JavaScript in 12 easy lessons

A beginners guide to modern JavaScript

Nik Lever

JavaScript in 12 easy lessons

Nik Lever

Table Of Contents

1. Introduction

JavaScript is the most popular programming language in the world.

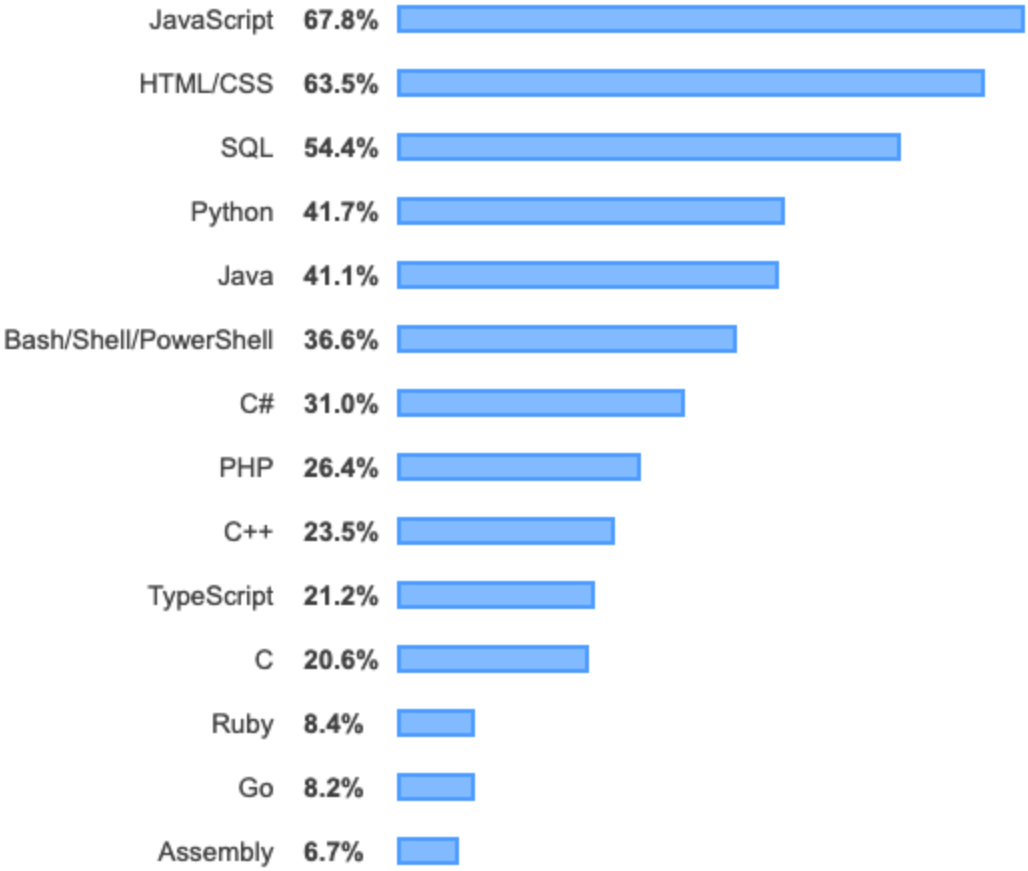

Diagram 1.1 – Programming language popularity

JavaScript, often abbreviated to JS, is the default language of the internet. But it doesn't stop there, JavaScript powers your smart tv, works with the internet of things, and helps build cross platform apps. VSCode, the program we'll use in this course to write the JS code, is written using JS.

Because JS is the default language of your browser, you can start to write JS code without installing any programs, you can just use your browser. Making JavaScript the easiest programming language for new coders. Having learnt the language, you can then more readily lend your skills to another coding language such as Python, Java or C++.

You'll be pleased to know that JavaScript is easy to learn. If you do one lesson a day, they vary in length from 10 minutes to an hour, then you will have a good grounding in the basics of coding.

Most of my coding has been for online games and you'll find a slant in the course towards solutions that suit game developers. But essentially the concepts apply to all developers.

JavaScript is super versatile, and you'll soon be using it to add cool dynamic elements to your web pages.

With JavaScript you're empowered to handle any aspect of app design. Often apps have front-end coding, the bit the user sees. JavaScript is ideal for that. But now with software called Node.js you can also use JavaScript to write the back end. Interacting with databases to deliver data to your user.

The most popular delivery format for data even has JavaScript in the name, JavaScript Object Notation (JSON) is the go-to standard for data exchanges on today's internet.

An increasing number of businesses and organizations rely on software for sales and production. Many using JavaScript as the principal language. There is a corresponding increase in demand for developers who are familiar with JavaScript. There's a shortage of good JavaScript developers, so you have increased demand in conjunction with a deficit of qualified people. The average salary in the USA for a JavaScript developer is over six figures.

If you're looking to become a programmer who can always be assured of a stable career and get well compensated for it, then you want to learn JavaScript. In this book I show you how.

2. Lesson 1 - Variables

All web pages that include client-side scripting, that is scripting that happens inside the user's browser, use JavaScript. In this book I hope to get you up to speed with writing JavaScript code. Writing code is great fun and the more you practice the better you'll get. I wrote my first computer program using a punch card device back in 1969 and I'm still learning new stuff every day.

The basic building block of all programming languages is the variable. Don't just read the words, open your Chrome browser. I recommend Chrome for this course. Then press F12. If you forget the keyboard short-cut you can also press the triple dot button at the top right, then select More Tools>Developer Tools.

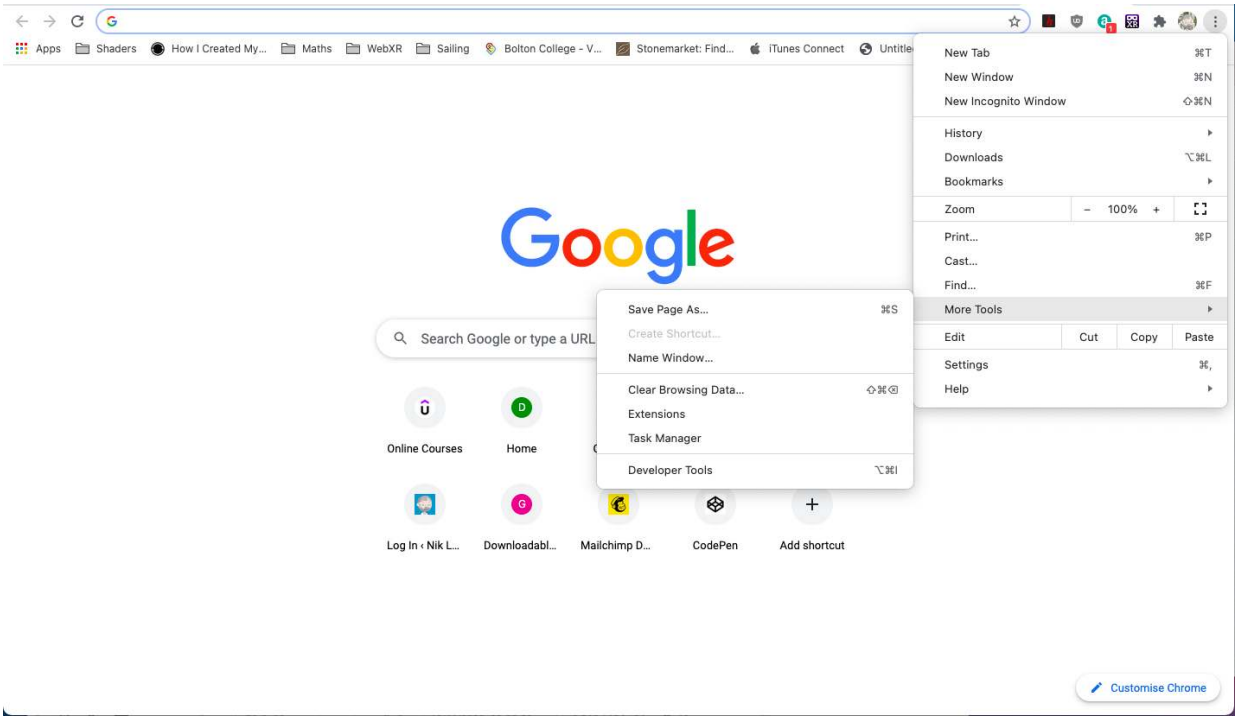

Diagram 2.1 – Opening Chrome Developer tools

You'll get a new window open. Select the Console tab.

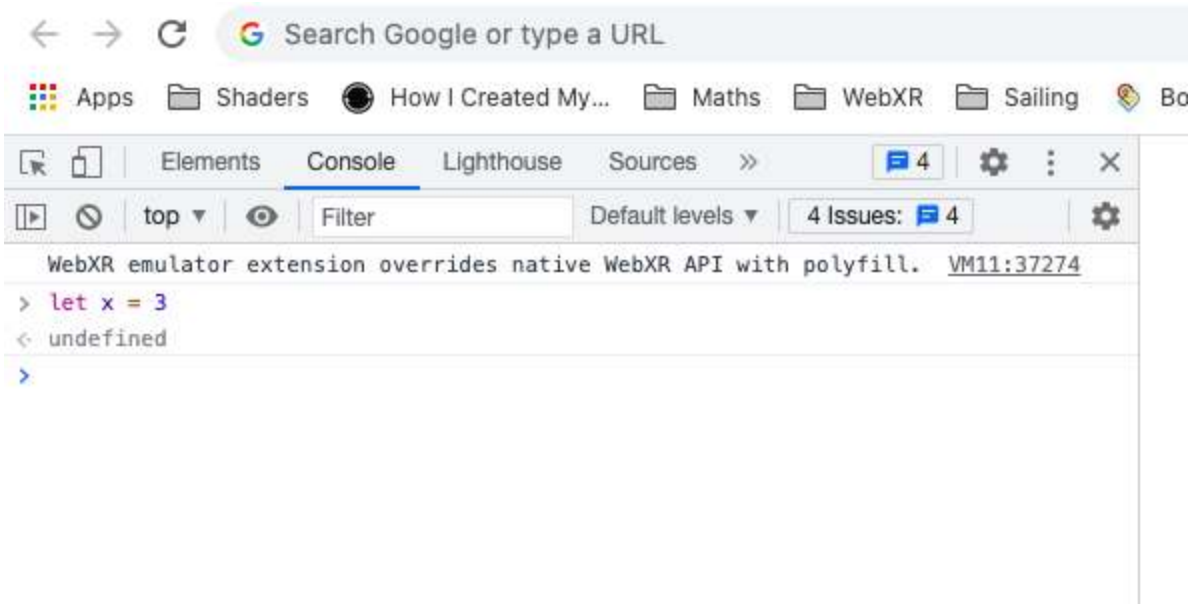

Diagram 2.2 – Using the console to enter code

Beside the greater than symbol, where the cursor is flashing enter

```
let x = 3
```

Then press enter, ignore the undefined that just means there is no value returned by this code. But you have created a little part of your computer's memory that you can access using the name x.

Enter

```
x
```

Now it shows the number 3. That is because the variable x has the value 3. Sometimes it's confusing about variable names and variable values. If it helps think of a pigeonhole system. Each pigeonhole has a name and inside each pigeonhole a number is stored. If we look in the pigeonhole named x the number 3 is stored there.

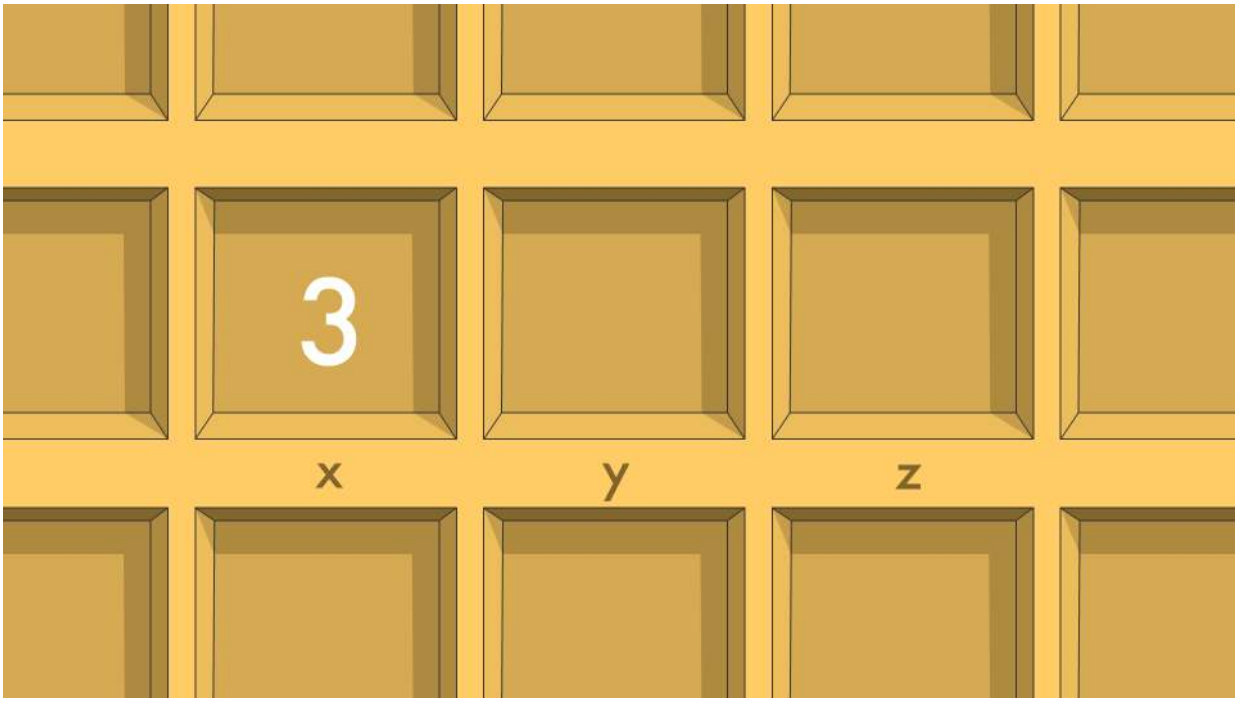

Diagram 2.3 – Thinking of a variable as a pigeonhole

Let's store a new value at the pigeonhole y, enter

```
let y = 5
```

Again, we get undefined. Why's that? Yes, assigning a variable does not return a value. What happens if we enter

```
x + y
```

You get 8, because adding together x and y does return a value. It is 3 + 5 which is 8.

What is y – x?

Yes, it is 2, because 5, y's value, minus 3, x's value, is 2.

Enter

```
let result = x + y
```

Variable names do not have to be a single letter. A variable name can contain letters, digits, underscores, and dollar signs. To see the value of result just enter result. It has the value 8. 3, x's value plus 5, y's value.

A variable is called a variable because it can vary. Enter

```
x = 12
```

Now the value in the x pigeonhole is 12 not 3.

Diagram 2.4 – Current values in the pigeonholes

Has this changed the value of result? Click the little eye icon and we'll add a live expression, enter result.

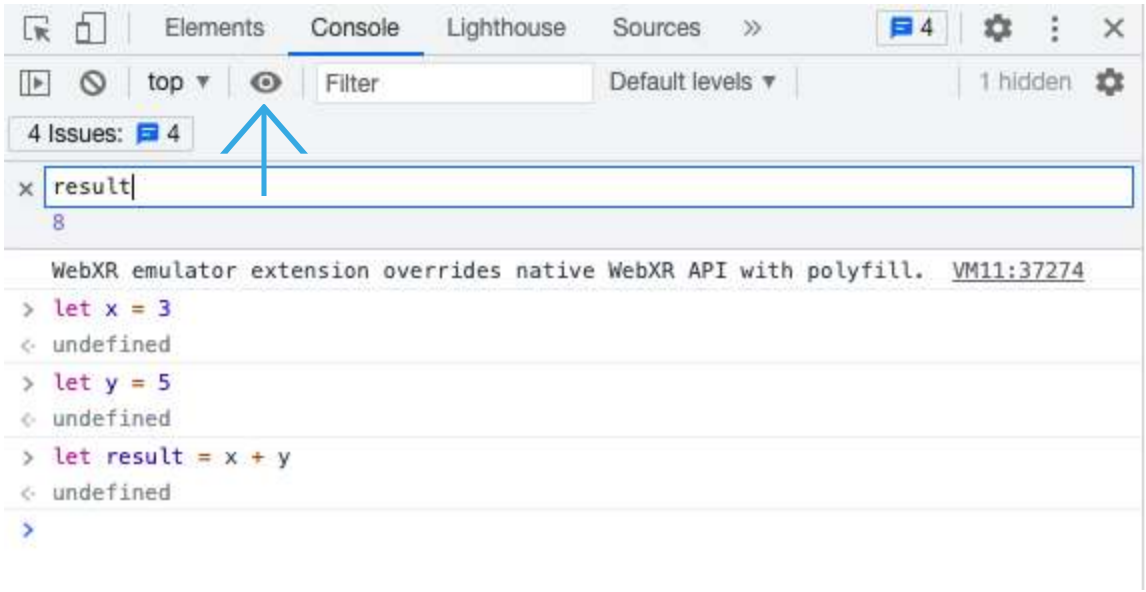

Diagram 2.5 – Adding a live expression

Now we can constantly see what the value result is. And it is still 8. Because once the value is set it will stay that way until it is updated. Even though it was set as x + y, it isn't tracking those variables. To update it we need to enter

```
result = x + y
```

once again. Now we get the value 17, x=12 plus y=5.

There is another way to declare a variable, in fact there are two other ways. Let's take them in order enter

```
const z = 7
```

By using const and not let, we create a variable that cannot be changed. The pigeonhole has a locked glass door, we can see the value, but we can't change it.

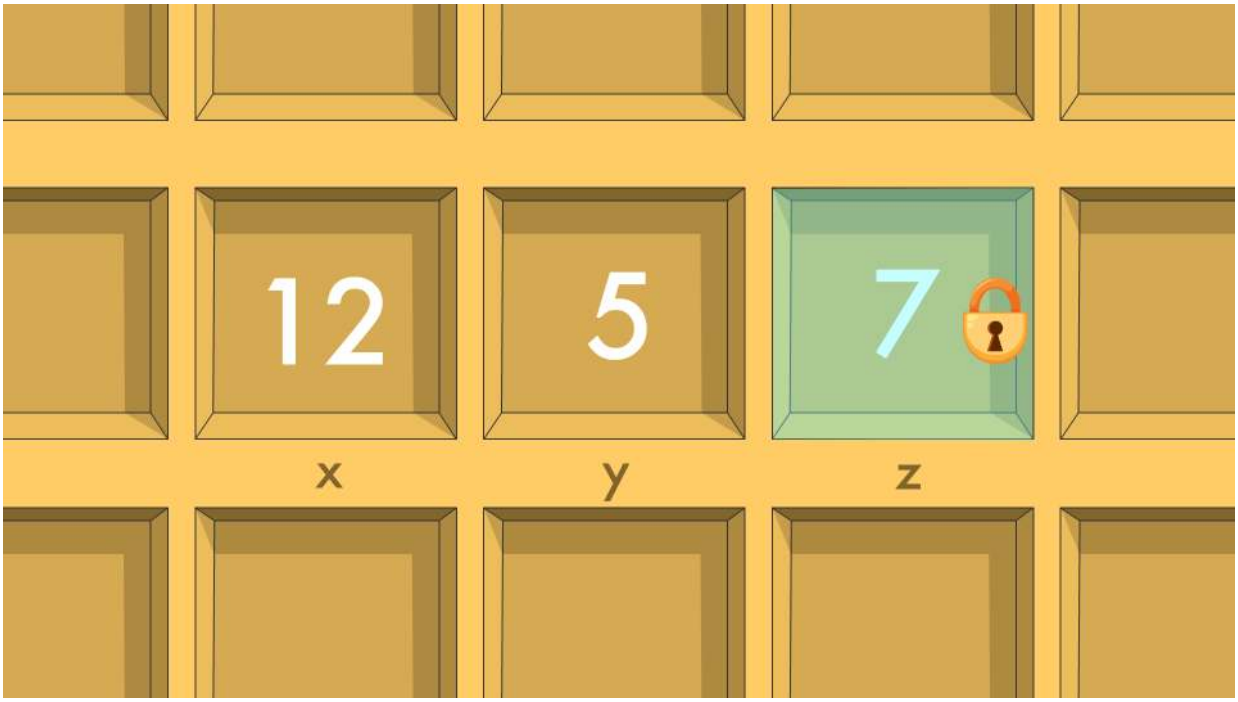

Diagram 2.6 – x, y and z in pigeonholes

Try entering

```
z = 8
```

You get an error message. 'Assignment to constant variable'. Constants can make your code more robust ensuring values that shouldn't change can't be changed.

