Computational Thinking

Video Transcript

Video 1 – What is a Computer?

What is a computer? This is something that we use every day. We use office suites; we use browsers to navigate the web. We use a multitude of applications. But what is it really, at its core? What are the underlying patterns? So, here at the outset, we want you to take a step back to think through, to write down, if you like, what are some of the components? What is a computer?

Video 2 – Restaurant & Library

There are several ways in which you could explain a computer. But I want to take an informal approach and make a comparison to everyday activities that are analogous to a computer. In this case, we're going to be talking first about eating in a restaurant. These are things that we recognize, and we understand the cycle. We wait, we sit, we place an order that order gets carried to the kitchen, and eventually, it comes back to us as food. And so, if we think about the parallels here, the analogous activities to a computer, we can think of the customers as those providing the input and later receiving the output.

And as the server that will be the one who carries the messages to the kitchen, to the work area, and where you're loading from storage, let's say your recipes, in this case, onto your work area. And of the cook as being the person who performs the work, carries out the instructions, the CPU. And finally, as mentioned, you get back that output and completes that cycle. Now, we could think of it as well as a library and a student trying to carry out work within it.

You could think of the work that needs to be performed in extracting that information that is necessary for you to carry out that work, and placing your order, sending your instructions, and giving them to the librarian. That would be the one carrying out the work, retrieving the information that you need it, and your work area, once again, in this case, being your desk, and that you couldn't load all of that information at once because you wouldn't have the space just like we couldn't load all the recipes and all the recipe books onto a work area in a kitchen, and so, in this case, you'd have to work with a batch at a time.

And so, one of the observations is that we can make here is that there are some analogous patterns to the physical world. And that in these first two examples, we're looking at domain-specific parallels. In that, you could think about these as fixed programs as opposed to a generalized platform like a modern computer that can run any application. And that is the wonderful thing of modern computers in that you can program them; you can have them carry out your instructions. And we have the amazing variability in creativity of the entire software industry,

creating products, creating applications, creating entertainment. And that is the benefit of having a machine that can be reprogrammed, and you can have it execute your own instructions.

Video 3 –Create A Web Document Using HTML

Let's write our first program, declarative programming in HTML, the Hypertext Markup Language. In this case, we're going to be using the same technology, the same language that is used to layout pages on the World Wide Web. That is, when you're navigating the web, say with Google Chrome, going to newspapers, going to medical sites, going to educational sites. The way these pages are laid out is using HTML. And in the old tradition of software, we will write our first program and call it Hello, World! So, let's go ahead and move over to the browser. As you can see here, I have a blank tab on Google Chrome.

And on the left-hand side, I have an editor. This editor is Visual Studio Code, and it's an open-source project and the product itself, the editor, is free for anyone. So, you can see here I did a search on Google for Visual Studio Code. You can see there the address, and in this case, the page that I'm looking at is for my operating system, which is a Mac. You can download the one for your operating system, and it will look very much the same. So, I'm going to go ahead and go back to the editor, and I'm going to start off here by creating a blank document, a new document. Then I'm going to go ahead and save it, and I will save it as mentioned, as hello.html.

The .html extension indicating that this is, in fact, a web document. I'm going to start off here by simply typing 'Hello World!' and then I'm going to drag and drop that document onto the browser. I'm going to show you here, and I'm going to go ahead and drag and drop, and as you can see there, we get 'Hello World!'. Now, this has no syntax, no markup. There's nothing taking place here other than simply giving us back the text that we type. Now, we're going to next use a declarative statement, and we're going to add a tag. This is the way you indicate intent to the page that is going to be interpreted on the browser.

And we're going to add an '< h1 >' this is a header tag, and I'm going to go ahead and move the text that we type into that tag. And you can see there that that tag has some symmetry. Many do on HTML, but not all of them. I'm going to go ahead and now reload the page, and as you can see there, the text looks much different, and the reason for that is that the browser recognized that this is HTML, an HTML tag, in this case, a header, and laid it out in that manner. In the same way, if we were to add a paragraph and the paragraph tag is a '< p >', and I'm going to add 'Hello World!' here a couple of times simply to simulate adding some text.

You can see there that the text looks different, and that is because we are already starting to layout this document. We had a header; we have a paragraph. Now, to make things a little bit more fun, let's go ahead and add an image. And I will show you here the tag, and you can see there that it has a place holder for the image. We're going to go to the web next and look for an image and let's say we want an image of Boston, and we will use one from the Wikipedia page. And I'm going to go ahead and select this one; this is the view across the river from MIT.

And I'm going to go ahead and look for an image here. I want to make sure of the size; this one is 1200. I'm now going to go ahead and save it, I'll save it to the same directory where we have our HTML file, and I will call it Boston. So, you can see there that the name of the file is boston.jpg. So, I'm going to go ahead now and add this to the source. So, this will be boston.jpg, as you can see there. And if we go back to our page and we reload it, you can see there that now we have the image.

You can see there that it's a bit pixelated because I made the texts very big, so it could be visible originally when we were doing the plain text with no tags. But now that I've made it a 100%, you can see there that it fits nicely within the page. And you can see there that we have a few components already. Now, the other thing that we could do is that we could add a link, and this is the mechanism that is used to connect the billions of documents that exist on the web. This tag, it starts off with an '< a >', and you can see there that I have now a placeholder for the address that we're going to navigate to.

And in this case, we're simply going to navigate to W3C; this is the standards body for the web. And I'm going to enter here 'W3C', then I'm going to go ahead and save the file, reload the page here. As you can see there, we get W3C. I'm going to go ahead and make the text bigger again. And we have it here at the bottom of the page. And when I click on it, you can see there that it takes us to W3C. So, as you can see there, we have the makings of a page. We have a header, we have a paragraph, we have some images, and we have a link.

Now, it's your turn to try to do the same. This is a fun thing to try. As mentioned, this is your first program. Go ahead and try to replicate what I did. Add a picture of your choosing, text of your choosing, and I will refresh you here with some of the options of what we did. We started out with a heading, and we used Hello World! Then we use '< p >' to enter a paragraph. Then we added an image. And lastly, we navigated to W3C. Now, you can enter any link that you like. Here I have added one for Wikipedia. So, now it's your turn.

Video 4 – Creating Shapes In HTML

Now, let's add a shape to the page. We will use a < div > tag, as you can see there, and we'll set the attribute of style. And you can see there's some of the properties of style z-index, position, left, top. Now, let's go ahead and move to the editor, and we'll start off by creating a new document. We're going to go ahead and save it and call it shapes.html. And as mentioned, we're going to start off here with a '< div >' tag. Inside of it, we're going to add the attribute of 'style='. And the very first thing that we're going to add is a 'z-index', and we'll start off with a 'z-index:' of '5'.

Next, I'm going to go ahead and return simply to make this a little bit more readable and let me go ahead and give myself some space. The next thing that I'm going to set is the position. We'll set our 'position:' to 'absolute;'. Then we're going to set the horizontal position. This will be set with the property of 'left:', and we will start off at a '100 px;'. Then we're going to add 'top:', which is the vertical coordinate. And in this case, we're going to go ahead and set it at '100 px;' as well.

Actually, no, let me set that one at '200 px;'. Then after that, we're going to set the height and width. We'll start off with the width. We'll set the 'height:' and 'width:' at '100 px;'. Next is the 'height:', '100 px;'. Then we're going to set the border-radius. So, this is the 'border-radius:', and we're going to set it at '50%;'. This gives us a nice rounded corner, which makes it look like a circle. Otherwise, if we set it at 0, it would be squared. And this is something you can try afterward. Then I'm going to set my background, and I will start off with a 'background:' of 'black;'.

So, as you can see there, I have set a '< div >', I have set the attribute of style, and then a number of properties within this style. Now, I'm going to go ahead and drag and drop that page onto the browser, and as you can see there, we get a shape. To demonstrate some of the properties here, let's go ahead and move the horizontal coordinate by '300px;'. Let's go ahead and move this '400px;' down in the vertical coordinate. Then let's go ahead and make this bigger. Let's go ahead and make it three times bigger.

Well, three times bigger and then, let's change the color. I'm going to go ahead and set it to 'green;'. And now I'm going to go ahead and reload the page. And as you can see there, we have moved the shape across the screen. We have moved it also vertically, and we have reset the color. Now, to emphasize those coordinates that I mentioned, you can see there the Z-index, you can see the Left and the Top.

And note that the vertical index, the vertical coordinate is positive downwards within the page; this is a convention of graphic systems. You can see there the X-coordinate, the horizontal direction being Left. And you can see z-index coming out of the page. As you can see here from our original coordinates, we set top at 200, left at 100, and we set our height and width at 100 as well, and the initial color was black. Next is your turn. Go ahead and create a couple of shapes. Play with the properties. Play with the positioning, and get comfortable with creating shapes.

Video 5 – Shapes Exercise

Now that you have some experience creating shapes, you can think of the coordinate system that you worked with as the following. The traditional X, Y, and Z is your left, top, and z-index. As before, noting that the coordinate system is anchored in the upper left-hand corner where you have z-index coming out of the page. The coordinate, the vertical coordinate going downwards for Top, which is the traditional Y-coordinate. And noting once again that positive is downwards and that this is a convention of graphical systems. And the last one, the horizontal coordinate, is called Left.

