Now in previous lessons we already started seeing Dart constructors and these are the bits of code that we write inside our Dart classes in order to be able to create a object from the class.

So if our classes are the blueprints for our objects where we plan out what properties the eventual object will have, what methods it will be able to perform,then the constructor is the part of the code that actually turns this blueprint into the actual object.

So if it was the plans for a house, then the constructor would be the builders who actually take that and turn it into the actual house.

Now we already saw with our previous code when we created the human class, that our class can have properties.

And these are simply just the instance variables that live inside the class.

And then we've got our methods which are essentially another word for functions that are inside a class.

And when I say inside a class, it basically means that the eventual object that gets created from this class,so in this case it would be a human object,it would have these properties and it would be able to perform these methods.

So these methods and properties are actually associated with an object rather than just free-floating around.

Now the third thing that we've already seen as well is the constructor. And this has a special bit of syntax, but essentially it tells the class what we want to initialize the values of the properties such as in this case, the height of the human object to be.

So when it actually comes to creating the object from this class blueprint, then effectively we're going to be using that constructor to do it.

So we'll write some code that looks a bit like this.

We're creating an object that has the type of this class human, and the object is called Jenny and it's created.

So the right hand side of that equation is using this particular constructor here where we start out with the word human and then we give it an input of a starting height, which it's going to assign to that height property. When we create the constructor,it's simply just a bit of preplanning to tell so that when we eventually create the object, we can specify which properties we want to initialize with a value at the point of creating the object.

Now this part of how constructors work we've already seen. And we created our Jenny object or our James object using the Human class. We provided a initial value for our height property of this particular object. And then later on we could tap into that height property or change that height property if we needed to. But this height property is associated with this object and that's why we're able to get hold of it by writing the dot notation. But this height is different from this height as we can see here because they're associated with different objects.

And the reason why they have these starting values is not because we gave it a starting value in our class, because we could have done this right? And they would all start out with the value 20. But instead we specified the value only when we constructed that object using the constructor here. And this makes a lot of sense in a lot of cases not just because babies don't tend to be born with a default height,but in a game for example, the user might not have a username value right?

We only have access to that username value once the user actually inputs one.

So very often, you'll create classes that start out without a value. So height currently equals null.

And only when it initialize or when it's constructed, do we actually give it a value to assign to that property. Now by default, even if we didn't create a constructor let's say I commented that out,you also have a default constructor. So I could in fact say something like this right?

I could say that I'm creating a new Jenny object from the human class and Jenny will still have a height property as will James, but that height property will be equal to whatever this value will be, which currently is null.

So that's what we get printed here and we would have to modify that value using the object.

So we would have to say jenny.height is now equal to 15, in order to achieve the same thing.

And you can imagine if we had a lot of properties which we wanted an initial value for, then it would take a lot of time to do this using these two steps.

That's why constructors are useful.

And this is also why by default, Dart will give us a free constructor.

So at the moment in my human class, you can see there are no constructors. But by default, Dart will give you a constructor that looks something like this is. It's basically an empty constructor which doesn't take any values, any input.

And it also doesn't do anything other than just create the human object from the human class. We don't have to type that because it's always going to be there for us. Now at the moment,we have this constructor so let's just uncomment it to put it back into the code. And we're using this parameter called startingHeight, to be able to pass it in when we construct a new object from this class and we set our property or height to the value of that input, that's startingHeight.

So that's why we're able to write startingHeight, and give it a value.

And this just reverts our code to what we had before.

And so Jenny has a starting height of 15 and James' starting height of 20, based on these constructors.

But what if we didn't want to call it startingHeight?

What if it made more sense to just call it height?

The input is called height and the property is called height.

So then we would set height to equal height.

And when we initialize the human Jenny, we say that we're creating a new human with a height of 15.

So this part of the code reads more naturally right?

We're creating a new object of type Human,it's called Jenny and it's created from the human class with a height of 15, instead of this starting height business.

But now down here, it gets kind of confusing because well, which height is this one that you're setting and which height is this one that you're setting it to right?

Because we're trying to get the value of this one and assigning it to this one.

But is that this height or is that this height?