I mentioned there is another way to declare a variable and that is to use var. In this course we won't be using var at all, it is the old school way of defining a variable and we're using the latest approach to use let and const.

So far, our variables have just been integer values, that is whole numbers. A variable can be a decimal value. Like let w = 4.78. Or it can be a word or a sentence. Enter

```
let myName = "Nik Lever"
```

Use your own name, and make sure to enter double quotes at the start and end. This type of variable is called a string and can be very useful. But what happens if we add together myName and x, a string and a number?

Because myName is a String the plus operation doesn't try to convert myName to a number, instead it converts x into a String and joins this on the end of myName, giving

```
Nik Lever12
```

Now a little challenge.

Can you create 3 variables, firstName, lastName and fullName?
When creating fullName use the variables firstName and lastName and put a space between them. Put the book down and give it a try.
Don't turn to the next page until you've tried the challenge.

For my name it will be

```
let firstName = "Nik"
let lastName = "Lever"
let fullName = firstName + " " + lastName
```

To check enter fullName to see its value.

Notice I used a capital N for name. Throughout the course I will be using what is called camel case. Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter, and in this course the first word will start with a lower-case letter.

Don't worry if you didn't get this right. The more practice you get the more likely you are to become an expert. In the next chapter we'll look at the things you can do with Strings.

3 Lesson 2 - Strings

In the previous video you met the special kind of JavaScript variable called a String. You created a string by surrounding a word or sentence with double quotes. But you can also use single quotes.

```
let sentence = 'The quick brown fox jumped over the lazy dog'
```

When you create a string in JavaScript it is converted into a data type called a string object. An object in JavaScript can not only store data, it can also be used to manipulate the data. What do I mean by that? Well, it has a series of properties and methods that we can use. An example of a property is

```
sentence.length
```

By using dot length, we get the value 44. That is the number of characters in the sentence, 'The quick brown dog jumped over the lazy dog'.

An example of a method is

```
sentence.startsWith('The')
```

Now we're asking if the string we've called sentence, starts with the string 'The'. We get the answer True. True and False are special kinds of number, called a Boolean, we'll see much more of them later. What happens if we use a lower-case T? You can repeat a previous entry by pressing the up arrow. Now we get the answer false.

The string object has lots of methods, one lets us convert each letter to lower case.

```
sentence.toLowerCase().startsWith('the')
```

Now we get true. See how we can chain together the methods. Here, up to the first dot, sentence starts with an upper-case T. Then we use the method toLowerCase, so after the second dot the string now starts with a lower-case t. When we ask if sentence starts with the, with a lower case t, the answer is true.

Often in code, you'll want to search through a string to see if it has a particular sequence of characters. As well as startsWith, there is also endsWith.

```
sentence.endsWith('dog')
```

Is true, but

```
sentence.endsWith('lazy')
```

is false

What if the word you're searching for is in the middle of the string. Then you can use includes.

```
sentence.includes('fox')
```

The answer is true. In programming speak we say the method returns true.

What if we want to know where the word 'fox' is in the sentence? Then we use

```
sentence.indexOf('fox')
```

Which returns 16, if we start from the beginning and count the letters, including the spaces, We find ourselves on the space before the word fox. That is because the first letter isn't index one, it is index zero. We can show it in the console using

```
sentence[0]
```

That shows the single character at position zero. If we swap the zero to 16, we get the f at the beginning of fox.

We can add another string to the string using +=

```
sentence += ". If the dog barked was it really lazy?"
```

+= is the same as sentence = sentence + '. If the dog barked was it really lazy?'

When we used the method indexOf, we included the word ‘fox’ in the brackets. All methods use brackets, it sets them apart from a property such as length that doesn’t need brackets. The word we supplied is called a parameter. We’ll meet parameters a lot in the course. The indexOf method is most often used with a single parameter like this, but we can use a second parameter. The sentence now has two examples of the word ‘dog’. If we use

```
sentence.indexOf('dog')
```

Then we get the return value 41. Which is the position in the string of the first ‘dog’. Suppose we want to find the position of the second ‘dog’. Then we add a second parameter, set to one more than the value previously returned. If the second parameter is 41 then the return value is 41. We’re setting the position in the string to start the search from, and it finds the word ‘dog’ immediately. If instead we set it to 42, then the return value is 53. This is where we find the second ‘dog’.

```
sentence.indexOf('dog', 42)
```

Sometimes we need to extract part of a string. For that we can use the method substring.

```
sentence.substring(53) >> "dog barked was it really lazy?"
```

This will return everything in the string sentence, from the character at position 53.

Just like indexOf, we can use two parameters. If we do the second parameter is the end index, where to stop accessing the string sentence. What value should we use to get the value ‘dog’ return? Put the book down and try it for yourself. Don’t turn the page until you’ve tried it.

Yes, 56 is the value to add.

```
sentence.substring(53, 56)
```

Diagram 3.1 Using substring

We can access a sub string from the sentence using a different method, slice. At first it seems identical to substring.

```
sentence.slice(53, 56)
```

Where it differs is when we use minus values for parameters.

```
sentence.slice(-5, -1)
```

Returns 'lazy', because a minus parameter will start at the end of the string and count backwards. Do this with the method substring and you get an empty string returned, because all parameter values below zero are changed to zero. So

```
sentence.substring(-5, -1)
```

Becomes

```
sentence.substring(0, 0)
```

With a start and end index of zero we get no characters returned.

Now that you know about variables it's time to start thinking about programming. In the next chapter you'll take your first steps.

4 Lesson 3 - Functions

A function is a series of lines of code in a block. You call a function by providing the name of the function followed by brackets. Sometimes you will provide values between the brackets, these are called parameters. A function can optionally return a value.

When we create a function in JavaScript, we use the keyword function followed by the function name, then an opening bracket. If we're using parameters then we enter them now, parameters are like variables, we supply a name. If we have more than one parameter, then we separate the parameters using commas. Once we've listed all the parameters, we enter a closing bracket. The keyword function, the function name and the brackets containing optional parameters are called the function definition.

Having provided the function definition then we enter an opening curly brace. At this stage we've entered the area where we add our JavaScript code. If the function returns a value, then we use the keyword return. This can optionally be followed by JavaScript code. When the running program encounters the word return, it will exit the function and return to wherever the function was called. The function must have a closing curly brace.

A few examples will help enormously.

```
function multiply( a, b ){
   return a * b;
}
```

In programming we use an asterisk to specify multiplication. And we end each line with a semi-colon. Now if we enter

```
multiply(2,3)
```

we get the answer 6. By entering 2 as the first parameter inside the function the parameter a will have the value 2. The second parameter is 3, so inside the function the parameter b will have the value 3. The function returns the value of a times b, which in this case is 2 times 3 or 6.

If instead we enter

```
multiply(12, 7)
```

we get the answer 84. Now a is 12 and b is 7. a times b is now 12 times 7 or 84.

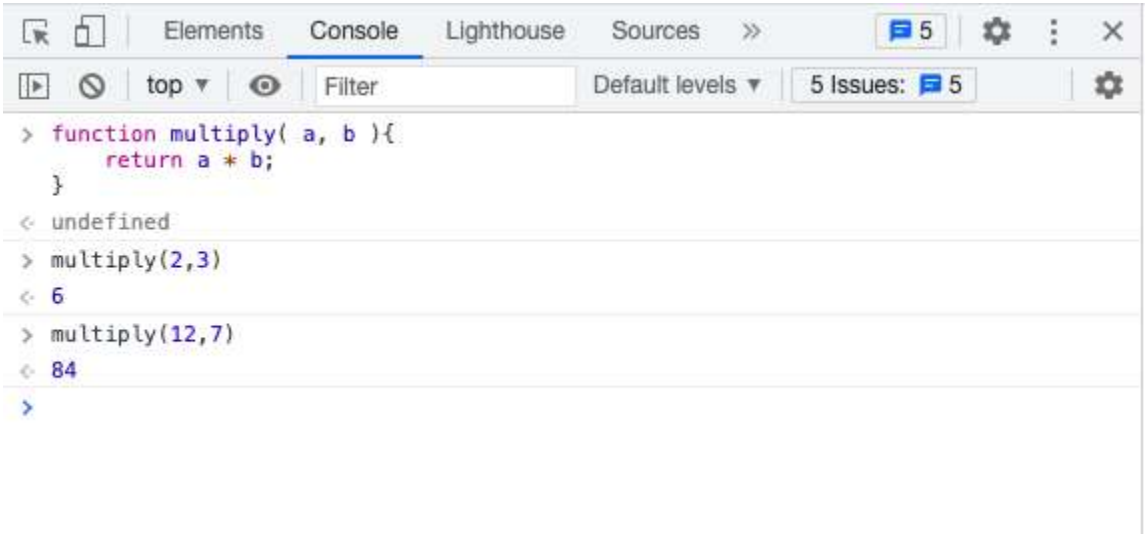

Diagram 4.1 – Introducing functions

When we create a function, we're creating a block of code statements that do a useful job and that we can reuse many times. We've met functions already in the course. The methods used by the String object, toLowerCase, startsWith, indexOf etc. are all functions.

In programming you will often find that angles are in radians not degrees. A full revolution in radians is 2π. Let's create a function that will convert degrees to radians

```
function degToRads(angle){
   const pi = 3.1416;
   return angle / 180 * pi
}
```

Notice here we have more than one JavaScript code line. A function can have many lines and may call other functions as it executes. Notice we use a forward slash for the division operation. Asterisk for multiplication and forward slash for division.

Since a full revolution should be 2π, a half revolution should be π. Enter

```
degToRads(180)
```

Here 180 is a half revolution in degrees and the function returns the approximate value for π that we used in the function.

Diagram 4.2 – Using the degToRads function

What happens if we just enter the function name, we get the function itself returned, not the result of its execution.

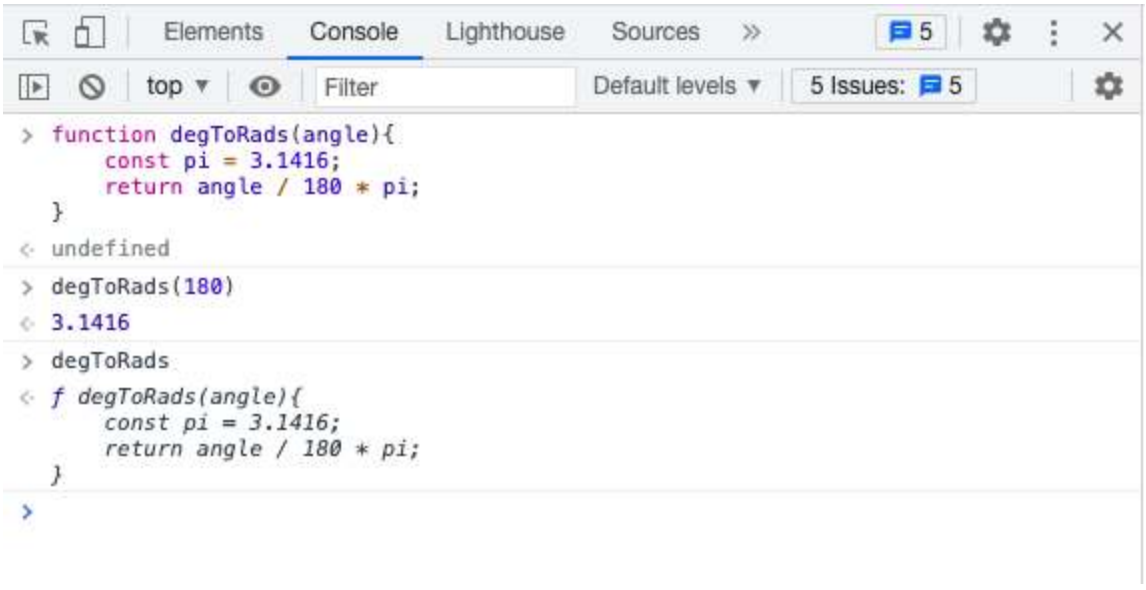

Diagram 4.3 – Returning the function

Notice inside the function that we created the constant pi. What happens if we enter pi. Do we get the value 3.1416? No, we get an error – 'Uncaught ReferenceError: pi is not defined'. That is because the constant pi only exists for code inside the function, it is described as local to the function. We'll learn more about this, it's called scope, in a later lesson.

Now you have learnt about functions ,can you create a function that will convert a distance in miles to a distance in kilometres?
A little tip, you need to multiply by 1.609 to get an approximate answer.

Put the book down and give it a try.
Don't turn the page until you've attempted this challenge.

My answer is

```
function milesToKilometres(dist){
   return dist * 1.609;
}
```

If we enter

```
milesToKilometres(5)
```

We get 8.045. Perhaps it is a bit too precise. Suppose we only want a single number after the decimal point. A Number is another JavaScript object. Just like a string the Number object has several methods. One is a function that will convert the number to a string with a specified number of digits following the decimal point.

Enter func and choose the function definition from the list. Now you can edit it. Surround dist * 1.609 with brackets and add

```
(dist * 1.609).toFixed(1);
```

Now if you enter

```
milesToKilometres(5)
```

the answer is “8.0”. Notice the double quotes surrounding the value, the return value is now a String. We want it to be a Number. So, we edit the line again like this

```
Number((dist * 1.609).toFixed(1));
```

The JavaScript execution will start with

```
dist * 1.609 = 8.045
```

Then we convert 8.045 into a string rounding the value to a single decimal place.

```
(8.045).toFixed(1) = "8.0"
```

Finally using Number we convert this String value back into a numeric value.

```
Number("8.0") = 8
```

Now you know about variables and functions. As your code becomes more complex you need a programming text editor and we'll look at that in the next chapter.

5. Installing VSCode and downloading the course resources

So far, we've used Chrome and the Developer Console to write our JavaScript code. But this means any code you write is lost when you refresh the page. Typically, JavaScript is written using a <script> tag in a html page or as a entirely separate file with the extension dot js. To write the html or js files we need a text editor. While you can use Notepad on windows or TextEdit on a Mac. For the rest of this course, I recommend VSCode.

Shoot over to https://code.visualstudio.com and install the application for your desktop.

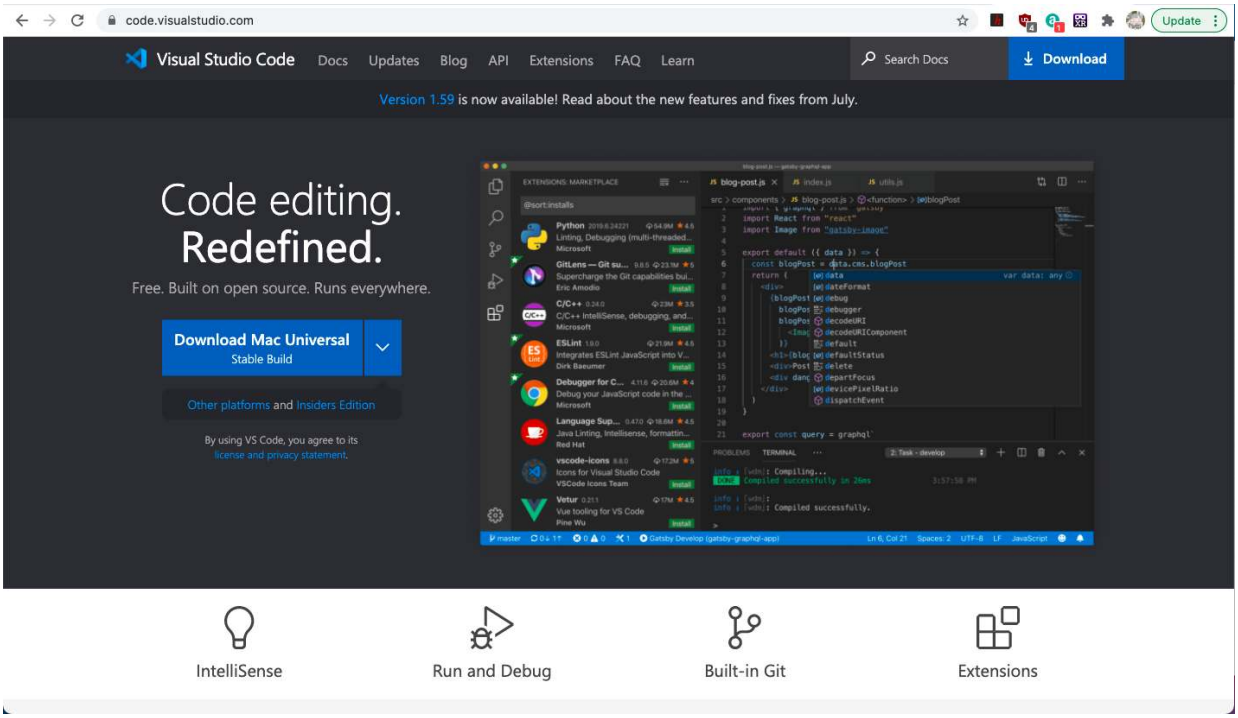

Diagram 5.1 – Download VSCode

Having installed the application. We need to grab the code for this course. Use the main menu, View>Command Palette … . In the search panel type 'git'. A list of options that include the word Git, appear. Choose 'Git Clone'. In the input panel enter

https://github.com/NikLever/JavaScriptLessons

Diagram 5.2 – Git Clone

You'll be asked where you want the downloaded files to be placed. Choose a folder, the course files will be placed in a sub-folder, named 'JavaScriptLessons', inside the folder you choose. Once downloaded you'll be asked if you trust the authors of the files in this folder. Choose yes.