The assignment we want you to carry out for this exercise is the following. We want you to create the following shapes as you see them; note that they have specific colors. Note that the z-index is specific as well. Certain shapes are on top of others, and that there is a certain symmetry to the page. They are arranged in a certain order. They are placed in a certain way. Notice that they are in the middle of the page as well if you would like to see some additional colors or to find colors that are close to the ones that we saw on the page. You can follow the link here on the upper right-hand corner.

And I have provided here for you once again the attributive style and the properties that we worked with. If you happen to move very fast through this exercise and you want an additional challenge, you can attempt to do the following. But note that this is a stretch in case that you move very quickly through the first one and are looking to test yourself and your understanding, and you want to try some additional shapes. So, now it's your turn. Go ahead and try to create the shapes in the configuration and the colors and the arrangements and the z-positioning that you see before you.

Video 6 –Introduction To Bootstrap

One of the advantages of writing software in the modern era is the great availability of very good open-source projects. Bootstrap is one of the most used frameworks for styles; it's one of the largest open-source projects as well. And it's a resource that has been used by millions of people to create millions of pages around the world. Let's go ahead and navigate to the site. And we will do so by creating a search on Google. And you can see there that it's the first head, and there is a great deal of code, and styles, and components within this project. This project has been running for over 10 years, and there's a great deal here to digest.

Now since you're just starting out, I just wanted to show you a very simple example and show you how by making one small change, you can load images onto your landing page that look stunning. So, let's go ahead and show you here the example that we're going to use, and you don't need to download anything. We will provide this file for you, but I just wanted to show you where it comes from. This is the page that we're going to be using, and as you can see here, I have previously downloaded this onto my folder. I have simply two files. One that provides some styles that's the '.css' and an HTML file.

As mentioned, this is not something you need to understand at this moment; we're simply going to show you how to make one small change to show you how you can load images onto this background. So, I'm going to go ahead and load, first of all, this page on to the browser. And as you can see there, we now have it in the browser. I'm going to go ahead and load the styles onto my editor. And as you can see here, I have a section in where I'm going to go ahead and make those changes. The green denotes that this is now comments, and I'm going to uncomment this.

And as you can see here, I have a background in a URL that is pointing to those three images that I have on that folder. If I bring back the listing of files, you can see here that my first three files are images. So, now that I have uncommented that code and that code is being used and interpreted by the browser. I'm going to go ahead and reload the page here. And as you can see, we get this stunningly beautiful page that has a very pretty picture behind it.

I'm going to go ahead and do the same now for the second picture, and if I reload the page, now we get this nice beach background, and if I do it once more, we'll go ahead and get to the right part. Then we get this impressive look at lion. So, this is something to try out simply to see how easily you can leverage some of those resources. There's no need to experiment more than that. We're just starting out, and this is simply a nice demo.

Video 7 – Variables

Let's look at the core components of languages, often referred to as Turing completeness. That is the structures within languages that allow us to write any program. Truly a remarkable achievement. Given that if that is the case, then we're only limited by our imagination. That is anything we can dream of, we can think of, we can create. Now, more concretely, we will be looking at the following; Variables, Arrays, Conditional statements, Loops, and Functions. Now, these can be pretty abstract terms. So, let's try to ground them. Let's try to ground them to things that we use every day in our physical world.

And we'll start off with the concept of a variable. And often, the example that is given is that of a bucket. You can put many things inside of a bucket. You can put a mop; you can put blueberries, you can put vegetables, beer, fish, you name it. A bucket can hold many things. And once we start creating code, we will see a very similar pattern when it comes to holding things in variables. That is, we can put many different types of data inside of a variable.

And, of course, when it comes to our physical world is not limited to buckets. You can have a similar concept when it comes to containers. This has transformed the shipping industry around the world. And today, in port cities, you can see there's big stacks that are certainly impressive, and you see similar things in votes as well and ships. Now, we're going to ask you to try to do the same. Look around your physical world, think about things that you touch, things that you engage with, and try to find other variables in your physical world.

Video 8 – Variables Code Exercise

Now, let's go ahead and write some variables. Let's write some code. Let's anchor those ideas in a very specific set of instructions that we will type here in a moment. And in order to do so, we will use the browser Console. As you can see here, I have a snapshot as to how to open the Developer Tools. And depending on which browser version that you're using, you should be using Google Chrome. But however, depending on which version of Google Chrome you use and what themes you have applied, it will look slightly different. However, all of them will have more tools, as you can see here within the menu, and all of them will have Developer Tools. And you can see there what are some of the shortcut keys for each operating system.

Now, let's go ahead and move to the Console. And I'm going to go ahead and use the shortcut keys to open up my developer tools. And as you can see there, I have a menu with a number of things, Elements, Console Sources, and so on. I want to be on the Console, and I'm going to go ahead and increase my font size so that it's more visible. Now, at this point, we're going to go ahead and create a few variables. We do so using the keyword 'var' to denote a variable; then, I'm going to call my variable 'number1', and I'm going to assign it the value of '10'. And what you see there that is prompting me is the IntelliSense.

Since I defined these variables previous to starting the recording, I'm going to do the second one, which I will call 'number2', and I will assign the value of '15'. And one more note is, well, you see, they're 'undefined'. For now, let's just ignore that message that's coming back; it does not conflict with anything that we're trying to do. Then we will enter 'number3'. This is the third variable that we are defining. And now, we have all of those in memory. And we can recall them using the Console log mechanism. And we can do that by typing 'console.log', and then inside of it, we will write 'number' and in this case, '1', as you can see there.

And we could do so; we could do the same thing by recalling all three of them. This would be '2', and this would be '3', and you can see there that we can recall those values. Now, I'm going to go ahead and clear the content here in the Console. And I'm going to go ahead and make some comments about the rules of naming variables. The very first one is that variables are case sensitive. So if we entered 'Number1' in uppercase, instead of like we did originally making number lowercase, this would be a separate variable, and let's call this one '99'. Then number with lowercase.

So if we enter 'console.log' once more, and we entered the values, and in this case, let's just remove the third one since we don't need it. And we're going to modify this one to be 'Number1' with uppercase the first letter. And we'll see there that we get '10' and '99', meaning that the platform recognizes case-sensitivity and that number starting with lowercase is different than number starting with uppercase. Another requirement is that variables start with a letter or an underscore. That is, if we created '1number', this way, we assign '99' to it. We'll see that we get an error message because variable names needs to start with a letter or an underscore.

Now, for the rest of the characters in the variable name, they can be any letter, number, or underscore. Now, as you might imagine, the name you give your variables matters. That is that if you make your variables very difficult to determine what they are, they will hurt the visibility or the understanding of the code that you have written. And when it comes to maintaining a piece of code, that would make it very difficult. And so, in general, you want to be as transparent as possible to capture your intent in your variable names. So, a simple example of this is imagine you had a program with a 100 variables, and you named all your variables V1 to V100.

Trying to determine what each variable is would become quite challenging. And so, in general, try to make your life easier. Because even if you're not thinking of other people, if you come back to your own code in six months, you will forget what you were trying to do. And having expressive variable names will be something that you're thankful for. Now, as a less point, we typed everything into the Console, but it would be nice to be able to have this written in a file. So, that's what I'm going to go ahead and do next.

I'm going to go ahead and save this file here as variables.html. And then inside of it, we need to add a '< script >'. And this indicates that what we're going to write inside of it is code, and in this case, JavaScript code. And so, I'm going to go ahead and copy and paste the variables that we wrote in the Console. And as you can see here, we have 'var' one, 'var' two, and we have 'number1', 'number2', 'number3'. And then, we are writing each of those values to the Console.

Now, if I were to go ahead and load this file onto the browser and I'm going to go ahead and do that now. I'm going to go ahead and drag. And if I open up the Console, you'll see there that we get '10', '15', and '25' as expected. I can reload this again, and again, and again, and I continue to get the same thing. So, once you start to write your code, you'll capture it in files so that you can execute it multiple times without having to retype everything into the Console.

Video 9 – Arrays

You can think of Arrays as lists. And although a closet may not seem like a list if you look at each individual compartment, you can think of its items as a list of things, say a list of shirts or a list of shoes. So, say a list of list is a closet. And more generally, it is some grouping of containers, in this case, of buckets, where we refer to each of the items in the position or its index. And you can see here numbering that we will use to access those positions, which would be 0 for the first element going all the way to four.

And this would be a representation of something in our physical world, which closely resembles that. An organizer with four containers that are numbered 0 through 3 for its position. So, more generally, you can see here that an Array has elements and each of those elements have an index position. And in this case, we're talking about a one-dimensional Array with four element. Now, if we consider a two dimensional Array, it would look something like the following. In an example of real life, this is a parallel, simply an example.

And if somebody asked you to come up with a naming convention and asked you to do identify the container with the yellow dot. You potentially could come up with a rows and columns approach, in where that it position for the yellow dot would be row 0, and the column would be the third one. And we could extend this more by having more dimensions. So, now take a step back and think about your physical environment and think about what other things are Arrays.

Video 10 – Arrays Code Exercise

Let's go ahead and write some arrays. We'll anchor our ideas of the physical world into now code implementation. And I'm going to start off here in the browser, and we're going to go ahead and open up our developer tools. And as you can see, there I am at the Console. I'll start off by creating an array of letters, and I will call it 'letters'. And you can see here that I am going to define an array using '[ ]'. Now, inside of the '[ ]' is where we put our array elements. And in this case, we're going to have letters. So, I'm going to start off here with the letter ' ['a', ' and then I separate each of the items with a comma. And I'm going to have four items in this array. One more letter, which will be the letter ' 'd'] ', and that creates my array.

