecting a future string when we call this method task 2. And right now if I decide to comment out this line and try to print what task 2 is equal to, then you can see that it's a instance of a future string.

It's not an actual string yet. It's only once task two is complete does that future string materialize and change into an actual string. Now asynchronous programming is not the easiest concept to grasp. But I hope through reviewing and re-reviewing this lesson and also completing the modules that are ahead of you, where we're going to be using async and await and futures in a lot more places, you're going to steadily get used to this idea of asynchronously doing something. Instead of doing everything in sequence which is synchronous programming,we're trying to be more efficient with our time or with the computer's time by trying to asynchronously do our tasks so that things don't necessarily happen in the order that we programmed it to, but we end up completing our tasks in the quickest way using the shortest route.

So this concept is going to come up again and again.

Don't worry if this hasn't completely settled in your mind yet. We're going to be reviewing this a lot more in the future. But in the next lesson we're going to get back to making our app and getting our location data as soon as we load up the screen.

So for all of that and more, I'll see you on the next lesson.