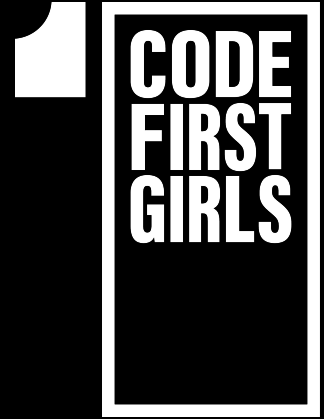


WELCOME TO CFG **YOUR INTRODUCTION** **TO JAVASCRIPT**



TECH SHOULDN'T JUST BE A BOYS CLUB.

COURSE JOURNEY

MODULE 1: JAVASCRIPT

INTRO
JAVASCRIPT



MODULE 01

CONDITIONS
& LOGIC

MODULE 02

THE DOM

MODULE 03

INTRO
REACT

MODULE 04

REACT
COMPONENTS

MODULE 05

STYLING
COMPONENTS

MODULE 06

STATES &
EVENTS

MODULE 07

PROJECT
PRESENTATION

MODULE 08

What is Javascript?

Javascript Variables

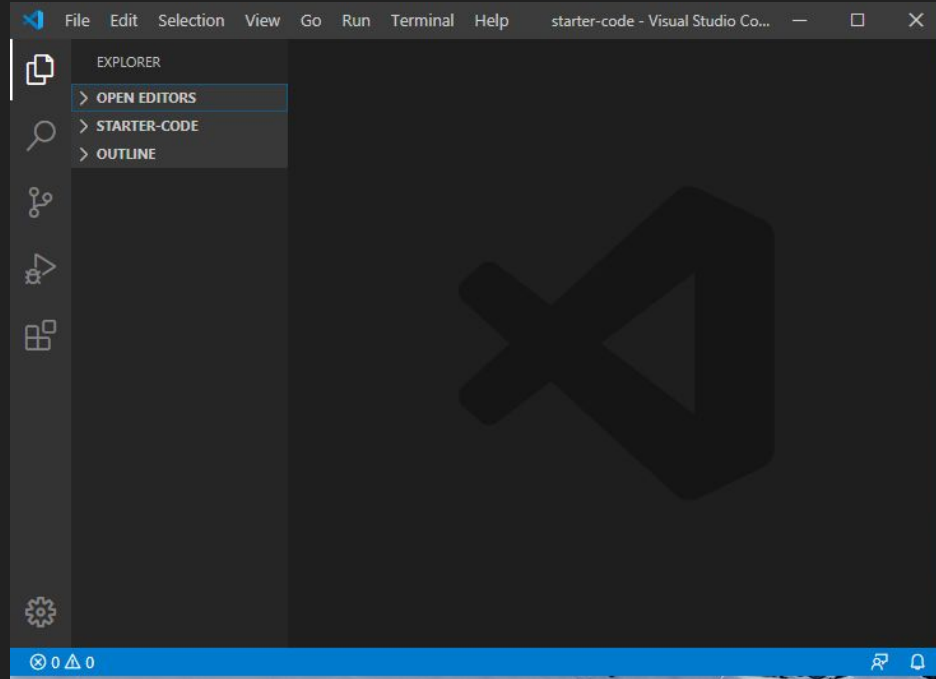
Javascript Data Types

Working with Data in Javascript

Complete interesting practical exercises

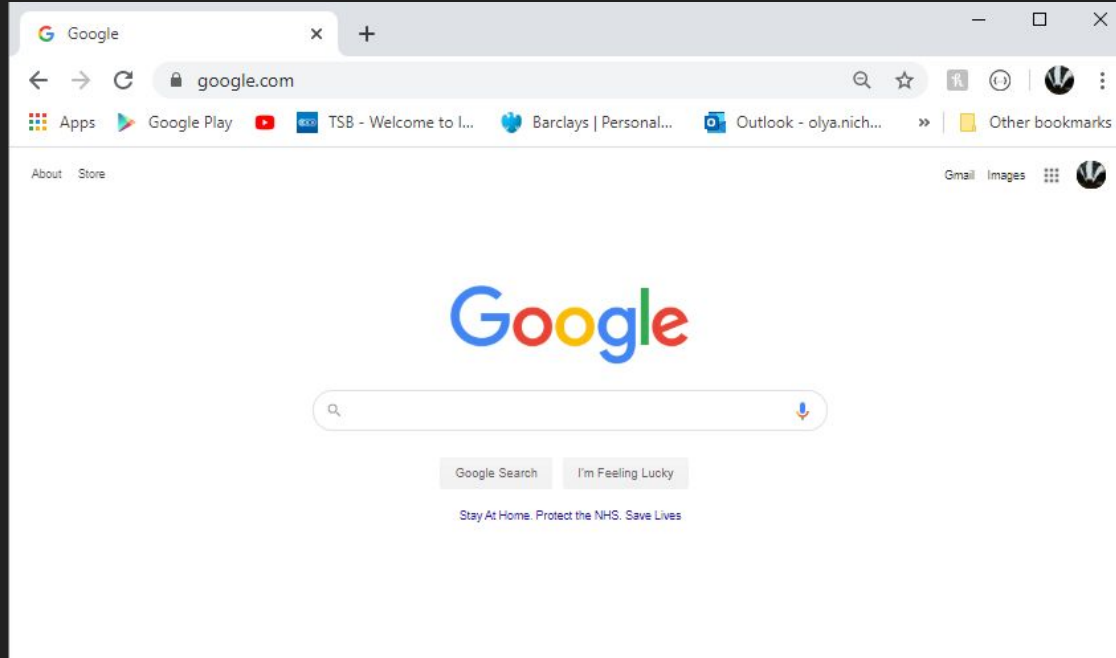
GETTING STARTED - TOOLS

VISUAL STUDIO



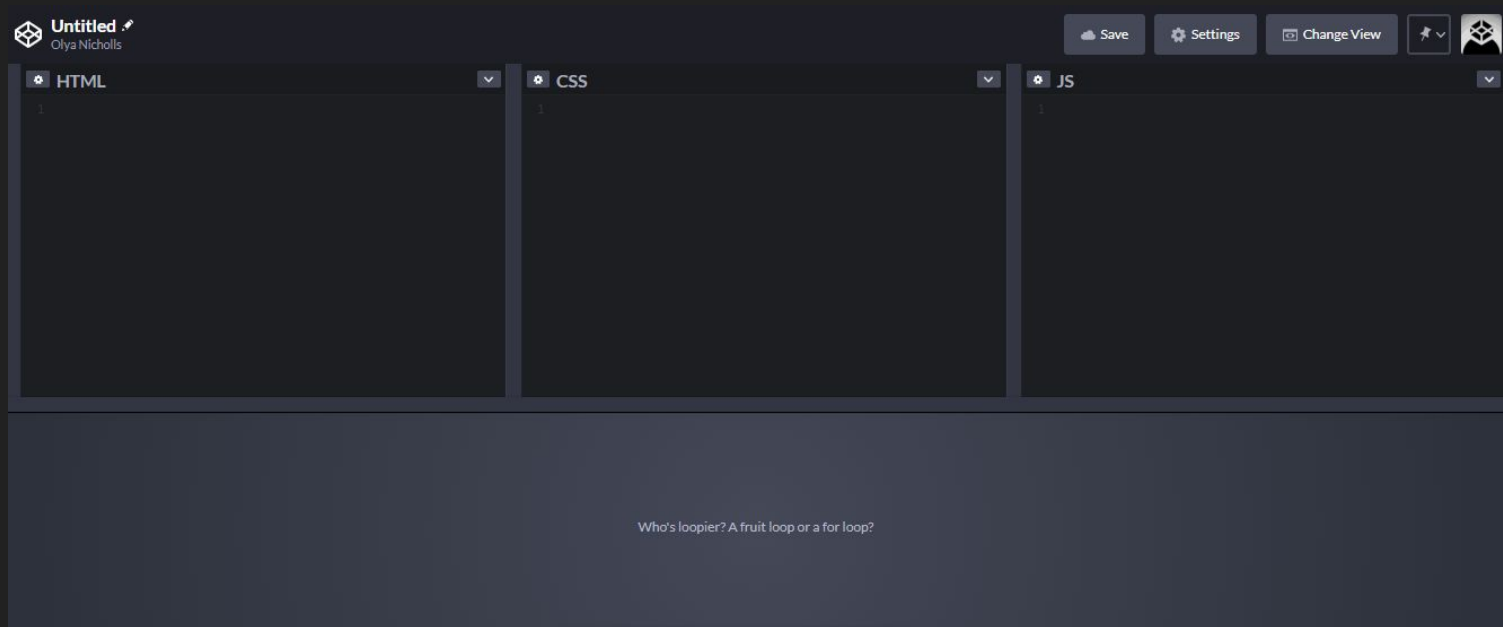
GETTING STARTED - TOOLS

CHROME BROWSER