Now, if I were to type 'letters', here at the Console, you would see that I can see what the array holds and that if I expand the arrow there, I can see the array index in positions. So, at position '0:', we have ' "a" ' at position '1:', we have ' "b" ', at position '2:', we have ' "c" ', and so on. So, if I wanted to access any one of those items, we can do so by using '[ ]' once again and then the index that we're trying to access. So, I will write here 'letters', and I'm going to go ahead and

access the last position, which would be ' "d" '. And you can see there that I get ' "d" ' back. Now, this is an example of a one-dimensional array. Let's go ahead and create a two-dimensional array.

I'm going to go ahead and clear the Console here. And I'm going to call the two-dimensional array 'numbers', and I'm going to put within the '[ ]' once again my items. But in this case, each of the items, as opposed to be in a single letter as it was in the past or a single number, say, I'm going to have inside of it arrays. So, I have an array of arrays. So, I'm going to go ahead and press the Shift key so I can get some space. I'm going to enter here a couple of lines, and I'm going to add my array. So, this is an array that holds as its items arrays, and so inside of it, I'm going to create an array that has four elements.

I'm going to separate this array from the second one that I'm going to add. And I'm going to once again press the Shift key. Then I'm going to add another '[ ]" for my second array, and this one is going to hold the numbers '[9, 10, 11, 12]'. So, at this point, I have created my array. And if I type 'numbers', you can see there that I have an array. You can see here the square brackets at the beginning, at the end. And inside of it, it holds an array as well; two arrays. And each of those arrays, as you can see, hold four items. So, if we wanted to access, say, the first element in this 'Array', we would do the following. So, I would enter 'numbers' and then enter '[0]' for the first item. And as you can see there, we get the first item back, which happens to be an array.

Now, if we wanted to access a position on this array that we have just returned, then we would add the position of that array on a additional set of '[ ]'. So, in this case, we want to access the fourth position, which, because it's zero-based, would be '[3]'. Then when we hit **Return**, we get back '8' as expected. Now, we can do more than just access the content of that position. We can also assign it. So, let's say we wanted to change '8' to '99'. And I'm going to go ahead and do so. If we access that position, again, you can see there that we have '99'. And if I accessed simply the very first row, you can see there that the number has changed. So, we can access arrays using the index. We can also assign the contents of array using the index.

Video 11 – Conditional Statements

Conditional statements, a core component of computation, has a strong analog to our physical world, that is, conditions upon which we go one way or the other. The literal fork in the road that determines which path we choose based on the condition. In this case, I'm showing here a simple example of a traffic light where if we encounter a red light, we stop, and if we encounter a green light, we move. Similarly, you can consider a scenario where if you have rain based on that condition, you deploy an umbrella.

Another is access to a location, say a shop, that you can only access based on the availability that is determined by the schedule you see before you; we would have to meet the condition of being within the opening hours for us to be able to access the location. Another one is that of roads that add some additional restrictions to the common ones of the open road. In this specific one, we have a bridge with low clearance. If we had a car with a low height, then we would be able to meet the condition, be able to use this bridge.

Otherwise, we would have to take the fork in the road and go on a different path. So, as you can see, these are scenarios that we frequently come upon and that, as you might expect, are also

very important when it comes to the world of computation. So, now it's your turn. Think about these analogs and think about a few examples that you could identify on our physical world that determine one set of actions. One fork on the road, one path on the fork on the road, or another.

Video 12 – Conditional Statements Exercise

When we write code for conditional statements, we're going to be making comparisons, and those comparisons are going to evaluate if it's true or false. As you can see here, I have the greater than symbol, and in this case, this would evaluate to true. This is the way we do greater than, or equal, which this would still be true. Now, if we simply wanted to make a comparison, as you can see here, of these two parameters, which are two numbers being equal, we could do that with the triple equal signs. We were checking for not equal; we would replace one of those equal signs with an exclamation point.

And finally, if we wanted to use logical operators where we were comparing a couple of additional comparisons, we could use '&&' there to check if both of them were true, or we could check if one or the other one was true as this, as you can see there on the second row. And then the last one is simply a negation or a 'not', where in this case, as you can see there two numbers are equal. And so, if we say they are not the not true, then we're going to get false. So, with that in mind, now let's go ahead and write some code. Let's start off by creating some simple comparisons. So, we will test if '5>1'. And as you can see there, that is 'true'.

We can test as well if '5>=1', and that will still be 'true'. We can do the same thing with the comparisons of simply checking to see if these two numbers are the same, and you can see there that they are not. Or we could do the same one by replacing the 'not equal'. And so, in this case, we get that to be 'true'. Now, to use the logical operators, we could do the following, you can see there that I have two comparisons. One of them is comparing '5' and '10'; '10 < 5 && 5 > 1', and we are checking both of them. And as you can see there, the logical operator returns 'true'.

On the next one, the 'or' operator, I am checking two values, two comparisons there, and you can see there that one of them is true while the other one is not. But since we're doing the 'or' operator, then we evaluate to 'true'. And then the last one is simply using 'not' on the evaluation of the internal comparison, which would be true. And so, if not true, we get 'false', as you can see there. Now, let's go ahead and clear the Console here. And let's go ahead and create a variable. I'm going to call it 'age'. And I'm going to enter here '18'. And then we're going to write now a conditional statement that is going to use comparisons.

For that, we use the notation of this to check and see if we evaluate or we compute what is within the 'if' statement. So, we could say something like 'if (age >= 18)', Then we say, you're an adult. So, let's write here 'console.log' and simply write '('adult')'. So, and if we evaluate that, you can see there that we get back 'adult'. Now, if you had a longer statement, you would want to put it inside of '{ }' that denotes a block of computation. So, let me go ahead and do that. Now, I'm going to go ahead and enter the '{ }' here, and I'm going to press Shift to be able to get a space. And then inside of this, we're going to write to the 'console'.

So, let's go ahead and try that, and as you can see there, we once again get back 'adult' in the 'console'. So, let me go ahead and add to that, then an alternative. And that will be the 'if then' part of the construct. So, let me go ahead and try that out. I'm going to do 'if', 'if then', and that is this part of the computation. But then, I'm going to check for 'else'. If that happens to not be true, then we will write 'minor'. And let me go ahead and make that change here. And now, we're going to go ahead and carry out that computation. And we get 'adult' once again. However, if we change our value of 'age' to be, say, '11', and we rewrite the code.

You'll see there that now we get back 'minor'. And so, as you can see there, we can use comparisons within conditional statements to be able to determine the path that our execution takes and the branch of computation that we are going to follow. In this case, this is a very simple example, but you could see making big decisions in the course of the execution of your program depending on data that you were evaluating. As a final point, we will type once again the information that we wrote or the instructions that we wrote in the Console into a file.

So, I'm going to go ahead and create a new file. I'm going to go ahead and save it as 'conditionals.html'. Inside of it, I'm going to add a '< script >' tag. And then inside of that, I'm going to copy and paste some of the syntax that we wrote. And as you can see there, I have added a little bit more this time simply to denote or to be able to determine what it is that we're looking at in the Console. Now, I'm going to go ahead and drag that file onto the browser, as you can see here. Now, we're going to go ahead and open up the Console. And as you can see there, we get the information as expected.

Video 13 – Loops

Repetition or loops is a core component in a computation. It's also commonplace in our physical world. As you can see here, we have the phases of the moon that repeat. You can see also that it's a component of learning. The old-style punishments require you to write a phrase over and over in the board. And we see it in many places. When it comes to calendars, the days of the week repeat, the numbers of the month repeat, the months in the yearly calendar repeat. And we see this in many other places as well. You can see here that when we look at a clock, there are three hands, the second, the minute, and the hour.

And that if you watch long enough, all of them repeat, cycle, loop through the face of the clock. And depending on certain conditions, certain events take place. And so, before we go forward, I want you to think about your world and think about things that repeat. And just as importantly, try to identify the conditions upon which looping repetition stops or that you move from one state to the next. Much like I mentioned in the case of the calendar, that June follows May, that Monday follows Sunday, what are the conditions in the observations that you make that stop the looping or move from one state to the next?

Video 14 – Loops Exercise

There are a few ways in which you could get code to repeat, to loop. In this case, we're going to take a look at a while loop. And as you can see there, the while loop checks a condition prior to executing the block of code. And as long as that condition evaluates to true, that block will continue to be run. Now, a do while loop does things a little bit differently. In this case, the block of computation is executed, and then the while condition is considered. So, in one case, you have the condition being checked at the beginning, and in the second one, you have it being done at the end. So, let's go ahead and write a while loop at the Console.

We'll start off with the 'counter', and we will initialize that counter at '0'. And I'm going to hit shift+return to be able to write all of the code that I'm going to write for the while loop in one line and to be able to get wrapping. And then I'm going to write the condition. And I will enter that as long as the '(counter < 10)' the loop will continue to run. So, I'm going to go ahead and write the body of the loop using '{ }'. And the very first thing that we're going to do is that we're going to write to the 'console', the value of the counter. And so, we will enter '(counter)' here.