It's really confusing.

And in fact when you run it, the code gets confused too. Because we end up with null for both Jenny's height and James' height because the code thinks that this height is actually talking about this one.

And so is this one.

So that means that this input, this variable here, is now equal to 15.

It's basically saying 15 should be equal to 15.

And you can confirm that by simply just printing height over here.

So that when Jenny and James get constructed, we should be seeing 15 and 20.

So let's hit run and you can see that we're getting 15 printed when we create our Jenny object, and we're getting 20 printed when we're creating our James object.

But this particular height doesn't actually get any values assigned to it.

So this is not what we want.

And it also makes your code really hard to read.

So what's the way around this?

Well in Dart, we can use a keyword code this. So we can say this.height equals height. And if you come from other programming languages such as JavaScript or Java, you might have seen this as well.

But basically when we start writing the word this, it refers to the eventual object that will be created from this class.

So in this case it'll be Jenny and in this case this will be James. And we're saying that set this eventual object height to this property,to the height that comes through as an input.

So if we delete this print statement here and we go back to our height equals height, height is gonna be set to itself.

So 15 will equal 15 which does nothing to affect this one.

And we know that when we run it, we get null for Jenny's height and null for James' height.

But as soon as we change this to this.height is equal to height,it means that this human class's height property is going to be set to the height that gets passed in from the constructor, which is right here and here. Then,now if we had run, you can see it suddenly works and the code now knows what we're referring to.

When we're talking about this.height versus when we're just talking about height. So what if we also wanted to give every human object a starting weight property as well?

So let's add another property in called weight.

And it also has no starting values,so we're not setting it equal to anything.

And then inside the human constructor, we're also adding the double that's called weight.

And then we're going to write this.weight equals weight.

So Jenny will have a starting weight of maybe 3.5 Kg let's say, and then we'll give James a starting weight as well.

Maybe he'll be 4.2 Kg. And this works,but you saw how much work I had to do in my constructor for this to be after to work.

I had to add it as an input parameter.

Then I had to write this.weight is equal to the weight, to specify that this property is going to be equal to whatever gets passed in here.

And you can imagine if I had 10 properties to initialize or 20, then this is going to be really time consuming.

So because this is such a common behavior, Dart actually has a little bit of syntactic sugar.

So something that they created to make this process easier for you.

So instead of doing all of this, we can simply delete everything that's in the curly braces including the curly braces. And instead of writing this is an input called height,this is an input code weight,we can simply write this.height and this.weight.

And we cap it off with a semicolon.

Now the same thing still applies when we're talking about these curly braces.

So if you have them there around these parameters, then it will be named when you construct human.

But if you delete these curly braces, then you can omit these named parameters and just initialize it in the order that it was defined down here.

So I prefer it looking like this because otherwise it gets confusing when we have too many input parameters.

But this dramatically reduces the amount of code that you have to write in order to initialize some of the properties of the class.

So this line of code does exactly the same thing as what all of this code does,but it just cuts it down by quite a lot. And it looks a lot simpler.

Now it's time for your challenge. If you remember in our question class, we still have the same format that we saw earlier in our DartPad.

So using what you learned just now, reformat this code so that uses that syntactic sugar that Dart gives you to make the constructor for our question class a lot shorter and a lot simpler.

So pause the video and try to complete this challenge. All right.

So this should be super simple.

Our inputs for our question class come in here and we're setting it to equal our properties up here inside the constructor.

So instead of having all of this code, we can delete the entire body of the constructor and add a semicolon.

And inside the parentheses instead of having input and having the data type of the inputs, all we need to write is this. and then the name of the property that we're going to initialize first, which is questionText. And then it's going to be the second one that we're going to initialize this.question Answer. And this will mean that your code in your QuizBrain when you're writing initialize a new question object passing in the question text and the question answer, will still work exactly the same.

But now, we've just vastly simplified this constructor by using something from Dart that makes this process a little bit less wordy. If you want to learn more about Dart constructors, then of course the Dart language towards the place to go, and they talk about everything that we learned in this lesson and even more things that you can do with it.

And of course there will be a link to this in the course resources.