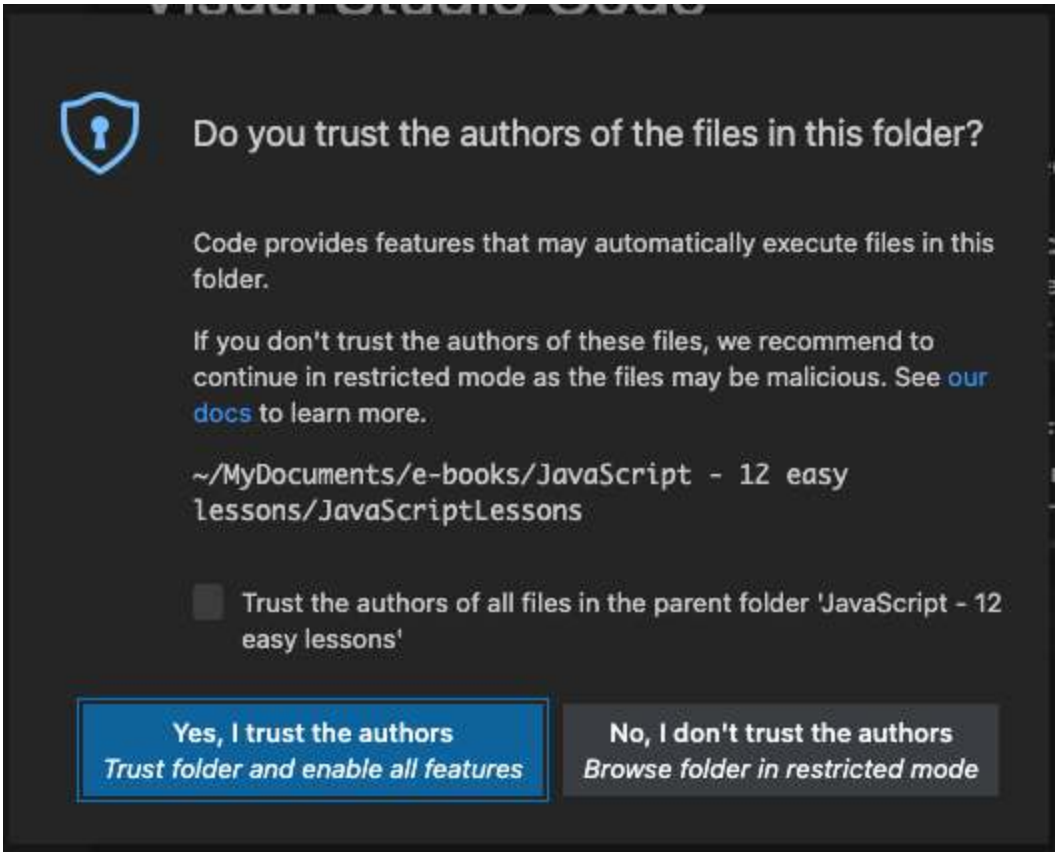

Diagram 5.3 – Trust panel

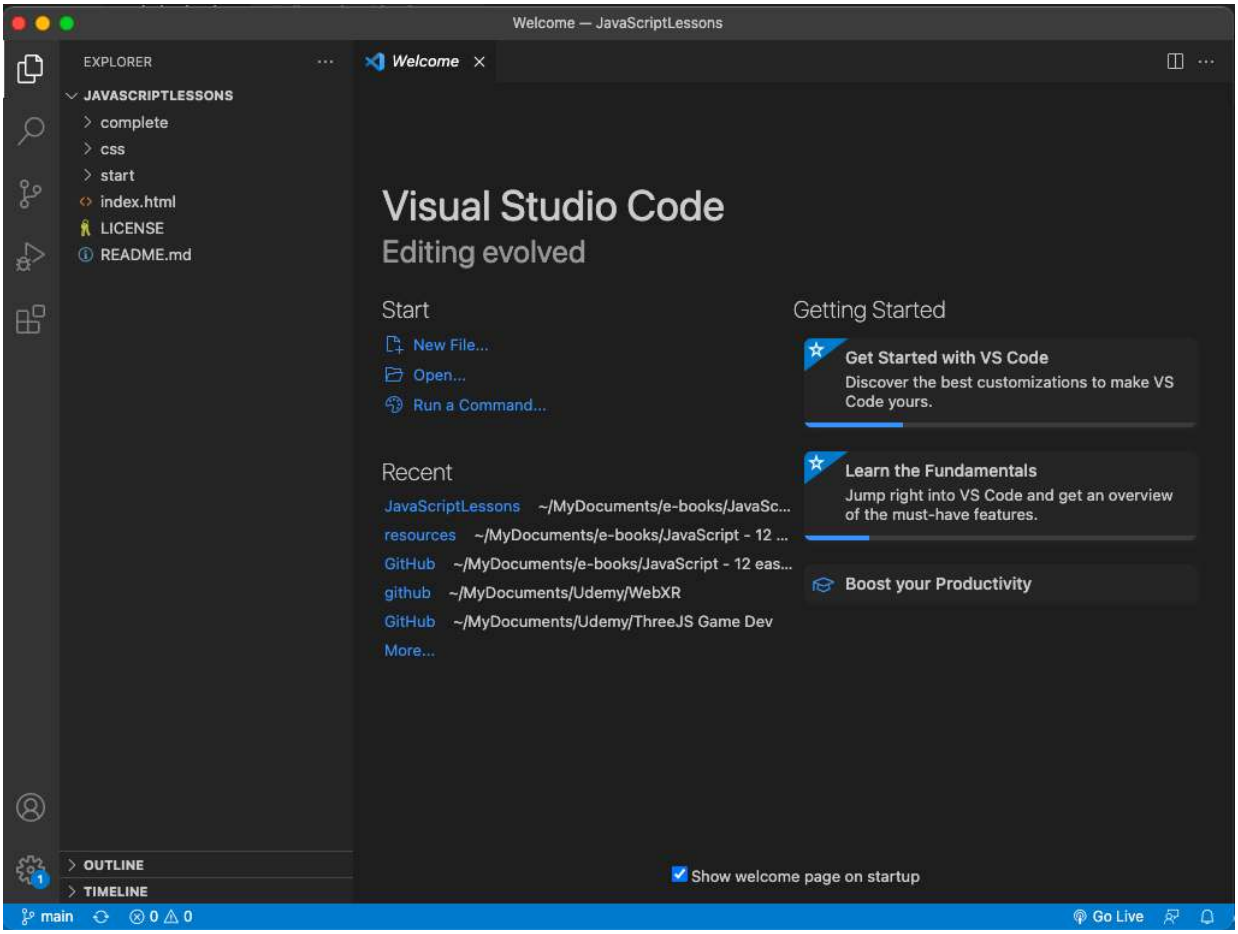

Diagram 5.4 – Explore screen

At this stage you should have a screen like the one shown in Diagram 5.4. Select the button at the top left to explore the folders and files in the resource repository you've just installed. Viewing the files in a browser requires one more step in VSCode. We need to install an extension. Find the Extensions button. It is in the toolbar with the Explore button and features 4 squares with the top right square offset from the others.

In the search bar enter 'live'. Find the entry 'Live Server'.

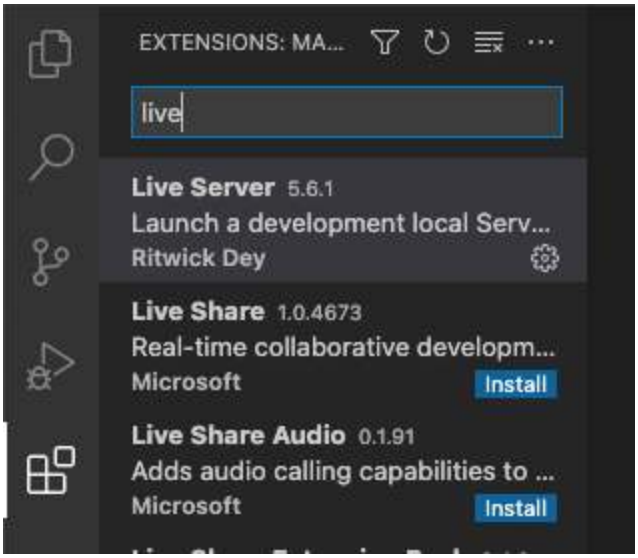

Diagram 5.5 – Adding the Live Server extension

In the right-hand pane choose install.

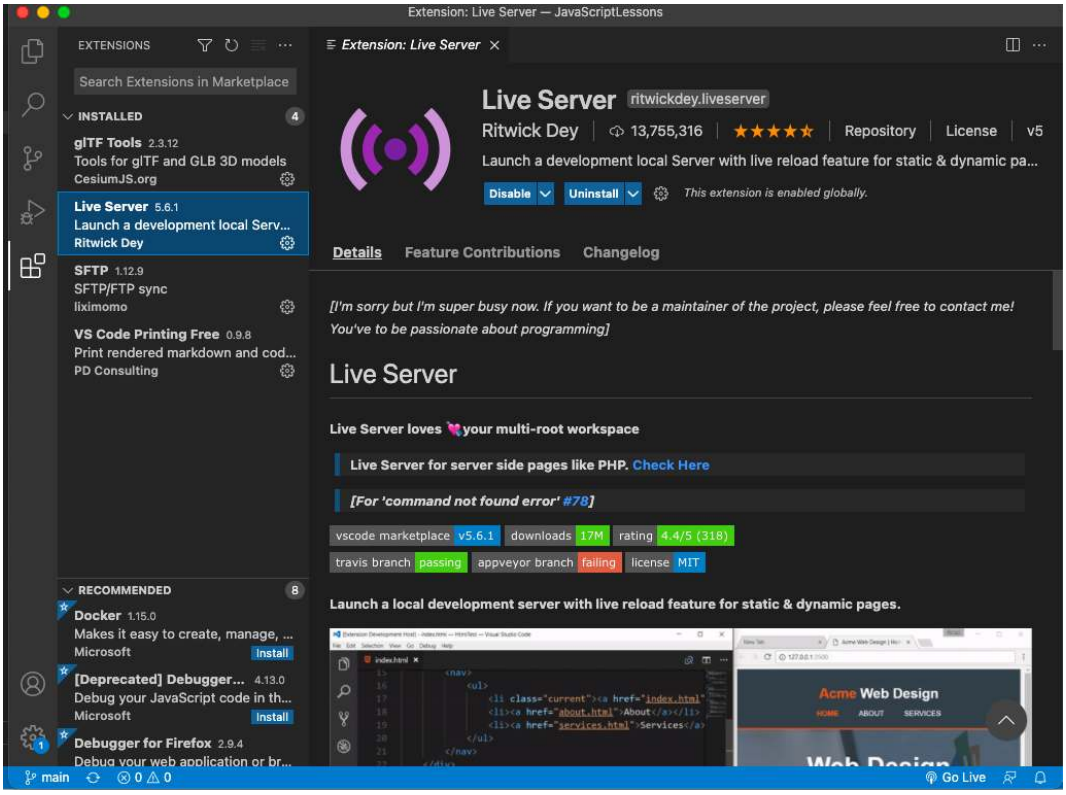

Diagram 5.6 – Installing Live Server

The last step is to find the file index.html using the Explore screen. Now right-click this file and choose Open with Live Server.

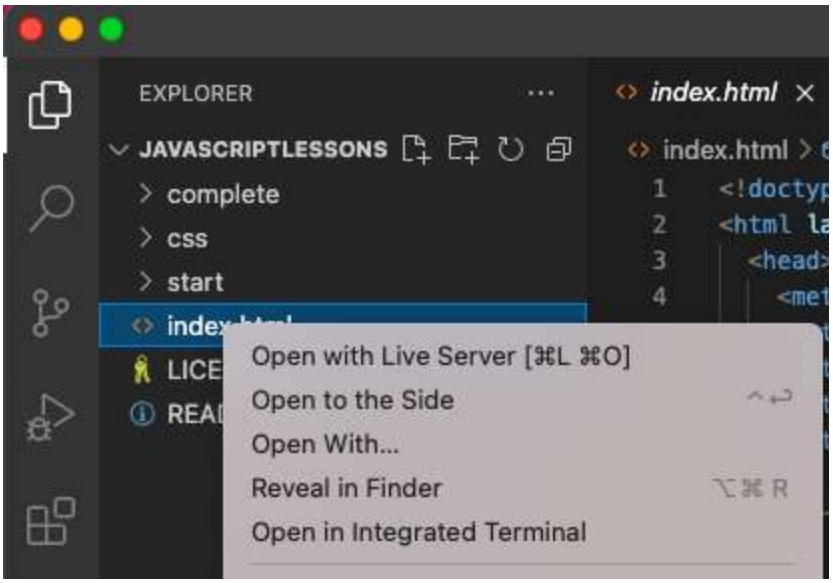

Diagram 5.7 – Opening a file in Live Server

Your default browser will open showing this index page.

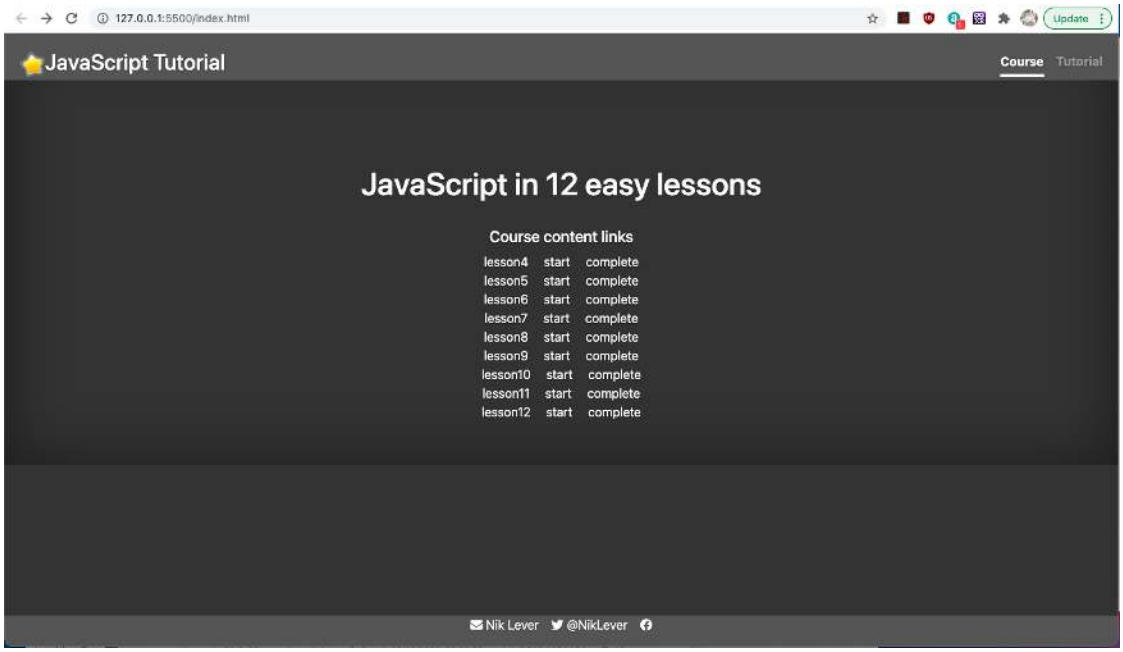

Diagram 5.8 – The default index.html page for the course

You're all set to carry on with the course. Great work.

6. Lesson 4 - Conditions

Hopefully you followed all the steps in the previous chapter and have VSCode installed, and you cloned the repository for the course. If you haven't then jump back and follow the steps to be ready to code along with this lesson.

If you open VSCode and open the workspace for this course and select the Explorer button. The top one on the left, you should see a file layout like this.

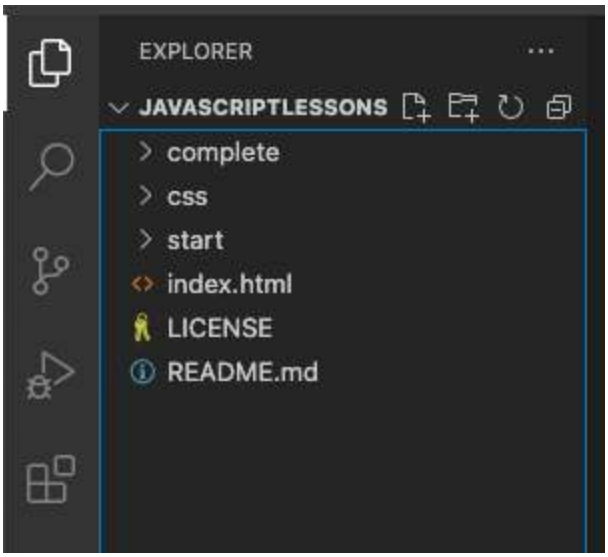

Diagram 6.1 – The course files

Notice the resources includes a complete and a start folder. For every example in the course the code along version is in the start folder and if you get stuck there is a complete version in the complete folder.

To work along with this video, I want you to open the file index.html in the folder start/lesson4. This is a very simple html page. I'm assuming you have some familiarity with HTML. An HTML page has a series of tags. A tag starts with a name inside less than greater than brackets and ends with the same name inside the same style of brackets only including a forward slash. Tags can be nested inside other tags. Here we have html as the outer tags. Inside the html tag is a head tag and a body tag. VSCode allows you to expand a tag. Something that can be useful as the code becomes more complex.

Before we enter any code, I'd like to show you how to view the pages. As we saw in the previous chapter, you should have the Live Server extension installed. If you haven't done this step, go back to the previous chapter and complete this step. Find index.html at the root of the resources. Right-click and choose 'Open with Live Server'.

In the browser you should be looking at the default index.html page. For this lesson select lesson2_4/start. You'll be totally unimpressed since all you'll see is a white screen. Bear with me here it will get better. Rome wasn't built in a day as they say. Now open the developer console. Remember the keyboard short cut? F12 to open Developer Tools, then select the Console tab. Now we're ready to enter some code.

Over in VSCode between <body> and </body>, add a script tag.

```
<script>
```

VSCode will automatically add the closing tag. Later in the course we'll use separate files for the JavaScript but in this lesson, we're going to add our code to the index.html page, for this the code must be in a script block.

We're going to enter a new function; we'll call it 'compare' and it has two parameters, a and b.

```
function compare(a, b){
}
```

Now the condition, this lesson is called conditions, and this is our first condition. We use the keyword, if, followed by brackets. What we enter inside the brackets must evaluate to a Boolean. Hold on, evaluate, Boolean, what does that mean? It means whatever we enter inside the brackets has to be either true or false. As an example, enter a is less than b. Angle brackets left is the less than operator. Following the closing bracket, we enter curly braces. Strictly speaking, if we only have a single line following the brackets then we don't need curly braces. But I highly recommend you always use them.

```
function compare(a, b){
   if (a<b){
   }
}
```

Inside the curly braces enter

```
console.log('a is less than b');
```

The complete code should now look like this

```
<script>
   function compare(a, b){
      if (a<b){
      }
   }
</script>
```

Remember to save your work. Then back in Chrome, let Live Server refresh the page and enter

```
compare(10, 20)
```

If you entered the code correctly you should get the response

```
a is less than b
```

In the console. What just happened? You wrote your first real JavaScript code. When you entered compare(10, 20). a is 10 and b is 20. The program reaches the line

```
if (a<b)
```

It reads this as

```
if (10<20)
```

10 is less than 20, so the statement inside the brackets is true and so the program moves to inside the curly braces. This prints the line a is less than b in the console.

Fine, but what if we enter

```
compare(20, 10)
```

Now a is greater than b and so we don't get a response in the console. We just get undefined because the function does not return a value.

When we have an if condition we can add a block of code that happens when the condition is false. To do that we use the key word else. Then curly braces and this time in the curly braces enter

```
console.log('a is greater than b')
```

The complete code is now

```
<script>
   function compare(a, b){
      if (a<b){
         console.log('a is less than b');
      }else{
         console.log('a is greater than b');
      }
   }
</script>
```

Save the file, refresh the browser and enter

```
compare(20, 10)
```

Again. This time you see 'a is greater than b'. But what if we enter

```
compare(10,10)
```

Again, we get a is greater than b, but a equals b. Why do we see the message a is greater than b? Because a is not less than b so the condition a<b in the brackets evaluates to false and execution jumps to the else block. We can fix this. As well as if and else blocks we can add an else if block like this. Before the else add an else if, notice the space between the words else and if. Now we can enter a new condition between brackets and a new code block between curly braces. In the condition enter

```
a==b
```

The complete code is now

```
<script>
        function compare(a, b){
            if (a<b){
                console.log('a is less than b');
            }else if(a==b){
                console.log('a equals b');
            }else{
                console.log('a is greater than b');
            }
        }
</script>
```

Notice in the new condition we have two equals signs. This is not a mistake. In JavaScript and many other programming languages two equals signs means we're testing for equality. Whereas a single equals sign means we're assigning a value to the variable on the left of the equals sign. Despite looking similar they are not the same at all and this is a common cause of bugs in code. Mistakenly using a single equals when double equals was required.

If you save your work and go back and refresh the browser. Now when you enter

```
compare(10,10)
```

You'll get the response 'a equals b'

What if we try to use a value that isn't a number? JavaScript will try to convert the value to a number.

```
compare("10", 10)
```

Will be fine.

But for

```
compare("ten", 10)
```

You'll get 'a is greater than b'. Not because a is greater than b, simply because a is not a number, so the JavaScript processor cannot evaluate a<b or a==b and so these conditions are evaluated as false.

At the start add a new condition

```
if (isNaN(a)){
   console.log( 'a is not a number' );
}else
```

isNaN is a built-in JavaScript function that tests if a value is not a number. For a string that can be converted to a number the function will return false. But if it cannot be converted the function returns true, the value is not a number.