And then we will increase our counter. If we do not do this, we will write an infinite loop because the counter will never increase. The while condition will always be true, and this is something that happens the very first time you try loops. You see nothing happening, and your computation never returns, and you're wondering what is taking place. And so if you do not do your counter right, you will go into what's called an infinite loop, meaning that while loop continues to run forever. So, in this case, we're going to run, we're going to assign to our 'counter', the value of 'counter + 1'.

That means that on each iteration of this loop, our counter is going to be increased by the value of one. So, once we finalize, and I'm now outside of the while loop, I simply want to write to the console the last value of the counter. And I will write a message here, writing '('counter last value: ' ', and I will then simply '+ counter); '. And this will simply concatenate, that is, bring both of those strings together and write that to the console. So, I'm going to go ahead and put it in here some spacing to make it more readable. So, let's go ahead and run that code. And as you can see there, we get values from '0' because our counter started at '0' all the way to '9'.

And we do not get '10' because that is already violating the condition of the while loop and that the last counter value is '10', that was the last value that was assigned to counter. And then once the while loop came back, the counter was no longer less than 10; it was equal to '10'. So, at that point, the loop stops, and we are done with this loop and with this computation. Now, let's go ahead and write a do while loop. And I'm going to go ahead and clear the Console for that. And I'm going to go ahead and paste some code. And as you can see here, I start off by resetting 'counter' to '0'.

I then have my 'do' in curly braces. And it's the same thing that we had before. We're right into the 'console', and then we're increasing our 'counter' by '1'. Then the last line is the loop condition 'while (counter<10);'. And then, in this case, I am writing one last line, just like I did before. I want to see the last value of the 'counter'. So, I'm going to go ahead and run that. And as you can see there, we get the exact same thing. We're going from '0' all the way to '9'. And the last value of

the 'counter : 10'. Now, once again, I'm going to go ahead and write the code into a file. I'm going to create a new file. I will save it as loops, loops.html. And in this case, I'm going to overwrite the one that I have in that directory.

So, I'm going to go ahead and replace it. I'm going to go ahead and paste code that I've written beforehand. And as you can see there, I have funny-looking syntax. And the reason for that is because I did not add the '< script >' tag. Now, once I do it, you can see there that I get different values, and I have a warning here. And if we mouse over it, you can see there that it is requesting, it was requesting a ' , '. But the issue is that I did not close the statement in line '8'. And so, I'm going to go ahead and do that. ' ; ' is the line terminator. I will do the same thing here. And let me make sure I am in the right place.

And now, we're going to go ahead and load that file onto the browser. And let me go ahead and do that. And I'm going to go ahead and open up the Console. And as you can see there, we get the same thing that we did before. We're running from '0' to '9'. The last value is '10', and then the last value is '10' once again. And we have in the first instance, we have a 'while' loop, and then on the second one, we have a 'do while' loop. So, go ahead and try variations of this. Try to avoid writing an infinite loop.

You may want to do it once just to see what happens. If you get stuck, you can go ahead and close your browser and then open it up once again, shut it all the way down to make sure that stops running. But you can get a sense there of the type of error that might occur if you make any mistake in your counter. You can try increasing in many different ways, and to give you a specific target, I've suggested here a couple of exercises. One of them is a loop that prints 1 to 1000 and then stops, and the second one is a loop that prints the multiples of 5 from 1 to 100.

Video 15 – Functions

As your code grows and your complexity grows. Having a great deal of lines of code on a page can become overwhelming. And so, you want to take blocks of these instructions that perform a certain task and set them aside. Give them a name and call them when needed. And so, what happens that every programming language lets you create these blocks that, when called, perform that task. And we call these blocks functions. And in general, we have a model that is, as you see here in this illustration, where we take some input, the machine, the block of code performs some tasks, perhaps a transformation, and then we get some output.

And if we think about examples in our physical world, you can think of a machine, say, in this case, a machine that creates yogurt. You give it the instructions, say, the input, the raw material, and the machine itself creates the yogurt. Similarly, if you think about a weaving machine, you could provide its input in terms of threads, and the machine itself would be able to put together a piece of cloth. So, in general, we have this model where you have some input. Then you have a machine that carries out the instructions, and then you get some output, perhaps a transformation.

Now, you can imagine of all of these different discrete pieces of computation, these blocks as coming together to create a system. And oftentimes, this is a good way to think about how to build a larger system composed of these smaller units. It makes it easier to think through. It makes it easier to debug once you start to construct bigger programs. More formally, when it comes to the syntax, what you do when it comes to a function definition is you use the keyword or function, then a function name, which in this case is simply name, then a number of parameters. And you might have here many, many, or you might have none.

And then inside of that body, inside of the curly braces, is the code to be executed. Similarly, when it comes to the invocation, you use the function name, which in this case is simply name, and then the parameters that are part of that function signature. And we call the function signature the definition, the one that we just saw. Now, when it comes to sending information back from that call, the function itself uses a keyword called return that you will see in a moment when we write some code. And so, when you invoke the function, whether it's named in parentheses and parameters if you have some.

Then you can receive the response from this function and assign it to a variable. To start, we will write a very simple function. This 'function' will be called 'add', and it will take two parameters to be added. In this case, I'm going to go ahead and add the '{ }' here for the function body. And I'm going to go ahead and, within it, get the result of adding a plus b. So, I'm going to then enter 'a + b;'. And then I'm going to return that result using the keyword 'return'. Now, once we do that, we will have defined the function. I'm going to go ahead and hit return. And now, if we invoke the function using 'add' and passing two parameters, in this case, let's say '(1, 1)'.

You can see there that the response that we get is '2'. Now, we could easily add an additional 'function'. And I'm going to go ahead and use the last one I wrote as a template. And we're going to write now a function called 'multiply'. And so, it takes two parameters again. And this time, we're simply going to add, instead of, sorry, we're going to multiply instead of add. And so, we've now defined multiply. And if I give it two parameters, let's say '(3, 3)' this time, then you can see there that we get '9' back. So, this is a very convenient way to wrap a set of tasks and be able to set them aside, as mentioned.

Once we start to write larger programs, you can separate these into separate files. And that cleans up your code and makes it easier to read, which means easier to maintain. And even for your thought process when it comes to being able to process all the instructions you have written in the page. Being able to have this encapsulation of a concept turns out to be very helpful. So, I'm now going to go ahead and add these two functions to a new file. I'm going to go ahead and create the file, and I will call it functions.html. I will then add the tag for '< script >', and inside of there, I will add my two functions that I have previously written.

And I'm going to go ahead and paste that onto the page. And as you can see there, it is the same syntax that we wrote. And so, now I'm going to go ahead and drag and drop that HTML file onto the browser. And as you can see this time, I have not written onto the Console. However, I can still call those functions because they exist; they have been defined when that page loaded. And

so, I can once again do, let's do '(5, 4)' this time. This is 'add', and you can see there the result that we get back.

And we can also do 'multiply', and let's do here '45', well, yeah, '(45, 6)' and you can see there that we get the result back. So, go ahead and try functions, play around, see what kind of functions you can create. Try to replicate what was written onto the console, try to write them onto a file, play a little bit with the possibilities here, and get comfortable with functions. As mentioned before, the more code you write, the more you will want to leverage the use of functions.

Video 16 – Functions, Libraries, Code Exercise

Once you start to write libraries, you'll find that you can use them in other projects. More than that, that you can wrap them into libraries and share them with other people, and share them in registries to share them in repositories. And so, this turns out to be a great way in which you can share and consume other people's code. And in this case, we're going to do precisely that. We have written a library that has a number of features. You can see here the descriptions of some of them. You can create a ball programmatically. This is very much the same thing we did when we did so declaratively by using HTML and writing into the page.

We will now be able to perform the exact same thing with a simple function call, which is 'create()'. We will be able to then, once we have that ball, to be able to set the size, the color, to move it, to set random colors, and other functions, as you can see. We will provide you with this code, and you will have three files, ball.html, ball.js, and magic.js. I'm going to show you here how they look within my file system. I have precisely those three files. And in a moment, I'm going to go ahead and drag and drop that onto my browser. So, let's go ahead and do that now. I'm going to go ahead and drag in 'ball.html'.

And I'm going to go ahead and open up my developer tools. And as you can see there, I am on the Console. Now, with the action of loading that page onto the browser, we have loaded in all of the libraries that we have written. All of the functions that are ready for you to use now, programmatically. So, the very first thing that we're going to do is that we're going to create a ball. And so, I will enter 'var ball', and then I will call 'create()' to create a ball. And all of that logic has been hidden away from you, and we have wrapped it up into those files that I just showed you. So, I'm going to go ahead and hit return, and as you can see there, I have created a ball.

You can see there that is green. I'm now going to go ahead and change the 'size' of the ball. And I need to pass in as a parameter, the ball that I'm working with, in this case, it's simply called 'ball'. And then the size that I want for that ball, I'm going to go ahead and resize it, and I'm going to make it much bigger. I'm going to go ahead and make it '300, 300)'. I'm going to go ahead and hit return. And as you can see there, the ball is much bigger now. Next, I'm going to go ahead and change the 'color'. Just as before, you can see there are some of the API, that is, the application programming interface, which is the fancy way to describe the function signature.