GETTING STARTED - TOOLS

CODEPEN




WHAT IS JAVASCRIPT?

Javascript is known as **the language of the web**. It works alongside HTML and CSS to give websites interactive features, and also is commonly used to work with stored data or API's.

It's made up of **data types, variables and functions** and is a **high-level language**, meaning that it's easier to use and incorporates some natural language elements.

We can do a vast array of things with Javascript, including adding **animation**, working with **user input**, **storing data** given to us by a user, and generally making websites **dynamic**. Dynamic websites mean that they can display different content to different users, as opposed to static websites that always stay the same.

A large, bold, black 'JS' logo is centered on a solid yellow rectangular background. The letters are thick and sans-serif, with the 'J' and 'S' being slightly larger than the 'J' alone.

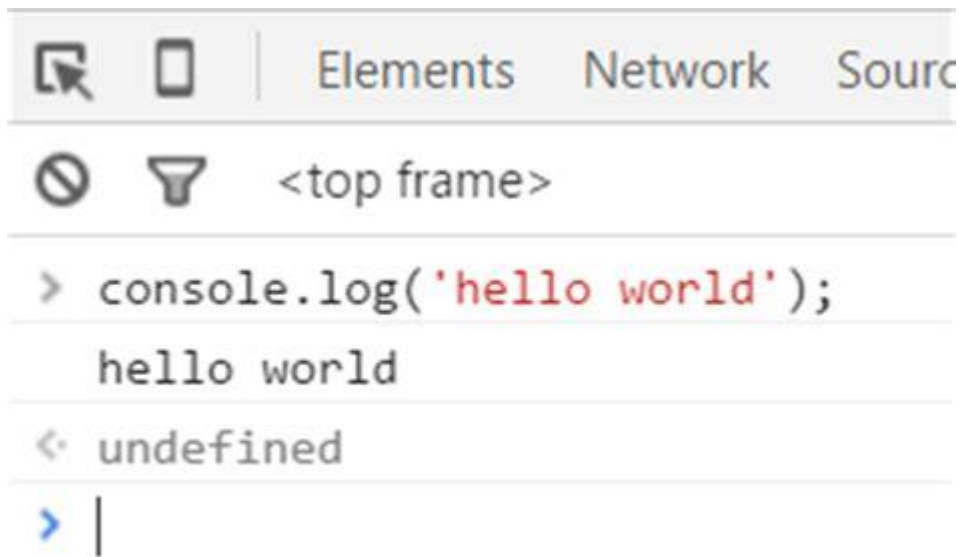
THE BASICS OF JAVASCRIPT: CONSOLE.LOG()

Before we learn more about the syntax and structure of javascript as a language, we're going to look at one of its most important built-in tools.

This is **console.log()**, a function that comes built-in to javascript, meaning we don't have to create it ourselves. You can recognise that it's a function because of the brackets at the end.

Console.log is essentially a print statement.

Whatever is inside those brackets, whether a variable or a string, i.e "Hello World" will be printed in the console.