The compare function now looks like this

```
<script>
        function compare(a, b){
            if (isNaN(a)){
                console.log( 'a is not a number' );
            }else if (a<b){
                console.log('a is less than b');
            }else if(a==b){
                console.log('a equals b');
            }else{ console.log('a is greater than b');
            }
        }
</script>
```

You know the form save, let Live Server refresh the browser, enter

```
compare("ten", 10)
```

Now we get ‘a is not a number’

But we can still get an erroneous message if b is not a number.

Can you fix this? You’ll need to add a new else if condition and code block. Put the book down now and give it a try. Don’t turn the page until you’ve tried.

You add

```
else if (isNaN(b)){
   console.log('b is not a number');
}
```

Giving

```
<script>
      function compare(a, b){
         if (isNaN(a)){
            console.log( 'a is not a number' );
         }else if (isNaN(b)){
            console.log('b is not a number');
         }else if (a<b){
            console.log('a is less than b');
         }else if(a==b){
            console.log('a equals b');
         }else{ console.log('a is greater than b');
        }
      }
</script>
```

Save, refresh and enter

```
compare(10, "ten");
```

Now we get

```
b is not a number.
```

Now we have four conditions and a catch all else. That is about the limit for using if.

There is another way of choosing a variety of options and that is to use switch.

After the closing curly brace of the compare function, enter

```
function today(){
   const day = new Date().getDay();
}
```

Here we create a Date object, we need to use the new keyword to create an instance of the Date object. If you create a Date object using no parameters, notice the empty brackets, then it will be initialised to the current date and time. A Date object has several methods, here we use the getDay method. This returns a number between 0 and 6, 0 means today is Sunday, 1 Monday all the way up to 6 meaning the day is Saturday. Now we have a numeric value we can use it to display a message indicating the day in English. We start with the keyword switch and then in brackets add the value we're using for the switch. Then we add case blocks. Case 0 followed by a colon, then one or more lines of code, here we just have a console log message. A case block ends with break, when the processor reaches break, it jumps out of the switch statement.

```
switch(day){
   case 0:
      console.log('Today is Sunday');
      break;
}
```

Can you add the remaining 6 case blocks? Put the book down and give it a try. Again try not look at the next page until you've tried this challenge.

Hopefully that wasn't too hard.

```
function today(){
        const day = new Date().getDay();
        switch(day){
            case 0:
                console.log('Today is Sunday');
                break;
            case 1:
                console.log('Today is Monday');
                break;
            case 2:
                console.log('Today is Tuesday');
                break;
            case 3:
                console.log('Today is Wednesday');
                break;
            case 4:
                console.log('Today is Thursday');
                break;
            case 5:
                console.log('Today is Friday');
                break;
            case 6:
                console.log('Today is Saturday');
                break;
            default:
                console.log('getDay returned
                             value out of range');
                break;
        }
}
```

Notice I also added a catch all default block, this will be used when no case block is true. Now save, refresh browser and enter

`today()`

You get the response 'Today is …" with whatever day it currently is.

Try to use a switch statement when an if else if else block has more than 3 or 4 conditions. It isn't always possible. It all depends on the problem.

There is one other use of conditions that I'd like to point out and that is the conditional operator. This is a short-cut version of an if else statement. It is used when we are assigning a value and it can take one of two different results based on a condition. An example will make it clear.

Let's create a function that will return a string saying if today is a weekend day or not. We'll get the day index as before using the Date object and the method getDay. Looking at the today function we can see that a weekend day means day is either 0 or 6. We enter return followed by brackets containing a more complex condition than we've met so far.

```
function weekend(){
   const day = new Date().getDay();
   return (day==0 || day==6);
}
```

Here we have 2 conditions joined by vertical lines. The two vertical lines means logical or, here we join day==0 and day==6 with an or operation. That means if either day==0 or day==6 is true, then the whole condition is true. Here is a table showing how it works for these two inputs

day==0	day==6	True	False
True		True	True
False		True	False

Table 6.1 – A truth table for determining if today is a weekend day

The column to the left shows the possible values for day==0 and the top row the possible values for day ==6. Notice if day==0 is true, row 2, and day==6 is false, column 3, then the result is true. Only row 3, column 3 gives a false value, when both day==0 and day==6 are both false. We'll meet more logical operators as the course progresses.

Now we enter a ?, the first thing after the ? will be the return value if the condition is true. If day is equal to 0 or 6. Then we enter a colon and finally the return value if the condition is false, day does not equal 0 or 6.

```
function weekend(){
   const day = new Date().getDay();
   return (day==0 || day==6) ? "It's the weekend" :
                               "It's not the weekend"
}
```

Save, refresh browser and enter

```
weekend()
```

The return string you get depends on the day you are watching this video.

Conditions are widely used in programming to control the flow of the program.

Now you know about variables, functions and conditions. Next up are Arrays, a special kind of variable that can be very useful.

7. Lesson 5 - Arrays

Arrays are a great way to keep lots of data in a list. To code along with this lesson in VSCode open index.html from the start/lesson5 folder.

You create an Array just like you create a variable. An array is just a special kind of variable. Unless you plan to keep recreating the Array then I recommend always using const for an Array variable. To create the Array just use square brackets.

```
const planets = [];
```

Now we have an Array. But the array doesn't contain anything. A JavaScript array has a property length that returns the number of items in the Array. Save your work, go back to Chrome and navigate to lesson5/start. Now enter

```
planets.length
```

This returns 0.

Let's add some items. Here each item is a string, but we can mix-up items in an array, we can have any data type as an item.

```
const planets = ["Mercury", "Venus", "Earth",
"Mars", "Jupiter", "Saturn", "Uranus"];
```

If we go save our work, go back to Chrome, refresh and enter

```
planets.length
```

Now the answer is 7. If you know your planets, you'll realise I've missed one out. Not a problem, we can add an item to the end of the array, even though the variable we're using is a constant.

```
planets.push("Neptune");
```

Let's create a console log that displays the number of planets in the solar system.

```
console.log('There are ' + planets.length + ' planets in our
Solar System');
```

Back in Chrome refreshing the browser and we get this message. See how by adding Neptune using push, planets.length is now 8.

When you have an array, you can access a single item using square brackets and an index value. Just like the first character in a string is index 0, so to the first item in an array is index 0, so despite the fact that we're the third rock from the sun, to get the item 'Earth' we use index 2.

```
console.log("Our planet is " + planets[2]);
```

If we save and refresh, we now get two statements about the planets.

Diagram 7.1 – Planet facts

If we add another item to the end of the array using the method push and another console message.

```
planets.push("Pluto");
console.log('Until 2006 there were ' + planets.length + '
planets in our Solar System');
```

Save and with live reload enabled with Live Server, we see another fact.

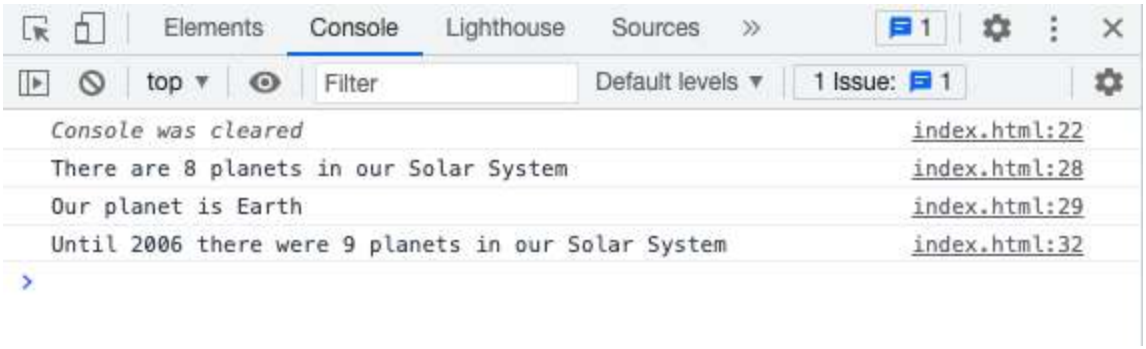

Diagram 7.2 – More planet facts

As well as adding an item using push, we can remove the last item from an array using pop.

```
const pluto = planets.pop();
console.log('In 2006 ' + pluto + ' was declassified as a
planet. Now there are ' + planets.length + ' planets');
```

Great work. As the course progresses, you'll see more things you can do with arrays. A key feature is the ability to access an item in an array using an index value. Recall in the last lesson we used a switch – case statement to display the current day as an English string. A much nicer technique is to use an array. The Date object has a method getMonth that returns an index from 0 to 11. Can you create a function, called month, that returns the current month as an English name? You'll need to create an array of months, get the current month index, then use this to return the right item from the array. Put the book down and give it a try. If you're a complete beginner with programming and JavaScript, it might seem quite hard. But there is no substitute for practice.

First, we create the function definition, that's just the function keyword and the name of the function followed by brackets. We have no parameters to pass to the function so that completes the definition.

```
function month()
```

The content of the function goes in curly braces. Now we create an array, we'll call it months. Remember we define an array using square brackets. Then we list each month, these are all strings so each item goes inside double quotes, you can optionally use a single quote, the apostrophe character. Each item is separated from the previous item using a comma. Now we get the month id by creating a new Date object, no parameters are included so the Date will be set to today's date and the current time. Then we use the getMonth method. At this stage the constant variable monthId will be a value between 0 and 11. All that remains is to return the item months[monthId]. The item in the array at the index month id.

```
function month(){
   const months = ["January", "February", "March", "April",
"May", "June", "July", "August", "September", "October",
"November", "December"];
   const monthId = new Date().getMonth();
   return months[monthId]
};
```

Save and allow Live Server to reload. Admire all the interesting facts about the planets once more, then enter

```
month()
```

And the current month will be displayed in English. If you'd prefer a different language, then simply replace the content of the months array with the names of the months in your preferred language.

Arrays are very important in programming. Another important data type is an object and we'll look at them in the next lesson.

8. Lesson 6 - Objects

You've learnt about variables and arrays already. Imagine you have a game where there are lots of non-player characters, NPCs. You want to be able to store information about them. Power, weapon, strength and damage for example.

We could store this as lots of individual variables. NPC1Power, NPC1Weapon, NPC1Strength, NPC1Damage, NPC2Power, NPC2Weapon, NPC2Strength, NPC2Damage etc. But, one of the really great things about JavaScript is just how easy it is to create a data structure that stores data in just the way you like it. If you're new to programming, then you won't understand how useful this is. Trust me it is. Instead of using four variables to store the data for each NPC, we can create an object that stores the variables in a single structure.

Like this

```
{ power: 0.5, weapon: "sword", strength: 1.2, damage: 0 }
```

We use curly braces and name-value pairs. A name value pair is separated using a colon.

Open index.html from the folder start/lesson6 to code along with this lesson. Let's create a function that will generate an object like this. It's a function so we use the function keyword. We'll call the function createNPC. The first thing we'll do is generate a power value, it will take a value between 0.5 and 1. So far in the course we've met the Date object. Another vital object is the Math object. This has many useful methods; we'll meet two of them as we create this function. The first method is random. This returns a pseudo random number between 0 and 1. The number could be any decimal value in this range. But we want a number in the range 0.5 to 1. So we multiply this value by 0.5, now we have a number in the range 0 to 0.5. This is still not what we want, we need to add 0.5, to move the range to 0.5 to 1. If this is confusing, imagine that Math.random returned 0.8, we multiply this by 0.5, now the value is 0.4, then we add 0.5 to get 0.9, that's in the target range. What if Math.random returned 0.1, we multiply by 0.5, getting 0.05, then we add 0.5 to get 0.55, again in the range 0.5 to 1. Math.random is a great method to use for random assignment of object properties.

```
function createNPC(){
   const power = Math.random() * 0.5 + 0.5
};
```

Next, we create an array of weapons. Then we select a weapon at random using this array. How do we do that? If we use Math.random() times weapons.length we'll get a value between 0 and weapons.length. Actually it will always be just less than weapons.length, because Math.random never returns 1, it can return 0.99999, but never 1. With Math.random time weapons.length we have a decimal value, but to select an item from an array we need an integer value. We wrap the code inside another Math method, floor. This returns the integer part of a decimal value. So, Math.floor(2.85) equals 2 and Math.floor(1.289) returns 1. Now we have a value that depending on the return value of Math.random returns one of the integer values 0,1,2,3. If the value is 0 then weapon will be set to 'sword', if its 1 weapon will be 'pistol', 2 – 'axe' and 3 – 'spear'.

```
function createNPC(){
   const power = Math.random() * 0.5 + 0.5;
   const weapons = ["sword", "pistol", "axe", "spear"];
   const weapon = weapons[Math.floor(Math.random() *
weapons.length)]
};
```

Can you set strength? We want a value in the range 1 to 3. Use Math.random, a multiplier and add a value. Put the book down now and give it a try.

It will be Math.random() * 2 + 1.

```
function createNPC(){
        const power = Math.random() * 0.5 + 0.5;
        const weapons = ["sword", "pistol", "axe", "spear"];
        const weapon = weapons[
              Math.floor(Math.random() * weapons.length)];
        const strength = Math.random()*2 + 1.0;
}
```

Damage is even easier, we simply default it to zero.

```
function createNPC(){
         const power = Math.random() * 0.5 + 0.5;
         const weapons = ["sword", "pistol", "axe",
"spear"];
         const weapon = weapons[
               Math.floor(Math.random() * weapons.length)];
         const strength = Math.random()*2 + 1.0;
         const damage = 0;
 }
```

Creating an object from these values is just a case of entering curly braces then listing them separated by commas. This will be the same as using these name value pairs.

```
function createNPC(){
        const power = Math.random() * 0.5 + 0.5;
        const weapons = ["sword", "pistol", "axe", "spear"];
        const weapon = weapons[
             Math.floor(Math.random() * weapons.length)];
        const strength = Math.random()*2 + 1.0;
        const damage = 0;

        return { power, weapon, strength, damage };
 }
```

This function will return an object like this

```
{ power: 0.5, weapon: "sword", strength: 1.2, damage: 0 }
```

We can create an array to hold our NPCs. Unsurprisingly we'll call it npcs. Then to add an npc to the array we use push, with the function call in brackets. This is going to call the function createNPC which returns an npc object, this is pushed onto the end of the array. If we repeat this 3 times, we should have an array of npcs.

```
const npcs = [];

npcs.push(createNPC());
npcs.push(createNPC());
npcs.push(createNPC());
```

Save your work, if Live Server is running, you can view the page using this url http://127.0.0.1:5500/start/lesson6/index.html

Enter

```
npcs.length
```

You'll get the response 3.

Now enter

```
npcs[0]
```

If you press the little arrow you can see more clearly the 4 properties of the first npc in the array.

Diagram 8.1 – Viewing the npc properties

If we're just interested in what weapon the first npc has we use

```
npcs[0].weapon
```

or to get the power value of the second npc we use

```
npcs[1].power
```

Everything about each npc is encapsulated in the single object. Just as we did with the string object, we use a dot to separate the object from the property. Later in this section you'll see how you can even add your own custom methods.

Let's look back on what you've learnt so far in this course.

First you discovered variables. Then you learnt about functions. You learnt how your code can branch in different directions using conditions. You learnt to keep lots of variables in a list using arrays and now you know about keeping lots of variables together in an object. With arrays you access an individual item using an index value and with an object you use a property name.

Often in code you will want to repeat a section of code several times. Like in this lesson where we repeated creating npcs. Rather than just pasting the same code three times like we did here, much better to use loops and that's what we'll do in the next lesson.

9. Lesson 7 - Loops

You're becoming a real programmer now. Remember in the previous lesson we created 3 NPCs. But we did it by copying the line and pasting it in two more times. In this lesson we'll look at methods to loop through code automatically.

The most common method is to use for. To code along open index.html from the start/lesson7 folder. You might be surprised to see this is where you got to in the previous video. I want you to delete two of the npcs.push lines.

Now before the remaining npcs.push line add for then brackets, followed by an opening curly brace. With the closing curly brace after the npcs.push line.

```
<script>
      const npcs = [];
      for(){
        npcs.push(createNPC());
      }

      function createNPC(){
        const power = Math.random() * 0.5 + 0.5;
        …
```

So far so good. But if you try and run it now, you'll get an 'Unexpected token' error. That's because we are expected to enter something in the brackets after the for keyword. What we enter for a for statement is positively weird. Rather than a typical set of parameters that you pass to a function, separated by commas. Instead, we enter 3 JavaScript statements. As such they are separated by semi-colons, just like you should end every JavaScript statement line. You enter three statements and each one, in order has a different purpose. The first is executed just once, before we run the code inside the curly braces, the second statement is a condition, while it evaluates to true the loop will repeat. The final, the third statement is executed every loop, after the code inside the curly braces has run. So what do we enter to cause the code block inside the curly braces to be executed 10 times. Creating 10 npcs.

First we define an index variable, we use let not const because we are going to be updating this variable. The second statement is a condition, we want to check if i is less than 10. Finally we want to add one to the variable i each time the code block executes. JavaScript has a short cut method for adding one to a number. We use plus plus. You might have seen there is a programming language called C++ and this is meant to indicate that it is the next iteration of the C programming language.

```
<script>
        const npcs = [];
        for(let i=0; i<10; i++){
          npcs.push(createNPC());
        }
```

Let's go over that again. This code means. Create a variable called i that is initialized to zero. We test if i is less than 10, it is, so the code inside the curly braces is executed. Then the final statement in the for definition is executed. The variable i is incremented by 1. Now i equals 1. The processor checks if i<10, it is, we run the code inside the curly braces and then add one to i. i now equals 2, we repeat until i equals 10, this is not less than 10. So, we jump out of the for loop.

If we save and visit start/lesson7. Remember you can right-click start/lesson7/index.html and select Open with Live Server. Or if Live Server is running enter http://127.0.0.1:5500/complete/lesson7/index.html in the browser address bar. Or use the root index.html page with Live Server, this allows you to return to the index page after a lesson. You would select Lesson 7>Start using this option. Whichever method you choose make sure the console is open by pressing F12 and selecting the console tab.

Enter

```
npcs.length
```

The response is 10. Great work.

While a for loop like this is perhaps the most common form for looping there are other options. We can reform the loop using while. First, we define the loop variable commonly called i for iterator. Then we enter while. While has a condition in brackets, we can enter i<10. Then curly braces and the npcs.push line. Then i++. If you forget to include i++, the while loop will loop forever. Because of this while loops can be dangerous. A while loop that loops forever can bring your web page to a grinding halt. Beware.

```
let i = 0;

while(i<10){
    npcs.push(createNPC());
    i++;
}
```

As well as using for with 3 statements. We can use for-in to access the properties of an object. Let's create a variable that accesses npcs[0]. Then we'll add a little console message. Then we enter for and create a new variable prop followed by in npc. Whatever we put in the code block will now repeat with prop equal to the property names in the object npc. We can print this to the console using the log method. We use both the prop name and the value that this is set to. You can access the value of a property in an object using either object variable name dot property name. Or you can use square brackets and a string version of the property name.

```
const npc = npcs[0];
console.log("npcs[0] has the following properties... ");

for(let prop in npc){
   console.log(prop + ":" + npc[prop]);
}
```

Save and let Live Server refresh your browser. The property names and their values are printed out.

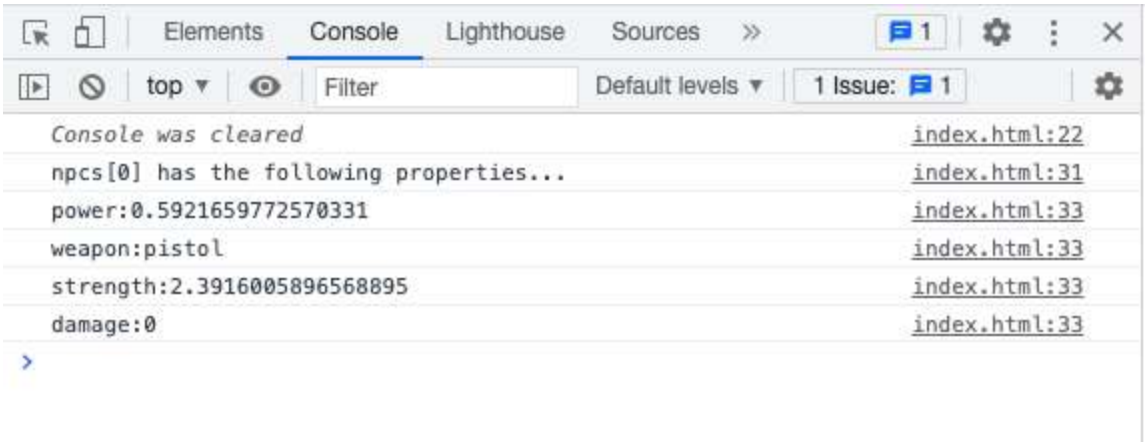

Diagram 9.1 – Using for-in to show property values

Notice how we can access an objects property either by using dot syntax, or the name of the property as a string inside square brackets.