In this case, it requires the 'div' once again, which is ball. Remember, we did that in a previous exercise when we did this declaratively, but now we are using variables at the Console and doing this programmatically. So, the very first parameter is 'ball', then its 'r, g, b', which stands for red, green, and blue. And each of them have an intensity from 0 to 255. So, initially, I'm going to go ahead and enter '0' for the two colors. And then the third one, I'm going to put in at full intensity. So, this should be blue. And as you can see there, we have changed the color to blue.

I could just as easily change it to red. And I'm going to go ahead and go pure red here. And you can see there that I have changed the color once again. Now, I can also move the ball. So, I'm going to go ahead and enter 'move' then the '(ball'. Then, in this case, I can move in horizontally or vertically. I'm going to go ahead and move it to the left by '-300'. And then I'm going to go ahead and move it slightly up because you can see I had to scroll a little bit to get it, to see it. So, I'm going to go ahead and move it up by '-300'. Remember, that positive is down. So, in this case, I'm going to go ahead and go up.

So, I'm going to go ahead and move it there. And let's go ahead and see, as you can see there, that has moved, in fact, up. And if I were to repeat this, you could see there that it has moved further to the right. And now, it's outside of the page. And if I want to bring it back, I could do so by changing this to, well, let's change it to '200' and then let's go ahead and move this just a little bit to the right, and so, we'll do so. And as you can see there, it's come back. And let me repeat it just so we can have it fully within the page, and there we go. Now, the next thing that I'm going to do is that I have the ability as well to change the color of the ball to a random color.

Remember, this is yet one more function that is part of that library that you have brought in when you brought in that page. So, I'm going to go ahead and enter here 'colorRandom', as you can see. And then, I'm simply going to pass it the '(ball)'. And you'll see in a moment here, how that changes. You can see there that it iterates through a number of colors, and then it settles on one. So, as you can see, using this library, you can do a lot of what you did by hand. And in this case, now you're doing it programmatically.

And so, this makes things a great deal easier. Now, it's your turn. Go ahead and create a ball, change the color of the ball, change the size, and change the position. And once you've gotten comfortable with what the library can do, you can go ahead and repeat the exercise that we performed initially in a declarative, with declarative programming. Now, you can attempt to do the same by using the functions that are provided for you.

Video 17 – Functions - Loop Exercise

Although writing instruction inside the Console for the functions that we have provided for you is certainly much faster than doing it by hand using declarative programming. Doing it in a loop would be even faster. So, let's go ahead and try that out. And you can see here the code that we're going to try, we have a 'counter', and we're going to try to create '10' balls using the call to the function 'create()'. So, let's go ahead and type that code in. We're going to start off with a

'counter'. We're going to initialize it at '0'. Then we're going to create our 'while' loop, and we're going to repeat this loop, the block of code in this loop, as long as the 'counter < 10'.

And then, we're going to make a call to the function that creates a ball. We're going to go ahead and enter that next, and oh! I just made a mistake there. As you can see, that instruction has carried in. So, I'm going to go ahead and reload my page, and as you can see there, the page is stuck, and the reason for that is that the page has gone into an infinite loop. That while loop has a counter that never increases, and so, that condition is never false. And the reason the page is stuck is because it's trying to create as many of those as possible, and so, it is consuming all of the resources of the browser, and everything is stuck.

If you happen to find yourself in this situation, don't worry, you can close the tab if that doesn't respond; you can just shut down the browser and start over. So, this is a great example here of what not to do. Because if you do it, you can see there that you go into an infinite loop, and everything just freezes. So, as you can see there, I had to start over. I'm going to go ahead and reload the page. And I'm going to go ahead and open up the developer tools, and we're going to go ahead and repeat our instructions. I'm going to go ahead and enter 'counter', going to initialize it with '0'.

And then, in this case, I'm going to go ahead and write the instructions in the same line. I'm using Shift to be able to do returns. Then I'm going to go ahead and enter my 'counter'. I'm going to go ahead and make it '< 10'. So, that is the block of code it's going to execute, as long as my counter is less than 10. Then I'm going to go ahead and 'create()' a ball on each iteration. And then I'm going to go ahead and increase my 'counter', which is what I forgot to do last time, and I'm going to increase it by '1' on each iteration. So, let's go ahead and run that code now. And as you can see there, we get '10' balls.

Now, we could go ahead and repeat this and add a '100'. And now, we're starting to have fun. Let me go ahead and make this a little bit smaller, and I'll go ahead and reload the page just so we can see a lot more. I'm going to go ahead and do it now '1000'. And as you can see there, we're starting to get more and more packed into the page. So, as you can see there, certainly having to do this by hand if we were to do a declarative approach and where we were writing the 'div' for each of those shapes would be very painful. And so, faster than that, of course, is writing the commands at the Console to be able to create each one of those. But if you're doing 1000s, that would still be painful.

Now, if you do it within the loop, you can see there how fast we can do that. And so, loops certainly are great to be able to carry out a large number of instructions. Now, remember; eventually, you might get stuck when it comes to the number of things that you display on the page if you exceed a certain number, and that's dependent on the power of the machine that you are on. You will get stuck. If you get stuck once again, don't worry, you can just restart. But play around and test the limits of what your browser can do and try to create some colorful shapes there. So, now it's your turn. Go ahead and try creating, as mentioned, in larger number of balls and try to see what you can do and push those limits.

Video 18 – Functions - Arrays Exercise

Next, let's take a look at an example that makes use of functions, and arrays, and loops at the same time. As you can see here, we're going to be using a function inside of arrays. This is part of the language. And we're going to be adding an element using push. Now, inside of that call to push, we're going to create a ball. And as you can see there, we will do it four times, which means that at the end of those four operations, we're going to have four items in that list. Now, once we have that list, we will be able to iterate through all of the items in the array and change the colors. And so, now let's go ahead and try this code at the Console. So, let's start off by creating the list of balls.

And we will start off by simply creating an array '[ ]' that doesn't have anything in it. Then we will go ahead and use the 'push()', as mentioned. And we will then call on create ball, the 'create()' ball function, which will return a ball, which will, we will then push into this array. So, we will go ahead and do that four times. So, as you can see there, we have four shapes on the screen. And now, I'm going to go ahead and iterate through these items in the array, which are all balls. And we're going to go ahead and change the color. I'm going to go ahead and clear here, my Console. And I'm going to go ahead and start off by creating a counter.

I will call this counter 'i', and I'm going to initialize it at '0'. I'm then going to go ahead and get the 'length' of the array. I happen to know what it is, but I wanted to show you a part of the language, which is a property in a race. And so, we can get that answer, as you can see here through that property. And so, now 'length', if we happen to look at it, is '4', which we knew already. But in this case, you might; you might consider the scenario in where you get some array that you don't know the length of or that potentially, the length has changed.

And so, you can always check the length of the array to determine how big it is. Now, I'm going to go ahead and write the 'while' loop. And I'm going to iterate as long as, ' i < length'. That is the number of items that we have inside of this array. And inside of it, we're going to get a handle on the ball. And we're going to assign that to a variable called 'ball'. And we will access the item on the array based on the counter. So, the first time we iterate through that, i will be 0. The next one, it will be 1 because, in a moment, we will increase it, and so on until we get until the end.

So, the next thing that we're going to do is we're going to 'colorRandom' because we want to change the color of the shapes that we have. And we're going to give it the ball that we have just, that we have just gotten from the array position. And then the last thing that we're going to do is we're going to increase our counter. A shortcut for increasing it is 'i++'. And that would work, but I'm going to go ahead and write it longhand, which is 'i + 1;' and now, I'm going to go ahead and hit return. And if all goes well, we should see our shapes here change color. So, as you can see, they're changing color and ending in a different one.

So, as you can see here, in this case, we're making use of the library. We're making use of arrays, some of the properties and functions in arrays, and then we're making use of loops. One of the advantages of having something in an array is that we can have a handle on all of those items.

Then we can iterate through that array using a loop and perform some operations. So, once again, we have gone a step further. We started off with declarative programming, then by writing statements on the Console, then by writing loops. And now, we're combining several of those components that we have discussed, which are many of the concepts of computational thinking of computation in general.

In fact, reviewing the code, we have actually used all of the core components of the language, which is variables. We created variables. We created arrays. We created conditional statements in our while loop. We created loops, which is our while loop. We called on functions, and we used all of them together in a pretty small example. So, now it's your turn. Go ahead and walk through this example, modify it, create one of your own. Push some of the limits of your understanding, and get comfortable. Because as mentioned, this is using all of those compound, all of those components, which are core to the language.

Video 19 – Functions - Objects Exercise

In addition to variables, arrays, conditional statements, loops, and functions, there's one more structure that is very important, and that is objects. And these are collections of data in key value pairs, as you see here. The definition is made within curly braces. And each of the items, as you can see, there are a key value pair separated by a colon. So, you can see there firstName : 'peter', lastName : 'parker', email : 'peter@mit.edu'. And each of those items, in turn, is separated by a ' , ' one from the next. We call each of these items properties.

And you can access those properties once you define the object, in this case, student, by using the dot notation, student.firstName, student.lastName, and so on. Now, with that in mind, we have created an exercise for you to combine everything that you have learned when it comes to the core components of the language. So, variables to raise, conditional statements, loops, functions, and objects. The exercise is a visual one, a graphical one. What we have done is we have taken a famous painting, not the one you see before you. And we have overlaid circles much smaller than the ones you see here in order to be able to get a better representation.