TEST OUT CONSOLE.LOG() IN YOUR BROWSER BY USING THE DEVELOPER TOOLS UNDER THE VIEW TAB. SELECT YOUR JAVASCRIPT CONSOLE AND OFF YOU GO!

THE BASICS OF JAVASCRIPT: VARIABLES

A variable in Javascript is essentially a **container** that holds **reusable data**. The most popular way to think of them is like boxes. We store different data types within these boxes, and then label them so that we can use them again and again.

For example, think of a cardboard box. We can store whatever we want inside it, like the solution to a problem, a number or anything else we can think of. If we intend to store apples, pears and bananas in the box, we might want to write 'fruit' as the label, or variable name.

The significance of the variable is its label. You can change what's inside this box, but not the label. It's only a name given to a memory location.

HOW TO CREATE VARIABLES

Traditionally, variables have been created using the **var** keyword, followed by the name we want to give to that container. We then use the = sign to assign a value to the variable. In this case, we are using the word apple, and placing it inside a box called fruit. With ES6 JavaScript, we have since replaced this with **let** and **const**.

```
var fruit = “apple”;
```

NOW, INSTEAD OF WRITING “APPLE” AND ADDING IT AS A NEW ITEM EVERY TIME IN OUR CODE, WE CAN JUST REFERENCE “FRUIT” AND GET THIS VALUE INSTANTLY!



Variable Syntax & Structure

variable identifier
↓
var name = 'James Bond';
↑ ↑ ↑
start with assignment value
operator

End of the statement
↙

ALWAYS MAKE SURE THE VARIABLE NAME MAKES SENSE AND PROPERLY REPRESENTS THE VALUE YOU WANT TO STORE!

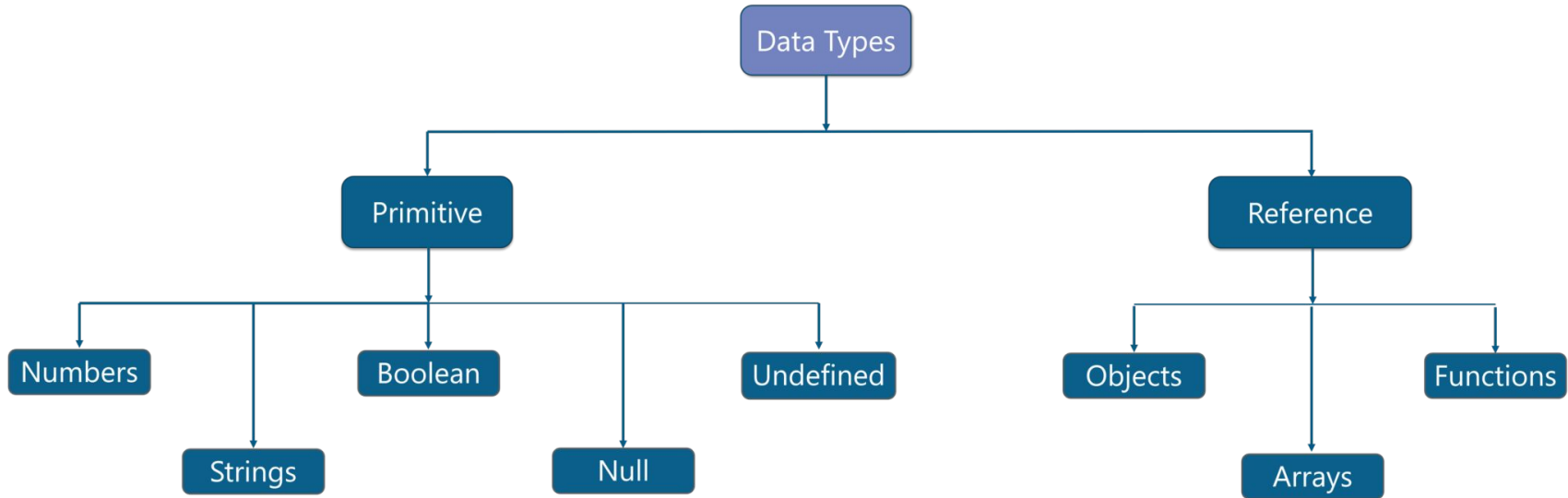
VARIABLES: LET AND CONST

We know that we can create a variable using the **var** global keyword, which can be redefined and reassigned. However, as previously mentioned, this is a bit old fashioned! We have two different keywords for variables that we can use to make our code more efficient. These are **let** and **const**. The simplest way to distinguish between them is as follows:

- **Let** is the closest to var - it has a scope that can be restricted to within its own block of code. **Lets are also mutable**, meaning the value can be reassigned.
- **Const** is a **constant value - it cannot be redefined or reassigned**. It also has a local scope, meaning it is restricted to a block of code.

```
1  const name = 'Nia';
2  let age = 25;
3  // can reassign age
4  age = 27;
5  // cannot reassign name
6  name = 'Jane';
7
8
```

THE BASICS OF JAVASCRIPT: DATA TYPES



Data types are just what they say they are - **types of data we use in our programmes**. They are divided into two categories, primitive and reference. Some of the most common data types we will be using are **numbers** (1,2,3), **booleans** (true or false) and **strings** ("isn't this fun?"). We can then assign these values to variables...

JOINING STRINGS

In Javascript, there are particular ways we can join variables. In the case of strings, these methods are **concatenation** and **interpolation**.

Concatenation uses the + sign to join two string values together, as you can see below.

Interpolation is when you use **template literals** to join pieces of data together. You can recognise template literals as they use the \$ sign, followed by the curly brackets { }.

CONCATENATION

```
var name = 'Jane';  
var greeting = 'Hi' + name;
```

INTERPOLATION

```
var name = 'Jane';  
var greeting = `Hi ${name}`;  
# Be sure to use backticks!
```

LINKING JAVASCRIPT TO OUR PROJECT FILES

```
<!DOCTYPE html>
<html>
<head>
  <title>javascript link</title>
</head>
<body>
<h1>Here is your code</h1>
<script src="yourfile.js"></script>

<script type="text/javascript">
  alert('Hello coder');
</script>

</body>
</html>
```

There are two ways to use Javascript with your project files. These both rely on using the `<script>` tags within HTML.

OPTION ONE:

We create a .js file within our project folder. Then, within our html page, we identify a script source i.e where our html file can go to find our js file. We select the js file we've created. And done! Now everything you have in that file can be accessed within the HTML doc.

OPTION TWO:

We create some Javascript code within the body of our HTML doc itself. We specify the type, i.e. javascript and then within those tabs, we create our code directly.

Whichever method you use, it is VERY important to always link/add your JS at the END of the body so that the HTML has fully been loaded before the JS can start affecting it!

NOW LET'S PRACTICE TOGETHER

LINKING JS AND CREATING VARIABLES

MODULE 1: JAVASCRIPT

5 MINS

Exercise 1.0 - LINKING JS

*Create an HTML file and a JS file and link them to each other. Let's open our file in Google Chrome and open the Developer Tool and go to the Console tab. This is where we will see our JS in action!

Exercise 1.1

Within the JS file, create a new variable called **myName** and assign it with the value of your name. Use your new knowledge of var, let and const.

Exercise 1.2

Create a new variable and **store your age**.

Exercise 1.3

Create a new variable called "future job" and assign it "web developer".

Exercise 1.4

Connect these bits of information together with either concatenation or interpolation, so that it creates the phrase **"My name is (name), I am (age) years old and I will soon become a web developer."** and log it in the console! Save and refresh the page in Chrome to see our new code working in the console!

10 MINS

GROUP EXERCISE



JAVASCRIPT BASICS: WORKING WITH DATA

Now that we know how to store single elements of data within variables, let's look at how to work with groups of data.

Data is the name we give to information and is often focused on **grouping**.

In coding, this process closely resembles creating data structures. **Data structures are where we store valuable information** that can then **be called upon, edited or adjusted** to meet our requirements throughout a programme.