Enter

```
npc.weapon
```

you see the weapon for this npc.

Enter

```
npc['weapon']
```

and you see the same return value.

The last loop option we'll review at this stage is the foreach method of an Array. We'll need an index value. Then we enter npcs.forEach, it is important to use the capital E. JavaScript is case sensitive. In the brackets we enter a function. A new way to define functions in JavaScript is to use arrow functions. Using the forEach method the callback function we create will have a parameter which is each npc in the array in turn. To track this variable enter npc then an arrow that is equals followed by greater than, then curly brackets.

```
npc => { }
```

We could have used

```
function(npc){ }
```

It is the same. But you will inevitably come across arrow functions and it is important that you understand them.

Then we enter a log string we use the index value i, in the string and the value of npc.weapon. This is not npcs[0], this is a new version of the variable npc, that will be npcs[0] for the first iteration, then npcs[1] all the way up to npcs[9]. So the number we print in the string goes up, we add one to the variable i using the ++ increment operator.

```
let i=0;

npcs.forEach( npc =>{
   console.log('NPC no.' + i + ' uses a ' + npc.weapon +
               ' as its chosen weapon.');
   i++;
});
```

Save and over in the browser allow it to refresh. Now you have a list of NPCs showing their weapon of choice.

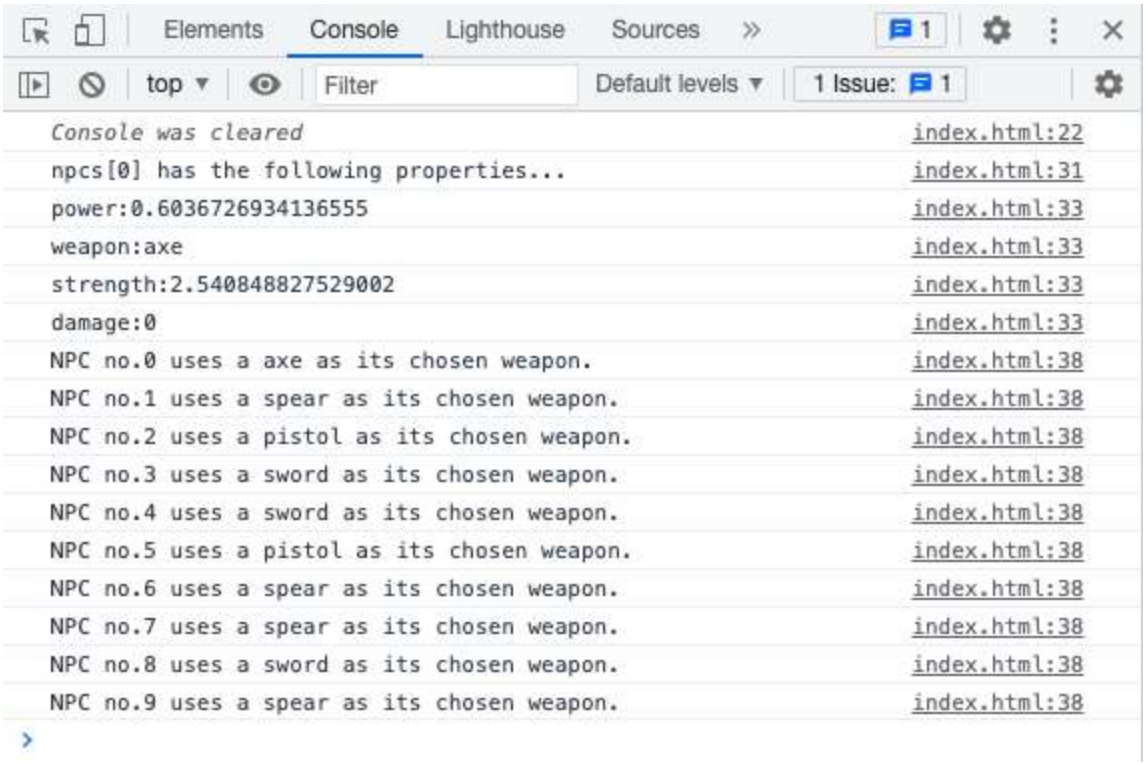

Diagram 9.2 – Using forEach

Loops are very important in programming, and you'll find many times where you'll use the different methods outlined in this lesson. If you get confused, come back to this lesson and review it.

JavaScript has a great technique for extending objects so they can easily include methods as well as properties and we do that using classes, which is the topic of the next lesson.

10. Lesson 8 - Classes

JavaScript classes allow you to combine properties and methods. To code along with this lesson open the file index.html in the folder start/lesson8. To create a new class you use the keyword class, then the name of the class you're creating followed by curly braces.

```
class myClass{
}
```

JavaScript is a big fan of curly braces. If you need to initialize values in a class you need a special function called a constructor. This function is called when you create a new instance of the class. To clear up all this terminology, let's create a simple class to represent a 2d point.

We enter class, all lower case, then the class name, we'll call it Point. Then the curly braces and for a Point we'll need 2 parameters, an x value and a y value. The constructor function is where we initialize the class properties. Inside a class you need to use this to access the class instance. We set this.x to the passed parameter x and this.y to y. That's it. To create a new instance of a point simply enter new Point(, then the values for x and y, followed by the closing bracket.

```
class Point{
   constructor(x,y){
      this.x = x;
      this.y = y;
   }
}

const pt = new Point(20, 65);
```

Save your work and over in your browser, open start/lesson8/index.html using Live Server. If you enter

```
pt
```

Diagram 10.1 – Examining the Point class

You will see that you have an instance of a Point. The great thing about classes is the way we can add methods. Before we see this in action let's create a new class. This time it is a class to represent a Rectangle. We'll have left, top, width and height as parameters. Like we did with the Point class we copy the parameters to class properties using the this keyword.

Now we can create 2 instances of the Rectangle class using new Rectangle and 4 parameters. Parameter 1 is the left position, parameter 2 the top position, parameter 3 is the width and parameter 4 the height.

```
class Rectangle{
        constructor(left, top, width, height){
           this.left = left;
           this.top = top;
           this.width = width;
           this.height = height;
        }
 }

const rect1 = new Rectangle(10, 10, 100, 50);
const rect2 = new Rectangle(50, 20, 200, 80);
```

If you save your work and back in the browser allow Live Server time to refresh then enter

```
rect1
```

You'll see the parameters of the first rectangle.

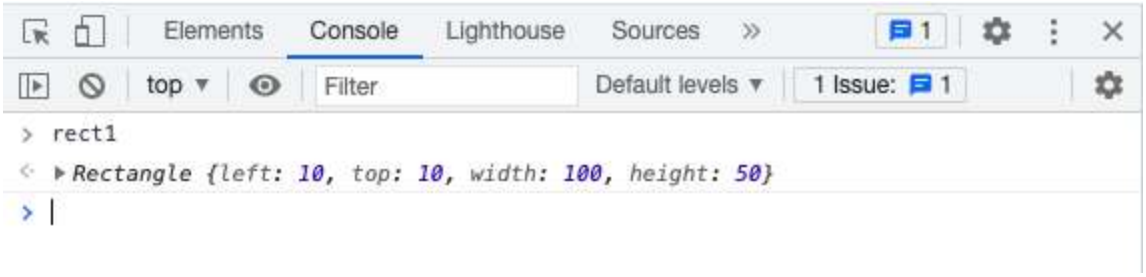

Diagram 10.2 – Using the console to view rect1

So far this isn't very different from an object. Let's start to explore how it differs. We're going to add a special kind of property to the class. If we use the keyword get we can then enter a function with no parameters that returns a value. If we use this syntax it is called a getter property. It is used to add a property that can be calculated from other class properties. We'll create a getter property that returns the area of the rectangle. This is simply width times height. Don't forget to use this for class properties.

```
get area(){
   return this.width * this.height;
}
```

Save, browser refresh and enter

```
rect1.area
```

We get the value 5000, 100 times 50. Notice we just used the property name area, no brackets. If you do add brackets you'll get rect1.area is not a function. Because this special kind of function, a getter property, cannot be called like a standard JavaScript function. It appears to anything calling it as though it is a property of the class.

Now let's create a method. Inside a class a method does not use the function keyword, you just use the name of the method, then any parameters inside brackets and finally the code for the method inside curly braces. Let's create a method that moves the Rectangle. We'll use the parameters x and y. x to indicate the movement to the left property and y the movement to the top property. We simply use the += operator. Remember this.left += x; is the same as this.left = this.left + x. It is simply a short-cut way of scripting this.

```
move(x, y){
   this.left += x;
   this.top += y;
}
```

Save, browser refresh and enter

```
rect1.move(0, 100)
```

Now if you enter

```
rect1
```

The top value is 110, not the original 10 value that the rectangle was when we created it.

Now for a more complex method. We're going to create a method that takes another Rectangle as a parameter and returns true if the passed Rectangle overlaps with the Rectangle calling the method. To do this we'll create four Points. The top left and bottom right of each Rectangle. If left of rect1, l1.x is greater than the right edge of rect2, r2.x, then there cannot be an overlap. If the left edge of rect2 is greater than the right edge of rect1 then there cannot be an overlap. We use the logical or operator || so that if either of these are true we return false, there is no overlap. This completes the horizontal test.

Next, we test vertically. For this we test if the top edge of rect1 is below the bottom edge of rect2 or the top edge of rect2 is below the bottom edge of rect1. If either of these are true, then there cannot be an overlap. For an overlap to occur all of these four conditions must be false. If that is the case, then we can return true.

```
overlap(rect){
          const l1 = new Point(this.left, this.top);
          const r1 = new Point(this.left + this.width,
                               this.top - this.height);
          const l2 = new Point(rect.left, rect.top);
          const r2 = new Point(rect.left + rect.width,
                               rect.top - rect.height);
          // If one rectangle is to the right of the other
         if (l1.x >= r2.x || l2.x >= r1.x){ return false; }
         // If one rectangle is below the other
         if (l1.y <= r2.y || l2.y <= r1.y){ return false; }

         return true;
  }
```

Let's finally enter some messages that use the methods we've created. First, we'll display the area of rect1, using the getter property area. Then we'll use the overlap method to print an overlap message to the console. Then we'll use the move method to move rect1 and now we'll repeat the overlap message.

```
console.log('The area of rect1 is ' + rect1.area);

console.log('rect1 overlaps rect2 ' + rect1.overlap(rect2));

rect1.move(0, 100);

console.log('rect1 overlaps rect2 ' + rect1.overlap(rect2));
```

Initially the rectangles overlap like this.

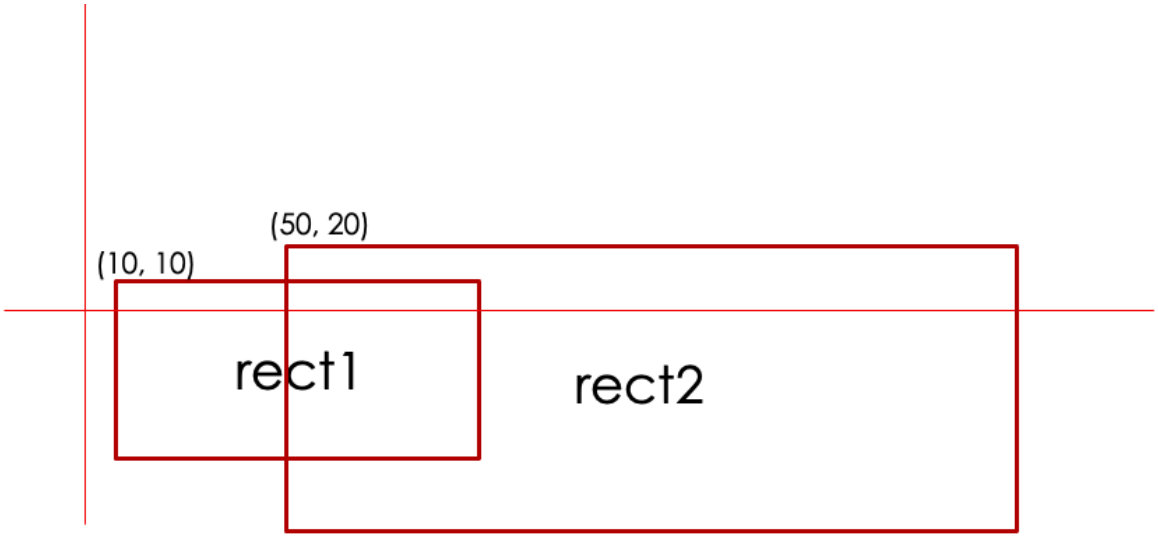

Diagram 10.3 – Before rect1 is moved

After moving rect1 they are like this with no overlap.

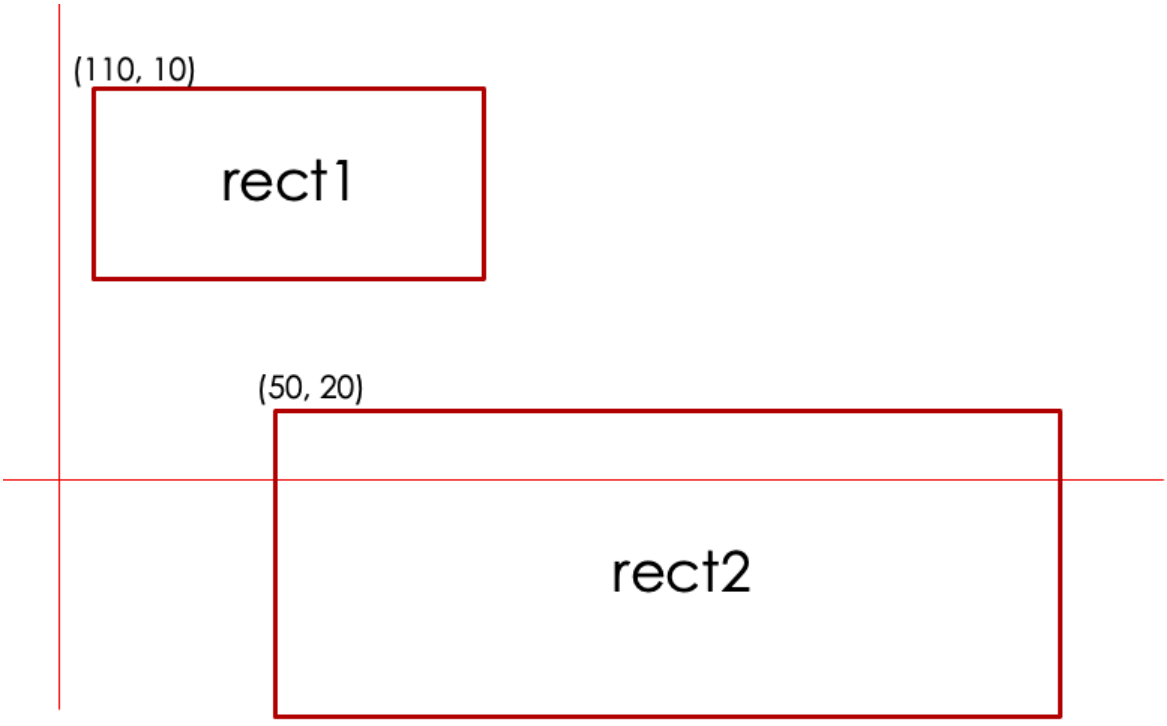

Diagram 10.4 – After moving rect1

Save, refresh the browser and you get 3 messages. The area of rect1 and then two messages about the overlap of rect1 and rect2. The first is true, the second after the move call is false. Great work.

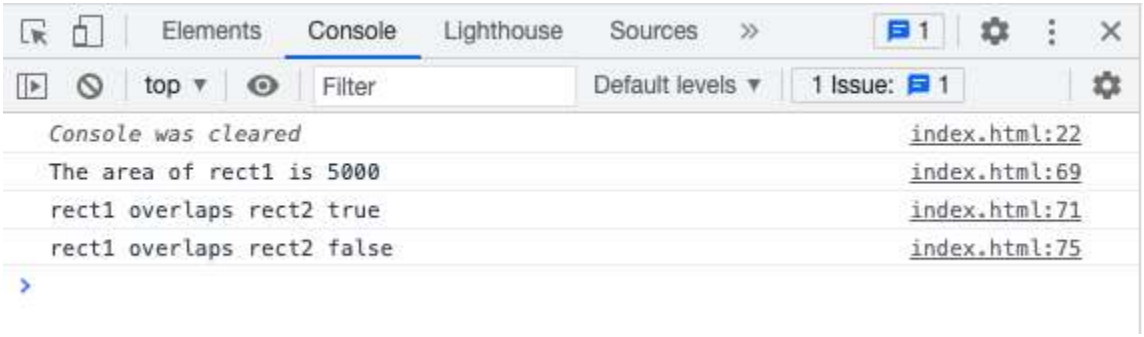

Diagram 8.2 – Output from start/lesson8/index.html

Classes connect properties and methods in a way that makes code clearer.

As your code gets more complex you are likely to use more than one text file to store your code. When this happens a great way to load the code is to use modules and we'll look at this in the next lesson.

11 Lesson 9 - Modules

So far, all our code has been inside a script tag in the index.html file. For small programs like the examples in this course, this approach is fine. But when you create a more complex web application you'll be using lots of code and if you try and keep this in a single file, it will become very hard to maintain the code.

You can add additional JavaScript files like this

```
<script src="pathtofile/myJSFile.js"></script>
```

But the approach we'll use in this lesson is to use modules.

If you open the folder start/lesson9 you see it has three files. index.html as usual and Point.js and Rectangle.js. Let's take a look at the file Point.js. Using the extension .js means this is a file containing JavaScript code. This file just contains the code we entered for a Point class in the previous video. But there is an additional line at the end.

```
export{Point};
```

```
class Point{
   constructor(x,y){
     this.x = x;
     this.y = y;
   }
}

export{Point};
```

When we use modules, we use the keywords export and import. We can choose what to export from the JavaScript file, if the file contains classes and functions, we can select which are exported leaving others private to the file. Whatever we are exporting goes inside curly braces, JavaScript loves curly braces. If there is more than one export then the names are separated by commas.

```
export{ Planet, Moon, Star }
```

Now take a look at the Rectangle.js file. You probably guessed that this contains the code for the Rectangle class. But if you recall the Rectangle class needs to know about the Point class. Notice at the start of the file there is an import line.

```
import {Point} from ‘./Point.js’
```

```
import {Point} from './Point.js';

class Rectangle{ constructor(left, top, width, height){
       this.left = left;
       this.top = top;
       this.width = width;
       this.height = height;
   }

   move(x, y){
       this.left += x;
       this.top += y;
   }

   get area(){
       return this.width * this.height;
    }

   overlap(rect){
       const l1 = new Point(this.left, this.top);
       const r1 = new Point(this.left + this.width,
                            this.top - this.height);
       const l2 = new Point(rect.left, rect.top);
       const r2 = new Point(rect.left + rect.width,
                            rect.top - rect.height);

       // If one rectangle is on left side of other
      if (l1.x >= r2.x || l2.x >= r1.x){ return false; }
       // If one rectangle is below the other
      if (l1.y <= r2.y || l2.y <= r1.y){ return false; }

       return true;
  }
}

export { Rectangle };
```

Again, we use the ubiquitous curly braces, then a name used in the export line from the file chosen as the from path. We use dot forward slash to indicate the current folder, the folder where you'd find the file doing the importing and ../ to move to the folder above the current one. All from paths must start with either ./ or ../.