And within that circle, we have averaged out the color. And we have created all that data for you. And we have overlaid a large number so we can get a better recreation. And as you can see here, we have over 4,000 entries. And we have created an object out of each one of those entries. And you can see here an example. This one comes directly from the data. This is a ball, and you can see there the position for the x-coordinate, the y-coordinate, and the color of that position. Now, all of them together are put into an array called data.

And you can see there that it would hold ball1, ball2, ball3, all the way to the 4,000 that we have. And each of those are really an object, as you can see here. And you can see the data for three of them. You can see there that we have for the very first one, a position of x, which is 435, then a y-coordinate of 105, and the color, and I've listed here three. Once again, we have over 4,000. Now, before we go to the code, let me point out a few characteristics of the variables that we have

defined. And you can see there that you can get the length of the data by using the array property of length.

You can also access any one of those items by using data brackets notation and then the index that you're trying to access. And once you get a handle on any one of those balls, you can access using the dot notation, the properties such as x or y, and color. And once you have that, once you have any one of those, then you can use the create function to be able to place that dot with the color that you have extracted from that property. And so, if you loop through all of the balls, all of the objects, and they were array, the array called data.

Then you will recreate the painting. And so, this is a mystery for you to solve. Now, we have created all of the supporting functions and data for you in three files. Once again, as before, we have a ball.html, a ball.js, and a magic.js. All of these together hold everything you need to do. Everything that you need to be able to recreate the painting. So, let's move over to the Console, and I'm going to show you here some basics just to get you started. Now, as you can see here, I have the files that we have provided for you. I'm going to go ahead and drag and drop ball.html.

And all of that scaffolding, all of those functions, all of that data is now within the browser. And I'm going to go ahead and open up the developer tools. And I'm going to go ahead and type 'data'. And as you can see, there, data exist. And so, that's all of that information that we talked about. Now, if I where do enter 'data.length'. We would be able to get the length of that array. And if I were to access any one of them, and in this case, I'm going to go ahead and create a 'ball' here. And I access the position, the '400'th position, which is, was one of the examples that we had in the slides.

Now, I hold within 'ball' the object for that position. And you can see there that I have the 'x' coordinate, the 'y' coordinate, and the 'color'. And now, if I use the 'create()', I can use the dot notation to be able to pass in that information. So, I would have 'ball.x', I would have 'ball.y'. And then I would have 'ball.color'. Now, if this works, when I hit return, we should be able to see a ' . ' corresponding to those coordinates into that color.

And as you can see there, I do. So, once again, now your task is to be able to move through all of this data, to place those dots within the page. And if you do it correctly, you will be able to recreate the painting that we provided for you in data. This should be a fun exercise. You will be able to flex your muscles with all of the core components of the language added to the new structure that we're using, which is objects. So, go ahead and enjoy and have some fun.

Video 20 – Computational Thinking Summary

We have covered a great deal, and we've seen a lot of examples. However, it's important to map back to where we started when we talked about variables, about arrays, conditional statements, loops, and functions. And we talked about the analogs to the physical world, to our everyday lives. I encourage you to think about it in those terms. If you do, it's pretty accessible. It's nothing different than what you've been doing all your life. A lot of the syntax and a lot of the tools can be challenging, overwhelming, confusing.

But remember the concepts of computational thinking because that is what it's important. And if you can think of these terms, then you can execute in a way that you had impossibly considered before. And that is what this is all about. Those beautiful patterns, those beautiful ideas, those powerful abstractions, and these are the concepts and the components of every language. It doesn't matter what language you are on. Python, C++, or many others. They're all founded on the same concepts. And that is what is amazingly powerful and exciting about these platforms.

Video 21 – Simulation - Scheduling Computation

In this session, we're going to be touching on some of the concepts of simulation, where we try to model the physical world in a virtual way. Now, we will do a very simple application or a very simple representation of some of these concepts, and some of what we will use will be the following. We will use timers. We will leverage the knowledge that you gained from coordinate systems, especially within the browser from previous sessions. We will touch on some computational geometry. And when it comes to simulation, we'll discuss the topics of time and space and how do we move within these concepts using computation.

When it comes to thinking on how to simulate a physical system or how to create a simulation, one of the concepts that turns out to be pretty useful is the concept of being able to schedule execution or computation into the future. Now, within the language, there is such a feature, and the command that we use is called setTimeout. This allows you once again to schedule computation into the future, and that function call takes two parameters. The first of those is the function that you're going to execute. And the second one is the amount of time that is going to elapse before that computation gets carried out.

So, here's a simple example of what we have, a very simple function call sayHello, that's simply writes 'Hello World' to the Console. And then below, you see how that computation is scheduled. You use setTimeout, the first parameter, and simply the name of the function, which, in this case, is 'sayHello'. And the second one is the time to wait. Now, the units for that is milliseconds. And we're going to wait two seconds before that computation is carried out. Now, just as before, we're going to start off working on the browser Console. And so, I'm going to move next to the browser. And I'm going to open up the Console. I'm going to move to the browser. I'm going to navigate through the menus just like we did before.

I'm going to go to More Tools. And although the sub-menu appears on the screen, what I'm going to select is Developer Tools. Now, once you are on Developer Tools, the tab that you want is the one that says Console. And so, now, within this page here, I'm going to replicate the example that I had on the slides, and so I'm going to start off by writing a 'function', and all I'm going to do is call it 'sayHello( ){'. I'm going to then define the body of the function. What we're going to do, as mentioned before, was to sayHello. So, I'm going to answer that now '("Hello World!");'. Close that off, and I'm then going to close the function.

Now, the next thing that I'm going to do is I'm going to use 'setTimeout' to be able to make that computation, to schedule that computation into the future. And the first parameter is the name of the function, which, in this case, is '(sayHello' and then the amount of time to wait. In this case, I will make it '3000);'. So, I'm going to hit return. And a note about what you will see. You will see a

number, most likely, in this case, a one that is simply the ID for the timer. And later on, we'll discuss on how we could use this. But in three seconds after I hit return, we should see the "Hello World!" coming back.

And we're now waiting, we're waiting, and as you can see, there it is, 'Hello World!' Now, if we go back to our slides, we'll see that there is an additional notation that we can use. The notation is the following. We still use setTimeout. We use two parameters just as before, but instead of putting the function name, we could put the function directly into one of those two parameters. So, let me go ahead and illustrate that. I'm going to go ahead and clear here, my Console. And I'm going to go ahead and simply write 'setTimeout'. I'm then going to define the function. And so, in this case, I will simply write '(function sayHello( ){'.

And then I'm going to write the body of the function inside of it. I'm going to do the same thing as before. And then inside of that, the 'console.log', I'm going to write '("Hello World!")", then I'm going to close the quotes, close the parentheses, and that's the function. Now, the second parameter is the timer. Let's make it a little bit longer, but make it '4000);', and then I'm simply going to execute. Now, note that the first parameter is actually the function being written within the function call. And so, this is the equivalent of what we did before. I'm going to go ahead and hit return here. And if this works, we should see in four seconds, and yes, there it is. 'Hello World!'.

And whichever form you use depends on what's more comfortable to you; if you feel this is more compact and looks good, use this one; otherwise, use the previous way of defining. The computation to be scheduled into the future. And so, now it is your turn. This is a pretty important concept. This is a very basic timer. Go ahead and write an example of your own based on what you have seen. Open up the Console. As I mentioned, navigate through the menus, write your setTimeout. Give it your function; give it the amount of time that you want to wait, and play with it a bit in push on those concepts, make sure you understand them well.

Video 22 – Simulation - Scheduling Repeating Computation

The next thing that we want to look at is not just a scheduling computation into the future, but scheduling computation into the future that repeats. And in this case, the function that we will use or the command the instruction that we will pass is setInterval. This is part of the language, and just as before, we give it a function, and just as before, we give it a time. And in this case, the time refers to the amount of time that will elapse between subsequent calls to the same function. So, this will run as long as the computation is in scope.

That means as long as the page is open or unless you stop the timer. But we will leave stopping for the future. For now, let's concentrate on how setInterval works. As I mentioned, it is very much like setTimeout, the example we just did, except, in this case, it continues to call at that interval that we specify. So, let's go ahead and try out the same example that we did before and where we defined a function. So, we will go back to our Console, and I will define a function. Just for clarity here, I will repeat the definition.

Here is the function body inside of it, just as before, we're going to do is 'console.log'; inside of it we're going to simply say 'Hello World!'. And then we're going to close our quotes, close our parentheses, and below it, we're going to write our 'setInterval' this time, not setTimeout but setIinterval. The first parameter is the name of the function. So, in this case, it's '(sayHello,'. The second one is the time between calls to the function. Remember, this time, we're not calling only once, but this will continue to be called.

I'm going to set it at '3000);' I'm going to go ahead and finish that line, and then I am going to hit return. Now, what is coming back here is a static Hello, and the Console is smart enough to know that this is not changing. All it's doing, it's increasing the little counter here next to 'Hello World!, you can see there we're up to '5', '6', and it will continue to grow. And all it's saying is you're really riding to the Console the same thing over and over. So, instead of writing 'Hello World!' a bunch of times, all I'm going to do is give you a little counter next to it. So, you understand that that number is increasing.