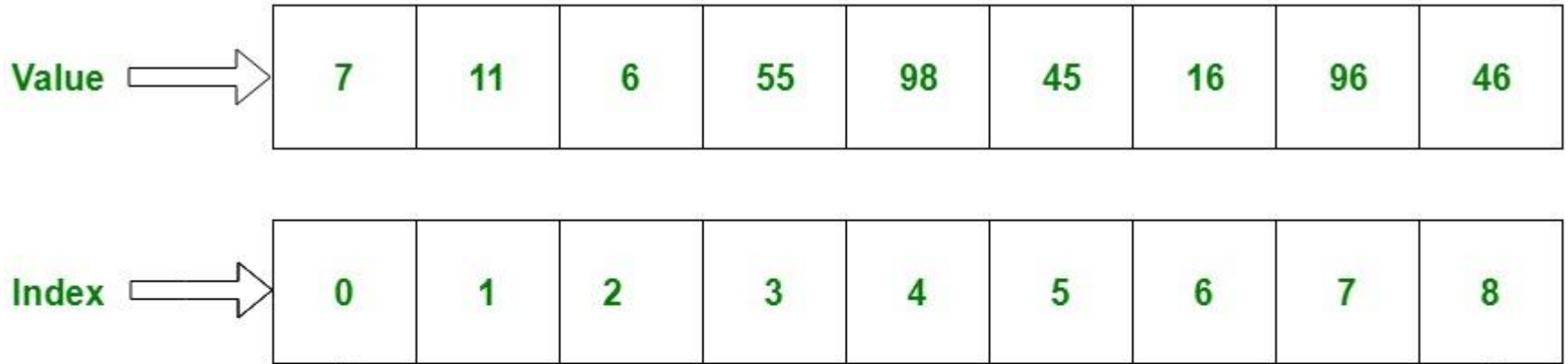
Structure of Javascript Arrays

```
1  let fruits = ["apple", "plum", "cherry"];
2
3  console.log(fruits[0])
4  console.log(fruits[1])
5  console.log(fruits[2])
6
```

Here, we are storing different fruits into one grouping known as an **array** - named "**fruits**". Think of it as a **variable but for multiple items**. Now, we could simply print the entire array to the console, however, we can also call each individual element through a process called **indexing**. The result of the above statements are: apple, plum, and cherry, respectively.

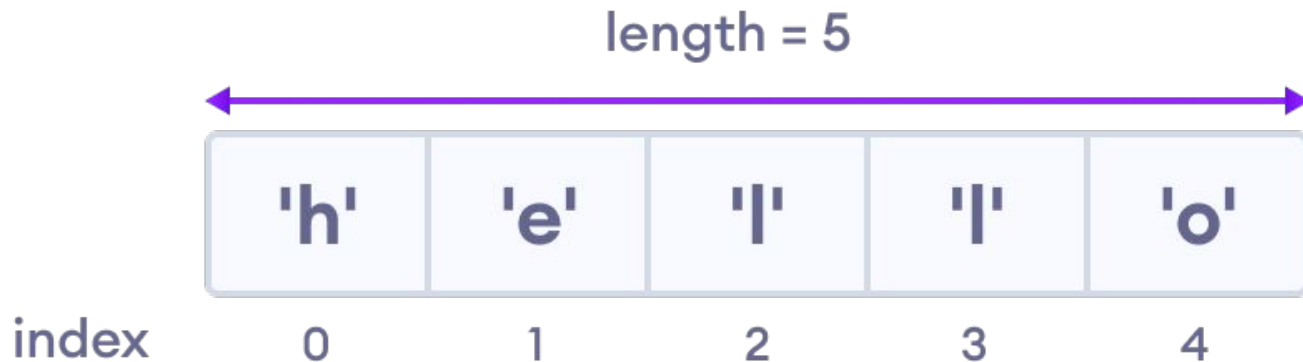


Indexing in Javascript



Indexing in JavaScript refers to the process of accessing elements in an array or a string using their position, or "index", in the data structure. JavaScript arrays are **zero-indexed**, which means that the first element of the array has an index of 0, the second element has an index of 1, and so on. You can access an element in an array by referencing its index in square brackets after the array name, like this. For example, element 2 would be 6, and 5 would be 45.