When we use modules, we need to inform the JavaScript compiler. We add

```
type='module'
```

To the starting script tag.

If you look in the index.html file you'll see it makes use of the Point and Rectangle classes, just like in the previous video. But there is no indication of a module or any attempt to import the necessary files.

If we open the page now with the developer console open you'll see

```
'Uncaught ReferenceError: Point not defined'
```

Can you fix this? You'll need to make sure that JavaScript knows that this is a module and specify import lines for the Point and Rectangle classes. Put the book down now and see if you can write the necessary code.

Nice and simple. First, we use type equals module so JavaScript knows to interpret this as a module. Then we use the keyword import, followed by what we're importing in curly braces then where we find the necessary file. Because the file is in the same folder as the index.html page, the path starts with ./, this is interpreted as the same folder as the file that is importing the external module.

```
<script type='module'>
   import {Point} from './Point.js';
   import {Rectangle} from './Rectangle.js';
```

Now if we save and refresh the browser, we get the log messages that we got in the previous video. Using modules allows you to reuse code you create and makes your code much tidier and easy to maintain.

One thing new programmers find confusing, is scope. You probably have no idea what I mean, let's address that in the next lesson.

12. Lesson 10 - Scope

Understanding how scope affects access to variables and class properties and methods can be very confusing. In this lesson I'll try and make it clear.

To code-along open the file index.html from the folder start/lesson10. There is a variable called name defined, a function called myFunc and a class called game. The name variable is defined at the root of the script, so it has global scope. You can access it anywhere. As you get more experienced with your programming, you'll learn that global variables should, where possible be avoided. But this isn't a lecture about good coding practice, it is about scope.

Make sure you're using Live Server to view lesson10/start/index.html. In the console enter name and the value of name should appear.

Back in VSCode add this console log statement and call to myFunc.

```
console.log(`name=${name}`);
myFunc();
```

Notice I'm using backticks, not single or double quotes, inside the log brackets. Using backticks we create a template string and if we use a dollar sign and the ubiquitous curly braces you can include JavaScript code inside the string value. This will print the value of name. Then we call the function myFunc. Before we run this, let's have a quick look at this function.

```
function myFunc(){
    console.log(`Inside myFunc name=${name}`);

   if (true){
      let course = "The beginners guide to 3D web game
development with ThreeJS";

      const lesson = 10;

      console.log(`course='${course}' inside if statement`);
      console.log(`lesson=${lesson} inside if statement`);
  }
}
```

We have a console statement to print the value of name. What do you think name will be inside the function? We'll see in a minute. Then we have an if statement where the condition it true, so it will always process the statements inside the curly braces. We create a variable and a constant, then we print the values of these to the console.

Time to use Live Server to run the start/lesson10/index.html

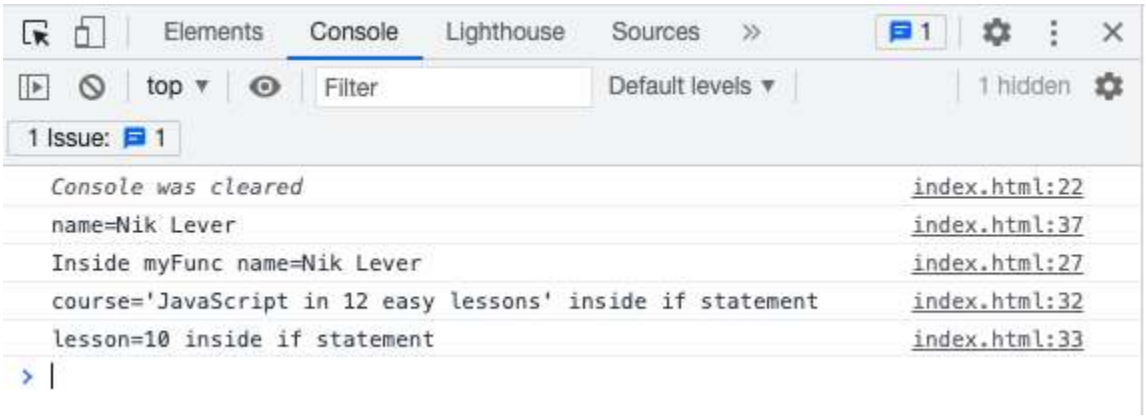

Diagram 12.1 – First scope messages

Outside of the function and inside the function, name has the value 'Nik Lever', since the variable 'name' has global scope. Then course and lesson have the values assigned to them. So far so good.

Now outside the if code block but still inside the myFunc function. Copy the course and section console log lines switching inside to outside.

```
console.log(`course='${course}' outside if statement`);
console.log(`lesson=${lesson} outside if statement`);
```

Let's add 'Inside myFunc' to the console lines, so we can clearly see the console lines called inside this function.

```
const name = "Nik Lever";

function myFunc(){
    console.log(`Inside myFunc name=${name}`);

   if (true){
       let course = "JavaScript in 12 easy lessons";
       const lesson = 10;
       console.log(`Inside myFunc: course='${course}' inside
if statement`);
       console.log(`Inside myFunc: lesson=${lesson} inside if
statement`);
  }

   console.log(`Inside myFunc: course='${course}' outside if
statement`);
   console.log(`Inside myFunc: lesson=${lesson} outside if
statement`);
}
```

Save, let Live Server refresh the browser.

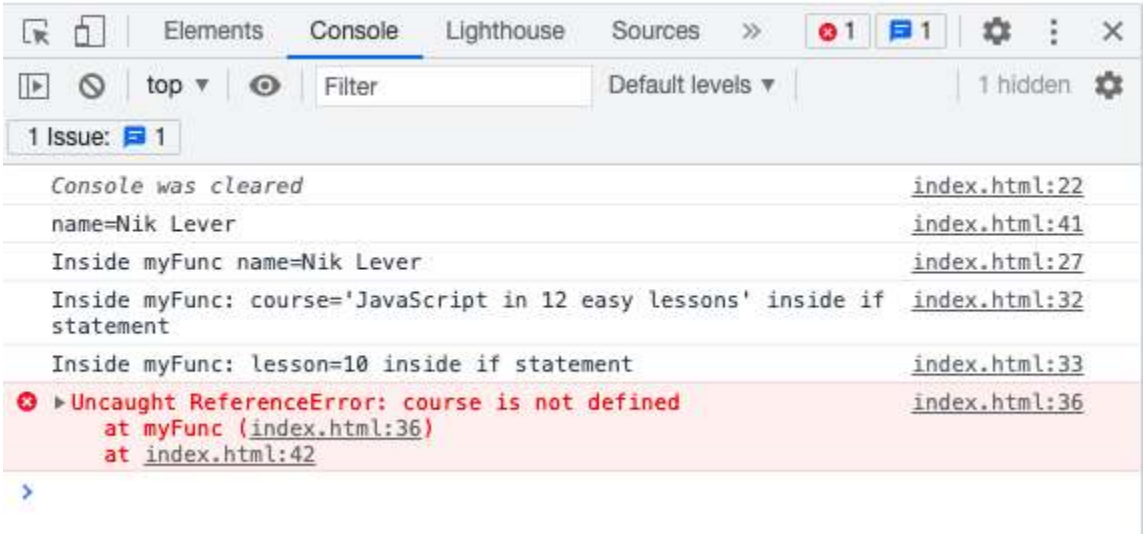

Diagram 12.2 – Uncaught ReferenceError

We get an Uncaught ReferenceError, if you click the link, you'll see it is the console log line outside the if code block, where we attempt to display the value of course. This is because the variable course only exists inside the if code block, not outside the code block. It has local scope to within the curly braces of the if statement.

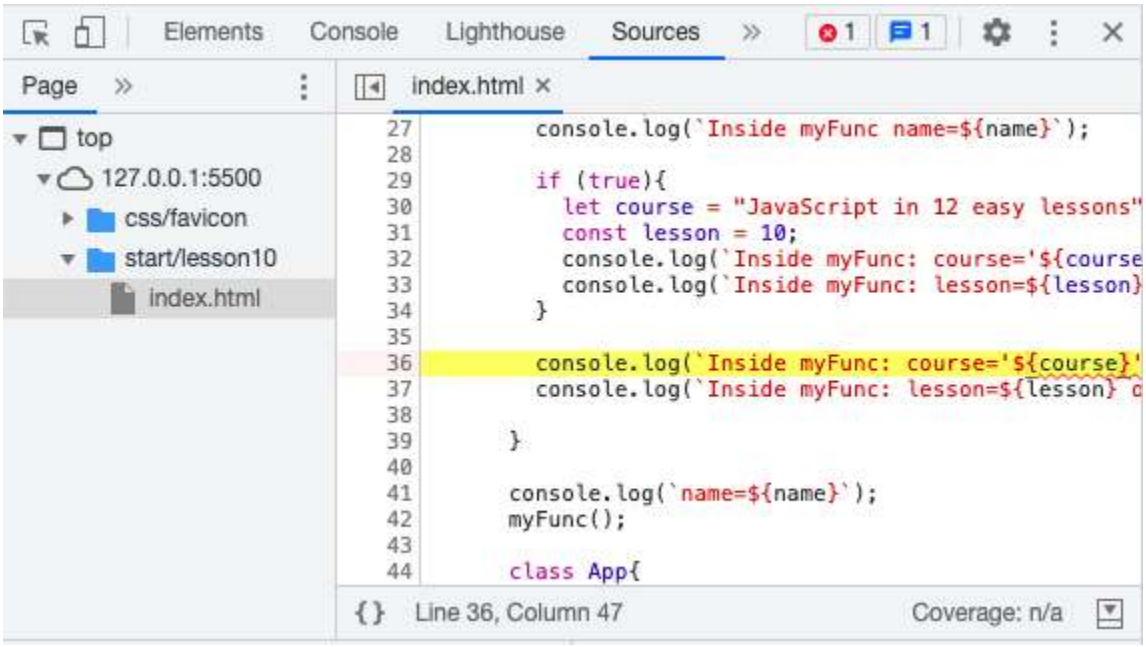

Diagram 12.3 – Displaying where the error occurs in the code

This error will stop any further code running in the function. To allow the code to continue, we can put the problem code inside a try – catch block. We use the keyword try then curly braces wrapping the code, followed by the keyword catch and brackets. Inside the brackets we add an error object, then we add more curly braces and inside these curly braces we add a console message that will display the message property of the err parameter to the catch block.

```
function myFunc(){
   console.log(`Inside myFunc name=${name}`);
   if (true){
      let course = "JavaScript in 12 easy lessons";
      const lesson = 10;
      console.log(`Inside myFunc: course='${course}' inside
if statement`);
      console.log(`Inside myFunc: lesson=${lesson} inside if
statement`);
  }

  try{
     console.log(`Inside myFunc: course='${course}' outside
if statement`);
    console.log(`Inside myFunc: lesson=${lesson} outside if
statement`);
  }catch(err){
    console.error(`myFunc:${err.message}`);
  }
}
```

Save and let Live Server refresh the browser. You can expand the error to see how we got to this line. Adding a try-catch block is a useful technique for difficult to fix code bugs.

Diagram 12.4 – Expanding an error console output

Now let's turn our attention to the App class.

In the mouseMove method add this console log statement

```
console.log(this.name);
```

And in the App constructor add an event listener. This neat bit of code, means whenever there is a mouse movement we can call a chosen function. Notice the event name is 'mousemove', that is the event we are going to be listening out for. The second parameter is the function to call when this happens.

```
document.addEventListener('mousemove', this.mouseMove);
```

Now immediately after the class closing curly brace, let's create an instance of the game class. Notice we need to pass a name as a parameter.

```
const myGame = new App("Scope example");
```

Save and refresh the browser. Lots of errors. Switch to the Sources tab and click on the number of the first line in the mouseMove method. Now when you move the mouse the processor will stop at this line, you've added a breakpoint.

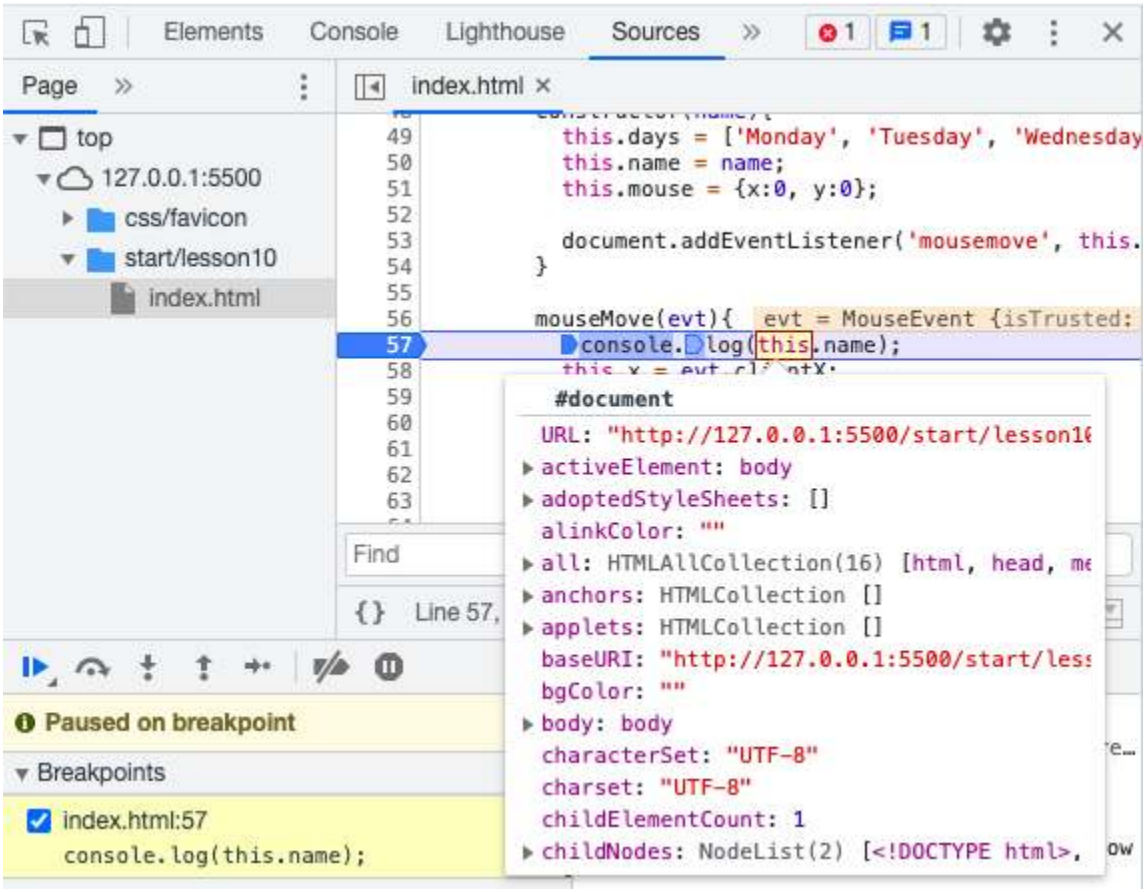

Diagram 12.5 – Inside mouseMove this is the HTML document

Hover over this and slide down to see that this is not the game at all, it is the HTML document. This is easily fixed.

Jump back to VSCode and back where we created the event listener, add

```
.bind(this)
```

```
document.addEventListener('mousemove',
this.mouseMove.bind(this));
```

Now inside the function, instead of this being the document, this will be the App as expected. Save and let Live Server refresh to confirm. We see the App name, 'Scope example', as expected. If we add a breakpoint as before, we can clearly see by hovering over the word this in the mouseMove method that this is now the App instance. Nice.

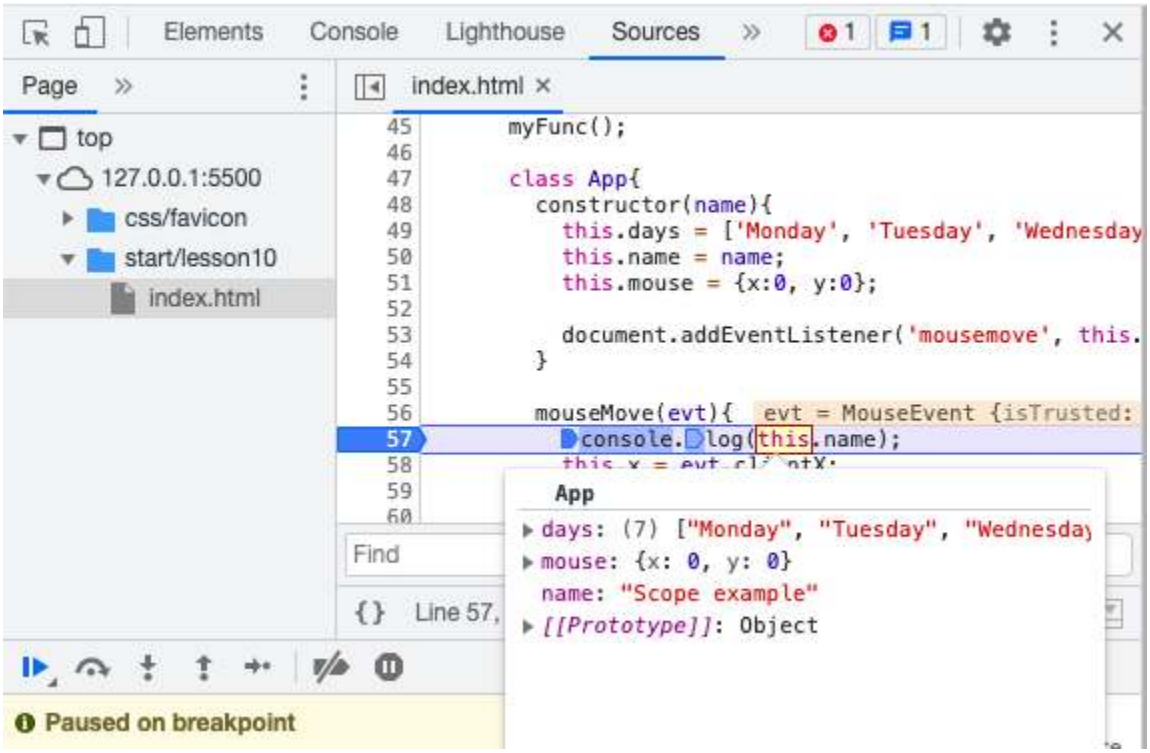

Diagram 12.6 – this is now the App instance

Notice the showDays method. In the constructor for the App we created an array of days.

```
this.days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday'];
```

In the showDays method we use the forEach method of an Array to attempt to display each day.

```
showDays(){
   this.days.forEach( function(day){
      console.log(`global name: '${name}' app
name:'${this.name}' day:'${day}'`);
   });
}
```

To emphasize the role of scope we also show the global value of name and the local value of name using a template string. Enter this code to call this method.

```
myApp.showDays();
```

Then save and let Live Server refresh your browser.