Now, just as before, as opposed to defining the function separately and passing it in as a parameter, we can define the function within the call to setInterval. However, I will not demonstrate that here since it is the same thing that we did previously. Now it's your turn to schedule a repeating computation into the future. Once again, push on those concepts, make sure that what you think is taking place is, in fact, taking place. Play with that time. Try to do it quicker; try to do it faster. If you get stuck or if your page for some reason freezes, reload the page, and that will clear the memory of the Console, and it'll get you back to square one. If all else fails, close your tab, and if you're really, really stuck, you can shut down the browser and start over.

Video 23 – Maintaining State

Now, the previous examples that we worked on were pretty basic. There were static; nothing was changing. In fact, that's why on the Console, we got that little counter on the side because the Console is smart enough to know that nothing is changing. You're really pouring theme back that same message. However, when it comes to thinking about simulation, we often have a much richer environment. We're thinking through something that is changing. Often time, and especially within the examples that we will look at today, we're moving in time, and we're moving in space.

And so, this next example is looking at something where that function that is called is changing every time it's called. So, here is the example. As you see, we start off with the counter, then we have a simple function and then a setInterval. So, let's go ahead and write that on the Console. I'm going to go ahead and open up my Console once again, going to reload the page to make sure that I've cleared everything within the page. Now, I'm going to go ahead and write that code. I'm going to start off by creating a 'counter'. This counter is going to keep track of the number of times that my function that I am going to define next has been called.

And so, I'm going to enter 'function myFunction( ) {'. That's going to be the name of my function, and then going to write the body of the function inside of it. I'm going to have two instructions. The first one is to increase the 'counter' I'm going to do that using '++;' which means that every time

that function is called, the global value of the counter is going to increase by one. I'm then going to write to the 'console.log' here the value of that counter. And so, I'm going to enter here '("counter: " '. And then I'm going to ' + 'to it the value of my 'counter);', as you can see there.

So, that's all I'm going to enter for my function, a pretty simple function. And then I'm going to 'setInterval'. The first parameter is going to be my function name, which in this case is '(myFunction', and then I'm going to enter the time between function calls, which in this case will be '3000)'. So, note that this is different than what we've done before. There is a state that's changing, and we are accessing that at those regular intervals every three seconds in this case.

So, I'm going to go ahead and enter the command. And as you can see there, we have 'counter: 1', 'counter: 2' next, we should get 'counter: 3'. And so, this would continue to execute for as long as we wanted unless we reloaded the page or we stopped the timer. So, again, this is now more useful; we're keeping track, in this case, of how many times the function has been called. But as I mentioned, when we are trying to model something, there is a much richer environment and many more variables that, that we want to keep track of.

Next, as part of active learning, I want you to think about what's taking place, discuss it with your classmates. Can you explain what is taking place here? Can you explain how you could use this to model something you might want to think through? Could you take that same simple example and make it grow at a faster rate? For example, by five. Could you make it grow at exponential growth? And how could you take those concepts of growth and use them to think or model something in the physical world or that you otherwise want to model?

Video 24 – Moving in Time and Space

When it comes to simulation, there's a couple of variables that we often want to keep track of. One of them is time, and the other one is position. And these can be somewhat abstract concepts. And so, I'm going to try to ground them here with an example of the physical world. And in this case, I'm going to take the example of somebody who is jumping forward at a certain interval, at a certain time. And so, I'm going to go ahead and move forward. And this is the example, or it's meant to be analogous to one cycle of that interval, of moving forward in time a certain amount and moving forward a certain position.

And so, we could do that once again. And you can see here how we could model these variables within a timer and have some notion of something that's moving forward once again in time and also moving forward physically in physical space. So, we talk about time, and we talk about space. Now, the great thing when it comes to simulations is that we do not have to be bound to the physical world, which means that we could move very, very slowly in time or very, very fast. And we could also move through space at any speed or any rate that we like.

So, to be more specific, when it comes to the type of examples that we are considering within the code, let's consider the case in where we're firing every 3000 milliseconds and that we're moving forward every time five units. So, if you think about a spectrum of motion and also of time, we

would have the following, 3000 milliseconds elapse. We move by five. Once again, the next time those 3000 seconds elapse, we fire and fire, and we move, and we move, and we move. And so, you can see there where we can go from position zero all the way to 25. And we're also making those time jumps along the way.

Now, to ground some of those concepts, let's go ahead and do the following active learning. In this case, you're going to have a time, a position, a velocity. And the velocity here maps to how fast you are moving forward. So, at what rate are you accelerating or moving forward in physical space? And as you see there before you, you have 'myFunction()', which is increasing your position based on the velocity. And then it's riding that position to the console. And then you have your set interval that is going to be calling 'myFunction()' over and over again. So, let's go ahead and move to our Console.

I'm going to go ahead and reload the page. And as you can see there, the counter from the previous example continuous to run. But once I do that, it's going to stop because I have cleared the memory by reloading the page. So, let's go ahead and write that code. I'm going to start off by creating a 'time' variable. And this variable is going to be '3000' milliseconds. That's going to be the speed at which we are going to be calling this function. The 'position' that we're going to start off is at '0'. And then I'm going to set a velocity. This is the rate at which we will move forward of '5'.

So, a fairly small velocity. And when you want to see very smooth movements, you can go with a very small velocity. If you want to see big jumps, you would increase that velocity. Now, I'm going to define my function. I'm going to call here 'myFunction()'. Just as before. I'm going to write the function body. And the first thing that we're going to write, the first instruction is going to be the 'position', and that position is going to increase by the 'velocity;' on each function call. And then I'm going to write that to the 'console'. So, 'console.log'. And then within it, I am going to write the '("position: " ' to the console. So, I'm going to '+' to that 'position);'.

I'm going to close my line there, and then I will write my interval. So, 'setInterval', we'll give it my function name, which is '(myFunction'. And then I will write the 'time)' at which this function needs to fire at. And so, you can see there all of the parameters and all the variables that we have. We have time at 3000 milliseconds. We start off at position 0. Our velocity is that of 5. And so, every time that function is called, we move forward when it comes to our position. So, we're doing very much what we did in the slides of hopping every 3000 milliseconds. And we're moving forward in time, in this case, at the velocity of 5.

So, I'm going to go ahead and enter that command with those instructions, and you can see there that we start off at 'position : 5'. Then we've moved to '10'. We will move on to '15', and we move forward in that manner. So if we go back to our slides, this is precisely what we had visually. Now, we're executing it when it comes to code. So, once again, I want you to consider about what is taking place? How rich this environment could be? What you could use it for? How do you think about the physical world and its position here, something that is relevant when you come to modeling systems? And how could you use this in your own context?

Video 25 – One Dimensional Motion

Next, we're going to apply some of these concepts to the shapes that we define within the browser coordinate system. And to remind you what those are, remember that left is the horizontal axis, top is the vertical axis, and z-index would be Z. Now, what you're looking at here is the definition of a page that brings together two concepts, the declarative programming we did initially. You can see there the div that has all of its attributes from z-index all the way to the color of the background. But note that we're adding one attribute that is id equal ball.

And we will use that once we are within the script to be able to get a handle on that div and to be able to move it based on what we want. So, if we move to the bottom part of that code, you will see there that we are defining a velocity, a position just like we've done. And then we're also getting a handle on that ball. So, we use document.getElementById. We give it the id, which is 'ball', and from there on, we have a handle on the ball. We also see there that we have the position, which will increase by the velocity. And then we set the property of the ball, as you can see there, to be left.

So, we do ball.style.left, and then we make it =position+ 'px'; And the reason we use px for pixels is because that is what is required, as you can see here in the top declarative part when you set your left pixel position. So, in this case, here on top, you have '0px;'. And down here, we have whatever it evaluates to, and we will add that px to make sure that we are setting the right values for the left coordinate, which in this case is the horizontal axis. So, let's go ahead and load that onto the editor; in this case, we're no longer going to be working on the Console.

And we have given you some starter code, so you don't have to retype all of this here. And so, I'm going to go ahead and move to the class materials and download that zip next. And I'll show you the contents in a moment. And you can see here that there are a couple of files that we will be using. We have moveBall and moveBall_bonus. And so, I'm going to load first moveBall onto the editor. And so, I'm going to go ahead and move to Visual Studio Code, in this case, which is the editor I'm using. And as you can see here, you see the same code that we had in the page on the slides. And then you have a section there that says 'YOUR CODE'.

Now, in this part, I wanted to type this in just so you saw it come together. We're going to set the position, and the 'position' is going to be based on the 'position' that you currently have '+ velocity'. Remember, this is the rate at which you're moving forward, your position. And then we're going to set the style. And yes, you can see there we get the autocomplete, meaning that this editor recognizes that that is part of the browser. And so, this is a horizontal axis position. And it's simply going to be 'position +' We're going to add to it the symbol for pixels, which is ' 'px' '.

And the first time this runs, we should be at position 5, so we will barely move. We're starting off at the origin. Note that we are at position 'left: 0px;' and 'top: 0px;'. I'm going to go ahead and actually let me go ahead and start at '0' '0' here. So, no movement on the velocity. And so, we're simply; we should simply be at the origin. And so, I'm then going to load up the moveBall file. You

can see here I'm dragging and dropping onto the page. And you can see there that we have at the origin, the green circle. I'm going to go ahead and go back to my code.