String Slicing in Javascript



Now that we have a grasp of indexing for arrays, we can also utilise it when working with other data types. For example, we can use a process called 'slicing' to sort through strings or string-based variables. Slicing is when we extract a portion of the string, creating a new "**substring**". Let's use the word 'hello' - if we wanted to get only the letters 'ell', we would need to cut out the elements between index 1 and 4.

In JavaScript, the **slice()** method works by specifying the **start and end index of the substring** you want to extract. The start index is included in the substring, but the end index is not.

So when you specify an end index of 4, the method will include the characters at indexes 1, 2, 3, but not the character at index 4. The character at index 4 is not included in the substring.

```
1  let str = "hello";  
2  let substring2 = str.slice(1, 4);  
3  console.log(substring2); // prints "ell"  
4  
5
```

NOW LET'S PRACTICE TOGETHER

WORKING WITH DATA STRUCTURES. DISCOVERING PROMPT, PUSH AND POP!

5 MINS

Exercise 2.0

Create a programme that lists your favourite colours as an array. Use the `console.log()` statement to show the first element and the last element.

9 MINS

Exercise 2.1

Let's modify this programme to get a new colour from the user and add it to our array. Using the **`prompt()`** input, get a new colour from the user and then use the **`push()`** command to add this new variable to the favourite colours array. Then use `console.log()` to show that this change has been made correctly.

Exercise 2.2

Let's change our minds and remove that last colour from the array. Do this using the **`pop()`** method.

GROUP EXERCISE



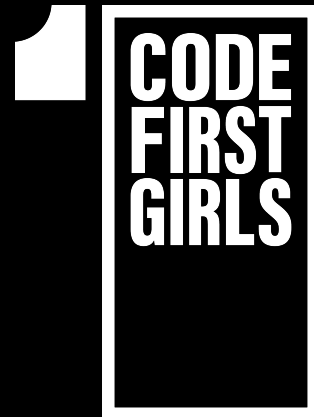
HOMEWORK

+ Homework Task

Let's try to create new slang words from a list of letters. Create an array of all the letters of the alphabet. Use indexing, slicing and any methods you study to create the words "hi", "def" and "nope".

Explore the additional material (slides 26 - 29) to learn more about storing data in JS: read up on lists and dictionaries!

THANK YOU
HAVE A GREAT
WEEK!



**ADDITIONAL
MATERIAL**

Grouping Data

Lists and objects are used to store and organise data - similar to the arrays we've just seen.

lists - in most cases, a list is an array. You use square brackets to create it and all elements within that group can be called together.

Objects - one of the most powerful data structures available to us. Similar to the idea of a dictionary in other languages, they use key-value pairs for more specific access to stored data, and are created using { } curly brackets.

Lists - Structure

Lists can be distinguished by looking for their square brackets []. Let's take a look at the list below to get a better understanding of how they work!

So, this variable is called 'ages' and stores a collection of data that is tied to this name. As you can see, it's currently unordered (but we can use tools later on to do this if we need to).

All items within this list are separated by commas, and correspond to their index points.



```
1 const ages = [12, 17, 19, 18, 22, 76, 45, 43, 32, 7];  
2
```

Object Notation

These objects can be distinguished by looking for their curly brackets { }. Let's take a look at the object below to get a better understanding of how they work!

So, this variable is called 'test_dictionary' and stores a collection of data. As you can see, a key part of object is their use of key-value pairs, each separated by a :. For example, the 'this is a key' is a key, and the 'this is the value of this key' demonstrates the value.

You can even store objects inside one another, which is called nesting!

```
1 var test_dictionary = {  
2   "This is a key" : "This is the value of this key",  
3   "You can make many keys" : {  
4     // You can even nest dictionaries in one another  
5     "Integer" : 123,  
6     "String" : "Hello, world!",  
7     "Boolean" : true,  
8     "Array" : [1,2,3,4,5]  
9   }  
10 }
```