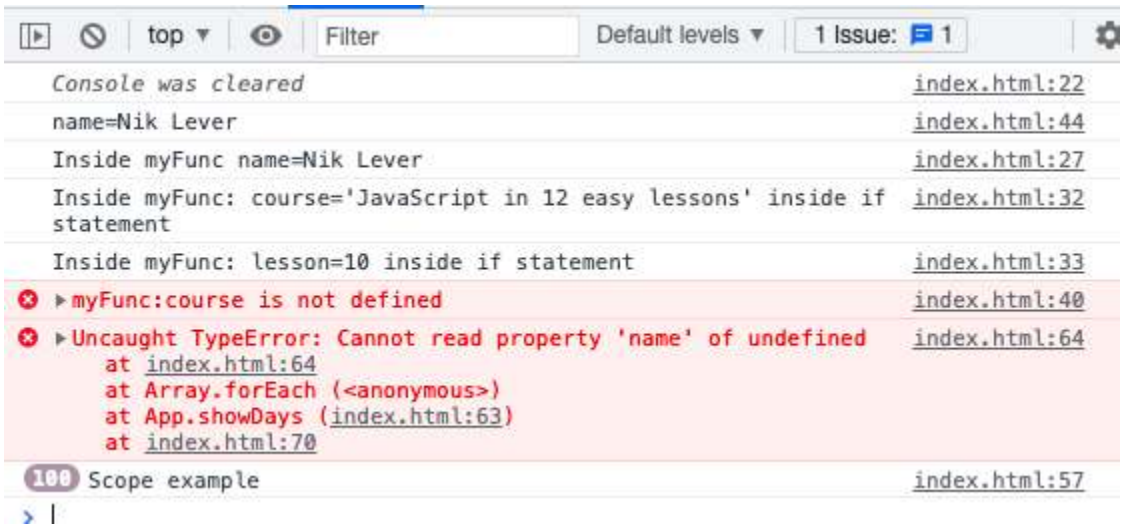

Diagram 12.7 – Uncaught TypeError

You'll get an Uncaught TypeError,: Cannot read property 'name' of undefined. That's because the function has its own scope and 'this' inside the function is not the App. There are two ways to fix this. The easiest is to switch to arrow functions. Arrow functions have the scope of where they are defined. So 'this' will be the App.

```
this.days.forEach( day =>{
```

If we save and refresh, we now get all the days in the Array displayed as expected. Notice also that global name and local name are as expected.

The optional way is to create a variable, often called 'self' and set 'self' equal to 'this'. Then inside the function we replace 'this' with 'self'.

```
const self = this;

this.days.forEach( function(day){
   console.log(`global name: '${name}' game
name:'${self.name}' day:'${day}'`);
});
```

Again, if we save and refresh, now self.name points at the game name and so no errors are generated.

Scope is challenging for new programmers but with practice you'll soon understand where the problems are.

Maths allows us to achieve some stunning visual effects. We've met the JavaScript Math object already in this course, in the next lesson we'll look at some super useful methods of this object that you can use in your web apps. Don't skip the chapter because you hate Maths!

13. Lesson 11 - Math

Often you'll be using a little Maths in your code. To code along with this video open index.html from the folder start/lesson11, and also open Vec.js from the same folder.

First notice that the Vec class has copy, clone, add, sub, distanceTo, angle and angleBetween methods and we'll review these as they come up in the code we enter.

First, we'll create 3 instances of the Vec class.

```
const a = new Vec();
const b = new Vec(4, 0);
const c = new Vec(4, 3);
```

The name Vec is short for vector. In Maths a vector is simply a value with more than one component. We'll see a lot of vectors over the course. When a value has a single component, it is called a scalar. The Vec class has two components, x and y. So, to create one we use two parameters. But if we don't provide any parameters the constructor code for a Vec instance has the default value 0 for both x and y. By adding equals and a value we can easily add default values for parameters to any function.

```
class Vec{
   constructor(x=0,y=0){
      this.x = x;
      this.y = y;
   }
}
```

Here we have the Vec instances 0,0, 4,0 and 4,3.

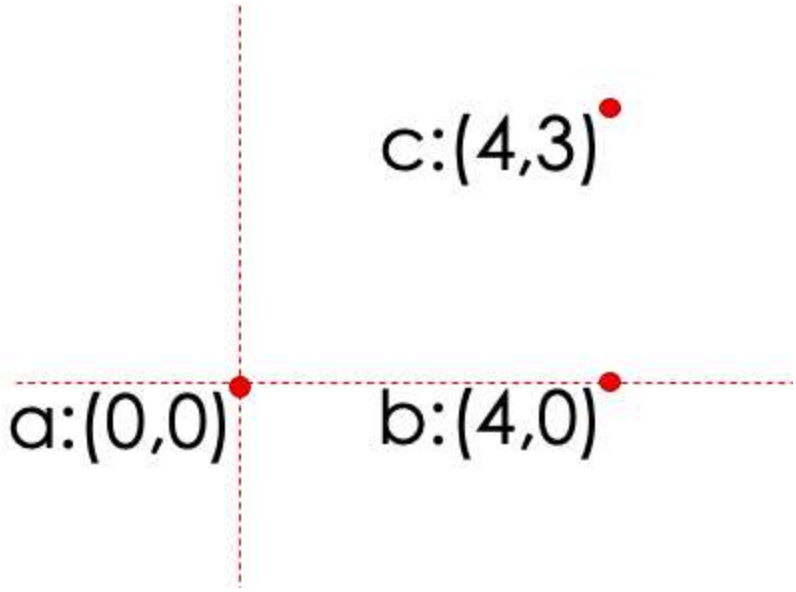

Diagram 13.2 – The Vec instances plotted

If we show these as a triangle they may be more familiar.

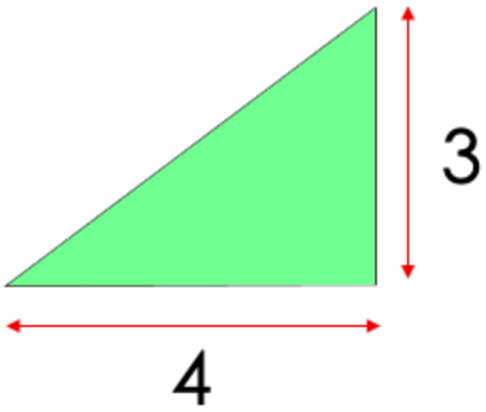

Diagram 13.2 – Vec instances forming a triangle

If you recall your Pythagoras, a popular example is a triangle, with one edge of 4 units and another of 3 units. If we square the 4 we get 16 and square the 3 we get 9, add them together we get 25. Find the square root of 25 and we get the length of the hypotenuse, the longest side, the one away from the right angle.

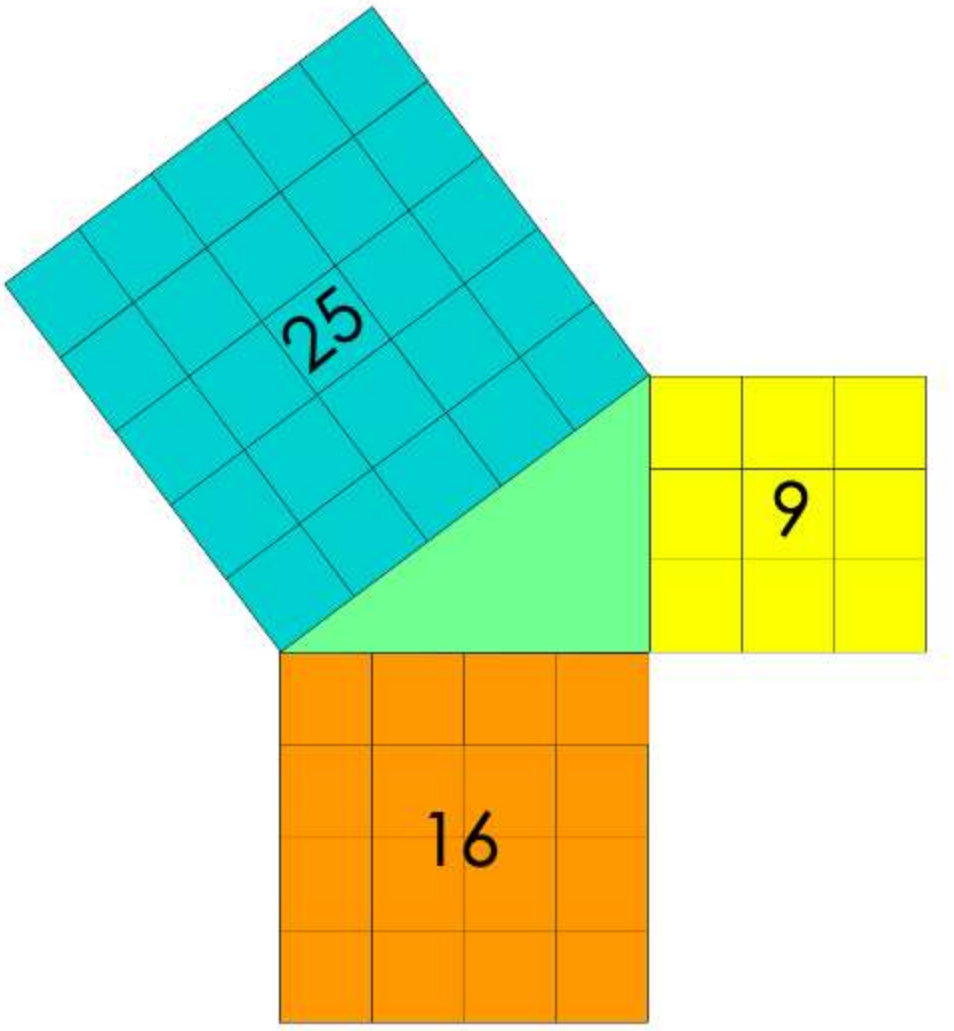

Diagram 13.3 – Using Pythagoras to find the length of the longest edge

If you recall the Vec class has a distanceTo method. The distance from a, 0,0 to c at 4,3 should be 5 units. Let's add a console message, we'll use a template string and inside the dollar curly braces call the distanceTo method. Now check if 5 is the length we get.

```
const a = new Vec();
const b = new Vec(4, 0);
const c = new Vec(4, 3);

console.log(`The distance from a to c is
${a.distanceTo(c).toFixed(2)}`);
```

Save and refresh and we get exactly that. I added toFixed(2), this limits the number of values after the decimal point to the value we enter as a parameter. So, we get 5.00.

Diagram 13.4 – Output from Vec.distanceTo method

Let's take a look at the Vec class method distanceTo.

```
distanceTo(pt){
   const delta = this.clone().sub(pt);
   return Math.sqrt( delta.x*delta.x + delta.y*delta.y );
}
```

First, we define a value delta this is a clone of the Vec instance calling the method, a copy in other words. Here we have the Vec instance a, which has the value 0,0. Then we use the method sub, which subtracts the passed Vec. At this stage delta has the values, -4, -3. We multiply -4 by -4, getting 16, then add -3 times -3 getting 16 + 9. I hope you remember that positive times positive is positive, positive times negative is negative but negative times negative is also positive. The value we pass to the Math method sqrt, short for square root, is 25. The square root of 25 is 5. You'll be amazed how often you'll use Pythagoras' theorem to find the distance between two objects if you do any game programming. The theorem also works in 3 dimensions just as well as in 2 dimensions.

Another common problem is finding the angle between two vectors. If instead of thinking of b and c as points in 2d space, we think of them as lines extending from the origin, 0,0. Then we can find the angle between them using the Math method atan2. Again, we'll reduce the precision to just two numbers after the decimal point. Add this code

```
console.log(`The angle between b and c
is ${b.angleBetween(c).toFixed(2)} degrees`);
```

Save and refresh and we get the angle between vector b and vector c.

Diagram 13.5 – Displaying the angleBetween output

The Vec.angleBetween code creates two angles using the angle method of the Vec class.

```
angle() {
   const theta = Math.atan2( -this.y, -this.x ) + Math.PI;
   return theta;
}

angleBetween(pt){
   const theta1 = this.angle();
   const theta2 = pt.angle();
   return Math.abs(theta1 - theta2) * 180 / Math.PI;
}
```

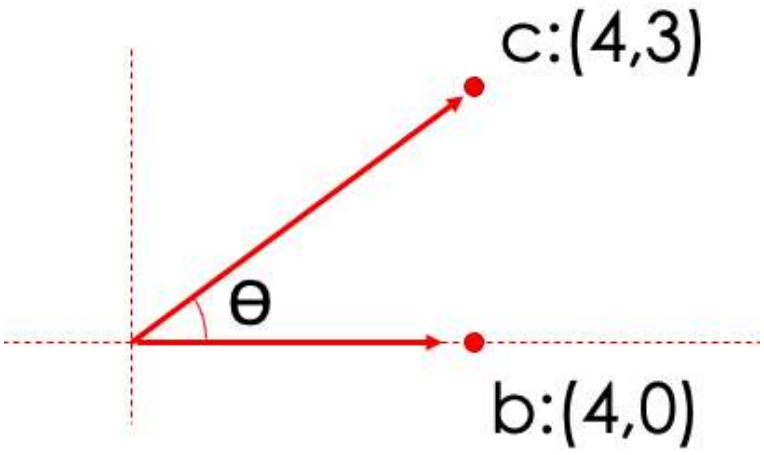

Diagram 13.6 – Vectors b and c

The angle method returns the angle between the vector and the positive x axis. For b which is inline with the x axis the value is zero. Whereas c has the angle 0.6435. You might have been expecting a value around 30 degrees. Vec.angle uses the Math method atan2 to get the angle. All Math trig functions and atan2 is one, return a value in radians. There are 2π radians in a full revolution. This is a very important fact, so I'll say it again. There are 2π radians in a full revolution.

The angle method uses the atan2 method, this takes the y value and the x value as parameters. The atan2 method returns a value between -π and π, to ensure only positive values we negate the parameters and add π to the returned value.

Now we have 2 angles, theta1 and theta2, we get the angle between them by subtracting one from the other. We use another Math method abs, this returns the absolute value, a negative value becomes positive. Then we multiply by 180 and divide by π to convert the radian value to degrees.

So far, you've met, sqrt, atan2, abs methods and the constant π from the Math object and have learnt how to find the distance between two points in 2d space and the angle between two 2d vectors. Two other useful methods are the sin and cos methods. Notice on this page we have a star displayed. We're going to write some code to rotate this star around in an orbit.

Enter

```
const center = new Vec(150, 150);

const radius = 100;
let theta = 0;

const star = document.getElementById('star');
```

To start with we need a couple of constants and a variable. center will be the point in the web page that is the centre of the orbit. Radius is the distance that the star will be from this centre point. theta is the Greek word commonly used to represent an angle, we'll initialize it to zero. Now we need to get the star element. It has been given the id star so we can use the document method, getElementById to access it.

Now we need to define a function that will update the value of theta then calculate where to place the star and set its style values to position it. We'll call it rotateStar. We want this to be called repeatedly, a great way to do that is to use the JavaScript method requestAnimationFrame.

```
function rotateStar(){
   requestAnimationFrame(rotateStar);
   theta += 0.05;
}
```

This is a built-in function and will be called around 60 times a second if the browser can keep up. It takes a function that it will call. Here we call rotateStar again. So, this function keeps calling itself. Now we have a repeating function we need to update theta, by adding 0.05, it will take 100 calls to change theta by 5. A full revolution is 2π or about 6.3. 6.3/0.05 is about 125. At 60fps the star will take about 2 seconds, 60 * 2 being 120, to complete an orbit.

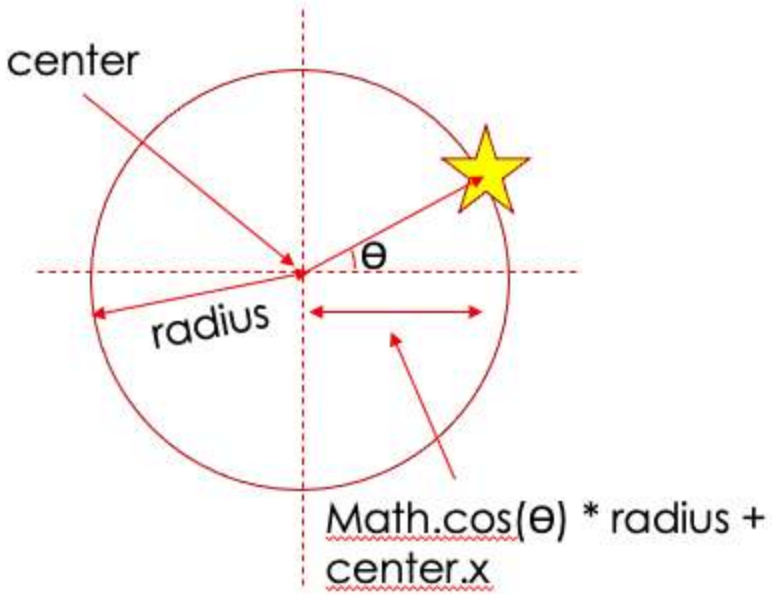

Diagram 13.7 – Setting the x position of the star

The x position of the star can be set using the Math method cos, which returns the cosine of the passed angle. The cosine will return a value between -1 and 1. We want a value between -radius and radius so we multiply the returned value by radius then we add the centre x value. That gives us the x position. When theta = 0, cos = 1, theta = π/2, cos = 0, theta = π, cos = -1, theta = 3π/2, cos = 0. So the x value slides left until theta = π, then slides right until theta = 2π. This then repeats.

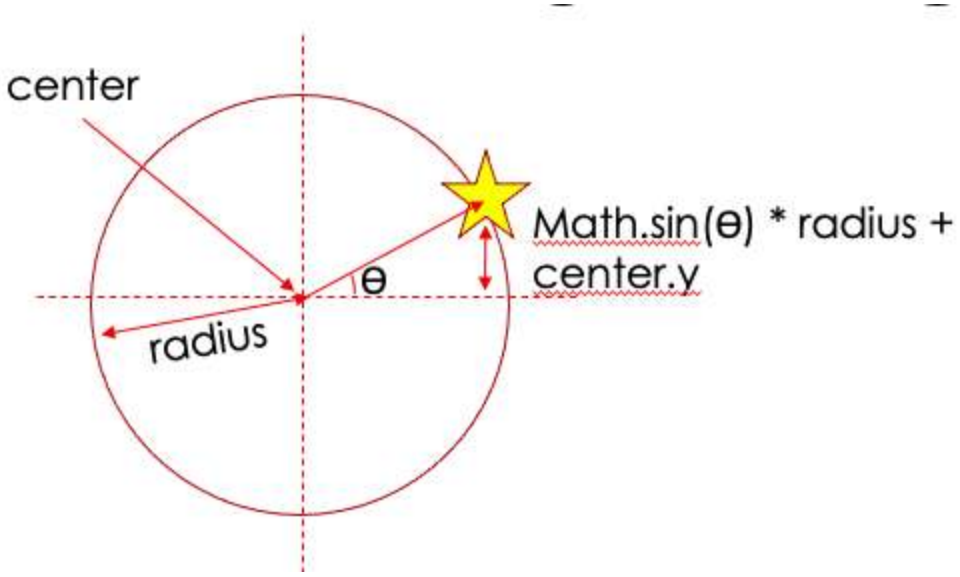

Diagram 13.8 – Setting the y position of the star

The y position is more or less identical, but we use the sin method not cos. When theta = 0, sin = 0, theta = π/2, sin = 1, theta = π, sin = 0 and theta = 3π/2, sin = -1. Here the y value increase to π/2, then decreases to 3π/2, then increases again to 2π.

We can use the calculated values x and y to set the left and top values of the style property of the star element. When setting CSS style values we need to supply a unit, here we use 'px', short for pixel.

Add a call to rotateStar to star the looping action.

```
const centre = new Vec(150, 150);
const radius = 100;

let theta = 0;
const star = document.getElementById('star');

rotateStar();

function rotateStar(){
   requestAnimationFrame(rotateStar);
   theta += 0.05;
   const x = Math.cos(theta) * radius + centre.x;
   const y = Math.sin(theta) * radius + centre.y;

   star.style.left = `${x}px`;
   star.style.top = `${y}px`;
}
```

Save and refresh and you have a rotating star.

The key uses of Maths in your apps will be to find distances, angles and to use sin and cos for smooth movements. Great work.

There just one more thing I'd like to tell you about before we finish and that is about JSON, a great way in which you can initialize your applications data. Find out all about it in the next lesson.

14. Lesson 12 - JSON

JSON stands for JavaScript Object Notation. It's great for loading data into a web app. A JSON file is just a simple text file. Take a look at characters.json from the folder start/lesson12.

```
[
  "King Arthur",
  "Merlin",
  "Lancelot",
  "Guinevere"
]
```

Because the first character is a square bracket, the JSON parser knows to interpret the remaining text as an Array. If it started with a curly brace then it would be read as an object, we'll see how that works in a minute.