And I'm going to go ahead here and move forward. I'm going to make my velocity much faster than I normally should if I want to see some smooth movement. But in this case, I simply want to make the point that we're moving. You can see there; if I go back and I move once again, you can see that I'm moving again. And I'm increasing by 100 every time. And so, I'm showing you the way this potentially, if you were using a timer, would increase. So, this is an example of doing it by hand. And the reason I want you to do this first is to make sure you're comfortable with the different pieces of the computation.

You have this first part, which is the declarative part, lines 1 through 9. Remember, we're defining here the idea of ball. Then we have a script block that defines a couple of variables. It then gets a handle on the shape, and then we can manipulate it. We're setting a position rate of increase here. We're adding the velocity to it. And then we're simply setting the position on the horizontal axis, as you can see. So, now it is your turn. Try it; make sure you can do it. And I've just given you the demo. And so, it is your turn. Go ahead and download that zip file. Play with the positions. Make sure you can do what I have done, and prove to yourself that you understand both the coordinate system and how you're setting those positions along the horizontal axis.

Video 26 – One Dimensional Animation

I'm now going to move back to the browser, and I will drag the solution onto the page. And as you can see there, the ball is moving across the screen on its own. It will continue to do so until it goes off the page. I should also note that I am moving at a specific velocity, and I could make it go much, much faster. Let me try to demonstrate that now. I've just changed my parameters, and now I'm going to reload the page. And as you can see there, in case you didn't see it, I'm going to go ahead and reload it again. It flies by. So, once you get it working, this is the next thing that you'd want to do, play with the rate of speed with which your ball will move.

Video 27 – Edge Detection

I just showed you a demonstration of what it would look like if you get everything working. Next, I'm going to show you the next set of functionality that we want to add to this moving ball simulation. The next thing that we want to do is we want to do edge detection. If you notice, the ball will continue to move off the page. And what we want to do is we want to set a certain limit. We want to detect the edge of a certain set of dimensions that, up to you, you set within the page.

And when you get to that edge, what you want to do is you want to reverse direction. So, you will move to the right until you hit that edge, then you will reverse direction and go to the left until you hit the other edge, you will reverse direction and go again. And so, I believe the next page shows a demo of this. This is a concept of what you will do. You're moving forward until you had the

edge, and then you will return, you will reverse your direction. So, let me show you what that looks like.

So, I've written a solution for edge detection, and I'm going to drag and drop that file. I've set the velocity to be pretty fast. So, we should see that ball moving pretty fast from edge to edge. And yes, you can see there high, goes back and forth, back and forth, back and forth. So, now it is your time to write the solution. This is one of those that once you get it, it's pretty simple, but it is not at all obvious what to do. So, think about it; think about it hard. Talk with your classmates if you get stuck.

Video 28 – 2D Animation Edge Detection

Once you master the horizontal axis, the x-axis, next, you can try to do two dimensions. In other words, in this case, you have to keep track of four conditions that you must reverse velocity for. You have your horizontal axis; you have your vertical axis. And in each place, you will need to place a limit. Now, we've given you a starter code that was within the file that you downloaded. Not the file, but within the zip folder that you downloaded. And you can see there that the first part looks the same. However, when it comes to the script, you now have velocityX, you have velocityY, as well as positionX and positionY.

So, in the horizontal case, you have a much more simplified environment. You only have to keep track of one dimension; in this case, now you have 2D. So, I'm going to show you next a demo of what it should behave like if you get it to work. And so, I'm going to go ahead and drag that now. And this should also move pretty fast. And yes, you can see there that it is bouncing both on the horizontal limits of the space that I have defined and also on the vertical limits as well. And so, you can see there that we have a pretty smooth simulation of a bouncing ball.

Video 29 – Bonus Exercise

Now, once you start down the road of additional considerations when it comes to this simulation, there's really a lot that you can do. Ultimately, you would end up with a physics simulator. But still, at a pretty simple level, here are some of the things that you could do. You could change your speed every single time it touches an edge. You could change a color. You could change directions; you could change sizes. You could have multiple of them bouncing around at the same time. And so, there's a lot that you could explore here.

So, I'm going to go ahead and show you here some possibilities of what you could do. In this first example, we have a change in color, as you can see there, once it hits an edge. On the next one, we have something similar, but we're changing the size. So, you can see there. The next one is the change in speed. You can see that it moves at one speed in one direction and a different one on the other. Here is a lot of different balls moving at the same time at different velocities, as you can see there. You can even take this and move it in the X and Y direction. Another thing that you could do is you could do edge detection with some of the other balls that are moving.

Meaning when you come into contact with one of those, you could add audio. There's so much that you can do. And the point here is not to try to replicate everything that I have shown you but to move through these concepts, make sure you understand them because they're illustrative of many of the things that you'd like to do in computation. And at a beginning level, when it comes to simulation, this can be a tool that you can use to understand a lot of what you might want to try to do. And at the very least, it's a way to explore many of the concepts that we have shared with you in this workshop. So, with that, go ahead and enjoy and try to push as far as you can on some of these notions.

Primer 1 – Debugging in Next Tech

While using Next Tech, you will inevitably run into situations where the solution code you write doesn't pass the task solution check. You must understand some of the types of solution checks that Next Tech uses and how to read their error messages best to get helpful clues on what part of your code is incorrect. So, let's walk through an example of a failed solution check and how best to debug the error messages it gives us. Here we have a Next Tech assignment that uses three types of the most used solution checks. First, is the code pattern test, which uses regular expressions to check whether you are using the correct code syntax and naming conventions such as variable names and methods.

The second is a jest test, which sets your code functionality, ensuring your functions return the correct value. And the last is a browser simulation test that checks the browser's content, ensuring you're using the proper HTML tags. As a developer, you will often have to debug your own code and write tests like the ones you will see here. Learning these skills will not only help you complete your assignments, it will make you a better developer. For this first step, our task is to add a paragraph HTML tag that will display the text < p >Hello World< /p > on the page. Let's copy the code it provides us and try running the first task to see the error message. The first thing we get is, "Make sure you're using the '< p >' tag to display the text "Hello World!" ." Looking at the code, I can see that I am using the correct < p > tag to display "Hello World!".

So, that's not the issue. To get more information about the error, I can click the caret icon to the right of the task, which will display the first step we saw previously when the task failed. And below that, I can see what code check this task is using. This task is using a code pattern test to check the solution. If we click on the caret to the right of the check, we will get more information about what that code pattern check is looking for. In this instance, we can see in the description of the check it says, "Searched your code pattern for a specific pattern: Hello World!". Now that we know what kind of check it is, and what pattern this code check is looking for. We can compare its Hello World! with our Hello World.

And we can quickly see that we are missing an exclamation point at the end of ours. If we add that and rerun the check, it should now pass. Now, let's move on to this assignment's second step and look at another type of Next Tech solution check. "In this step, our task is to modify the 'number2( )' function to return the number 2." Let's copy the code it provides us and run the first task to see the error message. The first hint we get is the message, "Make sure you are using the 'return' keyword inside of the 'number2( )' function." Before we look into the error details, let's use a common tool to make sure our code works using 'console.log'. 'console.log' will print whatever we put in between its parentheses to the browser's debug console when it runs, allowing us to test different parts of our code.

Let's add a 'console.log' and call our function inside of it. Then let's open up our Next Tech browser's debug console, by the "Open a new browser tab" at the top right of the Next Tech

browser. Now, in the newly opened window, we will right-click and select 'inspect' at the bottom. This will bring up our developer tools. There are multiple tabs here, and a tremendous amount of functionality, but we only want to focus on the Console tab right now. Here in the Console tab, I can see the 'console.log' is printing the 'number2( )' function value, which is 2. Since it appears on 'number2( )' function is returning the correct value. Let's switch back to the Next Tech window and dig deeper into the error message. Just like we did in the previous step and opened up the task to look at the check inside.

In this test output, there is a lot more than we saw in the code pattern test output previously. Right now, you likely won't be familiar with some of the terms you see, but the two important ones we're looking for is 'Expected' and 'Received'. Expected is the solution should the check is wanting to pass and Received is the solution it got from our code. If we compare the two, we can see the difference. It wants the numbered version of 2, and our code is returning the string version of "2". Now, let's change that and rerun the task. Now, let's move on to the assignment's final step and look at debugging a browser simulations solution check. A browser simulation test checks the HTML content in your browser. So, this evaluates the end result, not necessarily each line of code that you write.

While a browser simulation test won't be able to give you a hint on each line. We have grouped them to help you give a hint about what might be wrong with your browser. In this step, our task is to add an HTML paragraph tag that has a class name, a 'first-name' ." Let's start by copying the code provided and try running the tasks to see the error message. The first thing we get is the message, "Make sure your class name is spelled correctly and is in an HTML paragraph tag < p >< /p > ."

Looking at the code, I can see that I am using the correct paragraph tag and the class name is not misspelled. Opening up the task and the browser simulation check inside of it, we can see both the output and the test contents. The output isn't much help. But if we look at the test content, I noticed some type of method called 'find_elements_by_class_name', uses my class name. So, I know there has to be something wrong with my class name. If I copy the class name it's looking for and paste it next to mine, it's a lot more obvious that my class name isn't formatted correctly. I used an underscore instead of a hyphen. If I replace the underscore with a hyphen and rerun the task. It should now pass.