The aim is to load this file into your app so you can use it as a JavaScript Array. The easiest way is to use the built-in method fetch. This returns something called a Promise. The key feature of a Promise is it works asynchronously. While it is busy doing its work, the main program can continue. It doesn't block progress. When you work with a Promise you add a then method, this will be called when the Promise resolves, either successfully or unsuccessfully. In the case of a fetch, we get a response, the file that has been fetched. We can call the json method on this response, which is itself another promise. If this successfully parses the file into a JavaScript Array or object then we get this value as a parameter and can create a callback function using the forEach method of an Array to display each character in the console.

```
fetch('./characters.json')
   .then(response => response.json())
   .then(obj =>{
      obj.forEach( character => console.log( character ));
   });
```

Save, load start/lesson12/index.html using Live Server and you'll see key characters from the King Arthur legend.

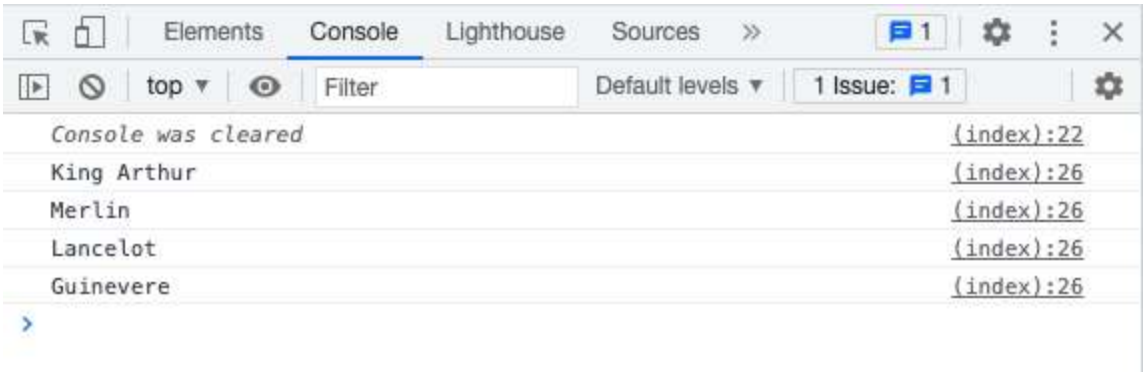

Diagram 14.1 – Outputting the character.json file

The JSON syntax is not at all forgiving. To see what I mean, remove the last double quote from Guinivere. Now save and when your browser refreshes you'll get an 'Uncaught in promise error. SyntaxError unexpected token in JSON'.

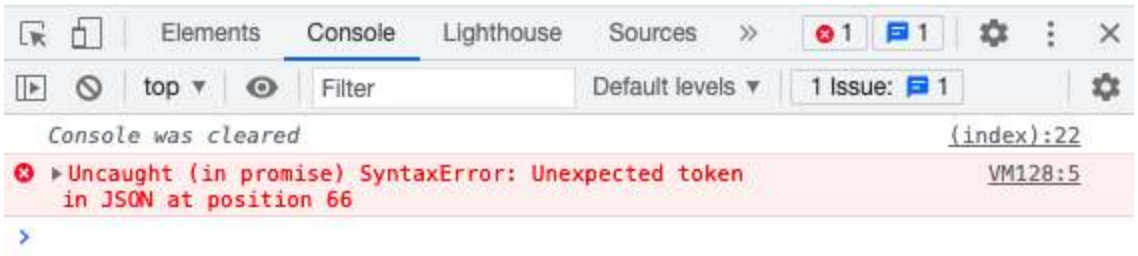

Diagram 14.2 – Problem parsing a JSON file

When using Promises it is usually best to add a catch block to the end of the then callbacks, like this

```
obj.forEach(
   character => console.log( character ));
}).catch( err => console.error(err) );
```

If you save and refresh now it dies much more gracefully.

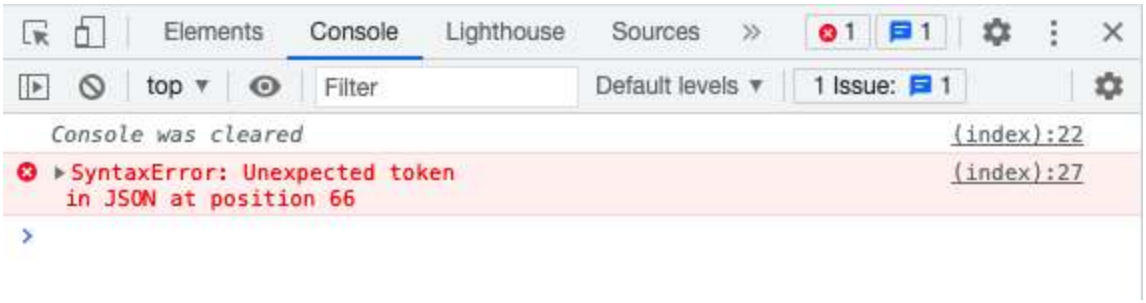

Diagram 14.4 – Using a catch handler in the fetch code

If your JSON file is complicated, then tracing the error can be tricky. VSCode can help

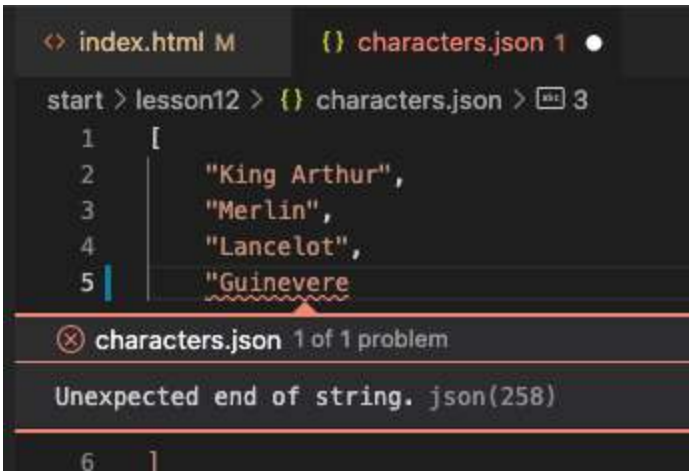

Diagram 14.5 – Using VSCode to track down JSON parsing errors

A useful online tool is jsonLint.com. Copy the text from the file, paste it in and press Validate JSON. If you get an Error, the problem is usually on the next line after the highlighted red one. Adding a double quote back, pressing Validate JSON and we're good to go.

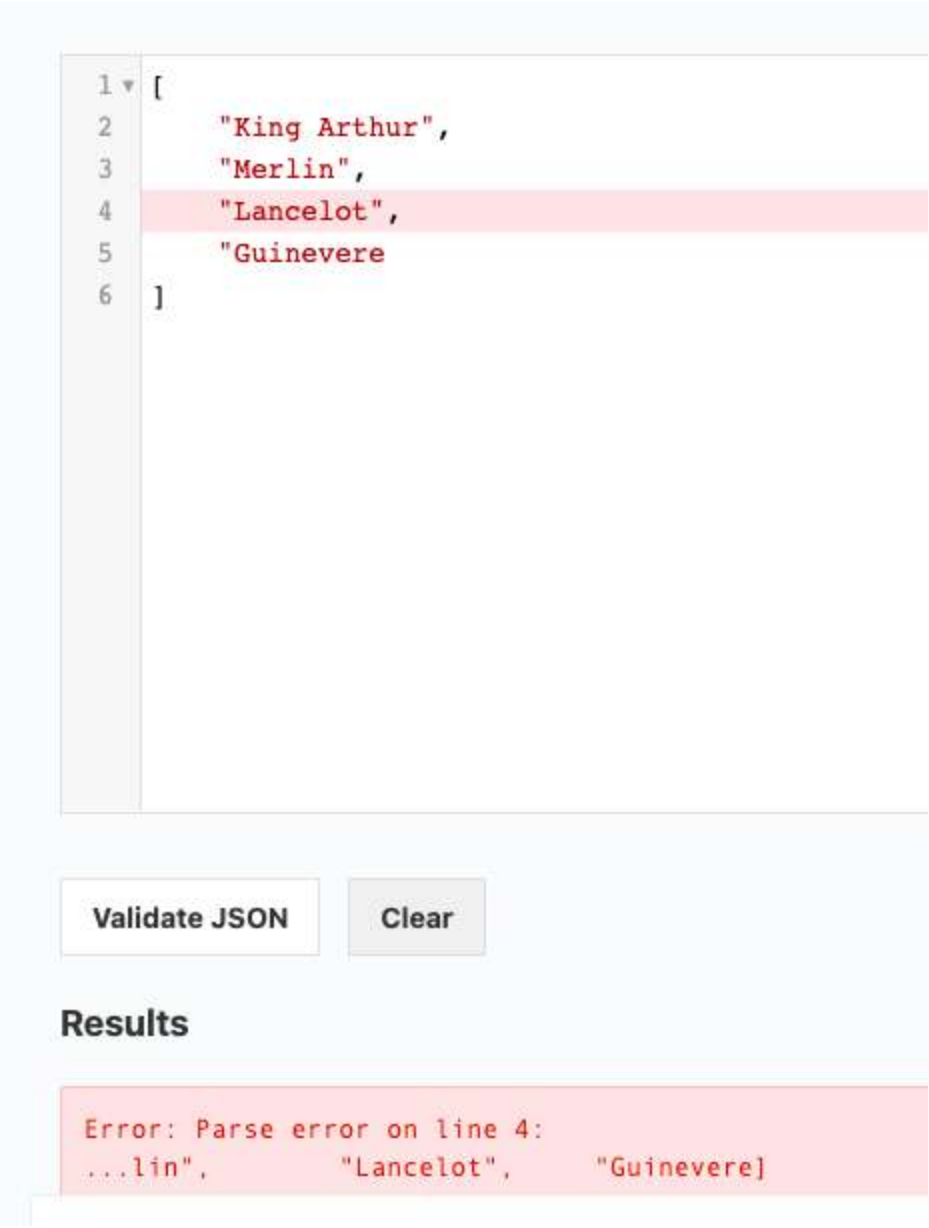

Diagram 14.6 – Using jsonlint.com

Don't forget to fix the characters.json file back in VSCode.

I mentioned if the file starts with a curly brace then the remainder is interpreted as an object. Take a look at planetinfo.json.

```
{
  "description": "Planets in the solar system",
  "planets": [
   { "name": "Mercury",
     "aphelion": 69.8,
     "perihelion": 46.0,
     "radius": 2439,
     "moons": [0]
   },
...
```

This is more complex, to start with it is an object. As such it contains name/value pairs. Notice the name description and then a colon and then the value of this property. Take a good look at how we loaded the characters.json file and see if you can load the planetinfo.json file and then display the description line in the console. Put the book down now and give it a try.

We can just copy the first fetch code block. Replace characters with planets. Then replace the forEach line with a simple console log statement. We log obj.description. Save and refresh to see the line printed to the console.

```
fetch('./planetinfo.json')
   .then(response => response.json())
   .then(obj =>{
      console.log(obj.description);
   }).catch( err => console.error(err) );
```

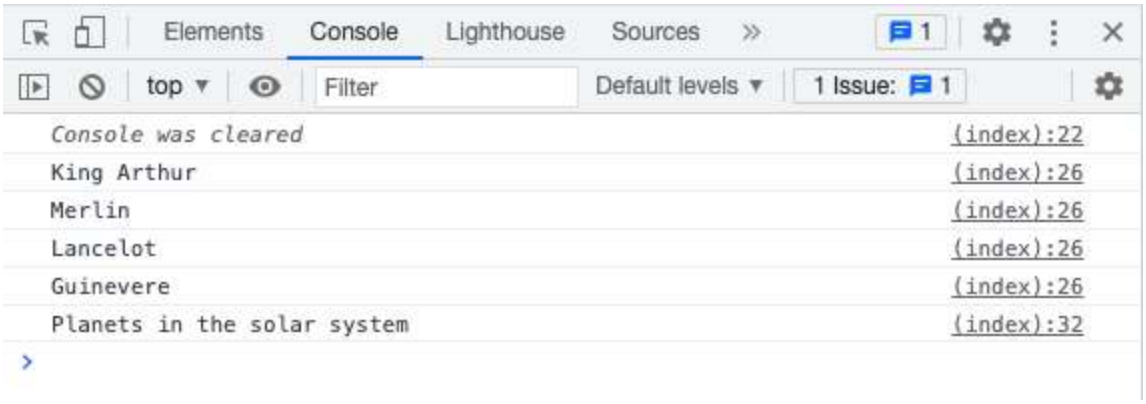

Diagram 14.7 – Showing the description line

Now we have a bunch of facts about the planets, let's see if we can display the largest planet. We'll create a function to do this, that takes the planets array, which is the other property in the planets object. We'll replace the console log line in the then callback with a call to this new function passing obj.planets, the planets property of the JSON object we've just loaded. If we look back at the planetinfo.json file, you'll see this is an Array. Notice after the colon, square brackets. So the planetinfo.json file is parsed as an object with two properties. Remember object properties and name value pairs. The first name value pair is description which is a string value and the second name value pair is planets and this is an Array.

Remember square brackets – arrays, curly braces – an object defined as a series of name value pairs. The name and the value are separated by a colon and each name value pair is separated from the next using a comma.

To find the largest planet, we have an Array of planets. Each item in the Array is an object. One property of this object is radius. We just need to find which object in the Array has the largest radius property. We'll use the reduce method of an Array. This reduces an Array to a single value. It does this using a callback with two parameters. Parameter one is the accumulator value, this is initialized to the first item in the array. Then the second parameter will be the second item in the Array for the first call back. We check if acc.radius is greater than loc.radius, if it is then we return acc, if not we return loc. This callback will continue, with acc set as largest value so far and loc set as the third, then forth up to the last item in the array. By returning the largest of the saved value and each item in the array we find the largest planet in the array. This table should help your understanding.

acc	loc	acc.radius	loc.radius	stored
Mercury	Venus	2439	6052	Venus
Venus	Earth	6052	6371	Earth
Earth	Mars	6371	3389	Earth
Earth	Jupiter	6371	69911	Jupiter
Jupiter	Saturn	69911	58232	Jupiter
Jupiter	Uranus	69911	25362	Jupiter
Jupiter	Neptune	69911	24622	Jupiter

Table 14.1 – How the reduce function returns the largest planet

The first time we enter the callback method. acc is Mercury and loc is venus. Looking at the radius values we can see that for this pass loc radius is larger so we store venus. As we enter the second pass acc is now Venus and loc is earth. Earth is larger so we store Earth as the acc, the accumulator, value. Now we test Mars, this is smaller than Earth so we keep Earth as the acc value. We move on to Jupiter, which is much bigger than Earth so we now store Jupiter as the acc value. As we iterate through Saturn, Uranus and Neptune, Jupiter is the largest, so Jupiter remains as the acc value. We exit the method with largest set to Jupiter.

```
function displayFacts(planets){
   const largest = planets.reduce( (acc, loc) => acc.radius >
loc.radius ? acc : loc );
   console.log(`The largest planet is ${largest.name}`);
}
```

Save and refresh you'll see that Jupiter is the largest planet. Nice.

Can you use a similar technique to display the smallest planet? *You'll need to adapt the reduce method to save the planet with the smallest radius.* Put the book down and give it a try.

How did that go? Just a case of copying the existing code and changing greater than for less than. And changing largest to smallest. Save and refresh and the answer is clearly Mercury. Nice work.

```
const smallest = planets.reduce( (acc, loc) => acc.radius <
loc.radius ? acc : loc );
console.log(`The smallest planet is ${smallest.name}`);
```

That completes your introduction to JavaScript. You're ready to start creating your first web apps. Before you go a quick quiz to see what you've learnt in the course.

15. What you've learnt

1. If x=3 and y="4" what does x + y equal?
   a) "34"
   b) 7
   c) NaN

2. If str="This is a string". What is str.indexOf('string')?
   a) 11
   b) 10
   c) 7

3. Which answer would return the sum of two numbers?
   a) function add(a, b){ return a + b; }
   b) function add(a, b){ a + b; }
   c) function sum(a + b){ return result; }

4. If a=1 and b=2. With this code snippet what gets printed to the console?

```
if (a>b){
  console.log('this');
}else{
  console.log('that');
}
```

a) that
b) this

5. Which answer adds the string 'Pluto', to the array planets?
   a) push(planets, 'Pluto')
   b) planets.push('Pluto')
   c) planets.add('Pluto')

6. Which answer returns the strength property of the third NPC, from the npcs array?
   a) npcs[3]['strength']
   b) npcs.get(2).strength
   c) npcs[0].strength

7. Which answer gives the correct syntax of a for loop?
   a) for(let i=0; i<10; i++){}
   b) for(i=0, i<10, i++){}
   c) for i in range(0,10):

8. Which is the correct syntax for defining a class?
   a) class MyClass(){}
   b) MyClass typeof class
   c) class MyClass{}

9. Which answer would add the Point class from the file Point.js in the current folder?
   a) import (Point) from ‘./Point.js’
   b) import { Point } from ‘Point’
   c) import { Point } from ‘./Point.js’

10. Given the code snippet below, what gets printed in the console

```
for(let i=0; i<10; i++){
      const n = i;
}
console.log(`n=${n}`);
```

   a) n=0
   b) n=10
   c) n is not defined

11. If a is at (2,4) and b is at (6,7). Which answer gives the distance from a to b?
   a) Math.sqrt((6-2)*(6-2) + (7-4)*(7-4))
   b) Math.sqrt((6-2) + (7-4))
   c) Math.length((6-2), (7-4))

12. Is this valid JSON?

```
{
  array: [ 1, 2, “buckle my shoe” ],
  “three”: “four”,
  “knock”: [ ‘@’, ‘the’, ‘door’ ]
}
```

   a) Yes
   b) No

Answers

1-a, 2-b, 3-a, 4-a, 5-b, 6-c, 7-a, 8-c, 9-c, 10-c, 11-a, 12-b

16. About the author

After getting a degree in Graphic Design, I started work in 1980 as a cartoon animator. Buying a Sinclair ZX81, back in 1982, was the start of a migration to a full time programmer. The ZX81 was quickly swapped for the Sinclair Spectrum, a Z80 processor and a massive 48K of ram made this a much better computer to develop games. I developed a few games using Sinclair Basic and then Assembler. The Spectrum was swapped for a Commodore Amiga and I developed more games in the shareware market, moving to using C. At this stage it was essentially a hobby. Paid work was still animated commercials. I finally bought a PC in the early nineties and completed an Open University degree in Maths and Computing. I created a sprite library ActiveX control and authored my first book, aimed at getting designers into programming. In the mid nineties along came Flash and the company I was now running, Catalyst Pictures, became known for creating games. Since then the majority of my working life has been creating games, first in Flash and Director, as Director published the first widely available 3D library that would run in a browser using a plugin. In recent years game development has involved using HTML5 and Canvas. Using both custom code and various libraries. A particular preference is to use the latest version of Adobe Flash, now called Animate that exports to the Javascript library Createjs.

I've worked for the BBC. Johnson and Johnson. Deloitte, Mars Corporation and many other blue chip clients. The company I've run for over 30 years has won a number of awards and been nominated for a BAFTA twice, the UK equivalent to the Oscar. Over the last 20 years I have been struck by just how difficult it has been to get good developers and have decided to do something about this rather than just complain. I run a CodeClub for kids 9-13 years old and I'm developing a number of courses for Udemy hoping to inspire and educate new developers. Most of my courses involve real-time 3d either using the popular Open Source library Three.JS or Unity. I'm currently having a lot of fun developing WebXR games and playing with my Oculus Quest.

JavaScript in 12 easy lessons

JavaScript, often abbreviated to JS, is the default language of the internet. But it doesn't stop there, JavaScript powers your smart tv, works with the internet of things, and helps build cross platform apps.

Because JS is the default language of your browser, you can start to write JS code without installing any programs, you can just use your browser. Making JavaScript the easiest programming language for new coders.

You'll be pleased to know that JavaScript is easy to learn. If you do one lesson a day then you will have a good grounding in the basics of coding.

Having learnt the language, you can then more readily lend your skills to another coding language such as Python, Java or C++.

If you're looking to become a programmer who can always be assured of a stable career and get well compensated for it, then you want to learn JavaScript. In this book I show you how.