WELCOME TO CFG
YOUR INTRODUCTION TO JAVASCRIPT

CODE FIRST GIRLS

TECH SHOULDN'T JUST BE A BOYS CLUB.

COURSE JOURNEY

INTRO JAVASCRIPT	**CONDITIONS & LOGIC**	THE DOM	INTRO REACT	REACT COMPONENTS	STYLING COMPONENTS	STATES & EVENTS	PROJECT PRESENTATION
MODULE 01	MODULE 02	MODULE 03	MODULE 04	MODULE 05	MODULE 06	MODULE 07	MODULE 08

02 JAVASCRIPT

Logical Operators

Comparison Operators

If / *else* / *else if* statements

Loops - For & While

Functions

WHAT DO WE MEAN BY LOGIC?

In programming, logic means identifying the similarities and differences of elements to identify how they operate together to perform a specific task.

That might sound complicated, but really all it means is determining the relationship between data - whether they're connected or not, if one is greater than another, if they match types or cannot be concatenated.

To do this we use **logical and comparison operators.**

These operators come into play when we use **conditional statements**.

WHAT IS A CONDITION?

In programming, conditional statements are rules that we create to determine how our code runs. Depending on our conditionals, some bits of code might run, and others may not.

There are different types of conditional statements, but the most important for you to know are:

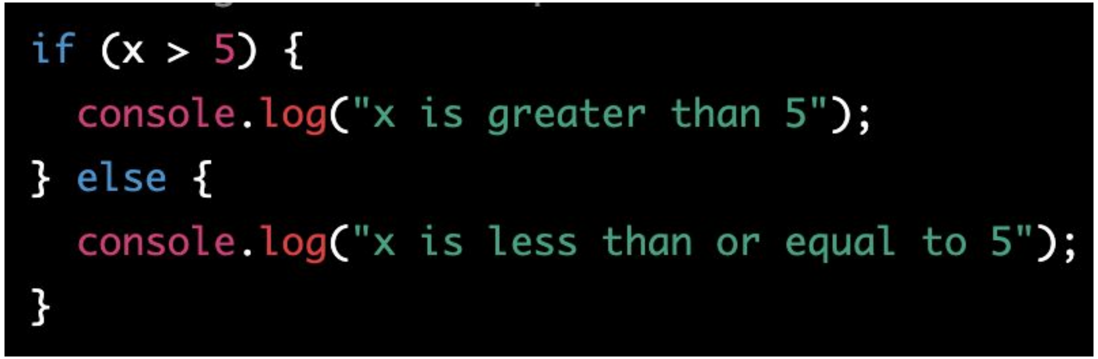
- **“If” statements:** where if a condition is true it is used to specify execution for a block of code.
- **“Else” statements:** where if the same condition is false it specifies the execution for a block of code.
- **“Else if” statements:** this specifies a new test if the first condition is false.

CONDITIONAL STATEMENTS CONTINUED

We use conditional statements to **control the flow of our code**, giving us more options for what situations we need to solve and address. These conditional statements work based on **boolean values** - whether something is true or false.

Let's say that you want to check if a number is greater or smaller than another.

In this instance, you might use **if, else and else if statements** to check the value of x (our number) with the value of 5. You then have two ways of reacting to whether this is true or false.

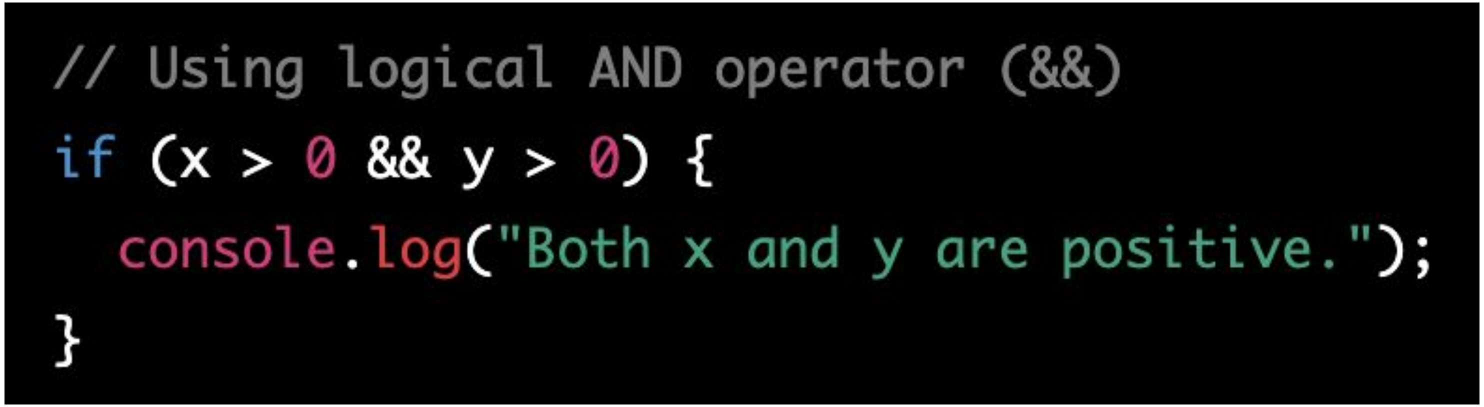


```
if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is less than or equal to 5");
}
```

LOGICAL OPERATORS

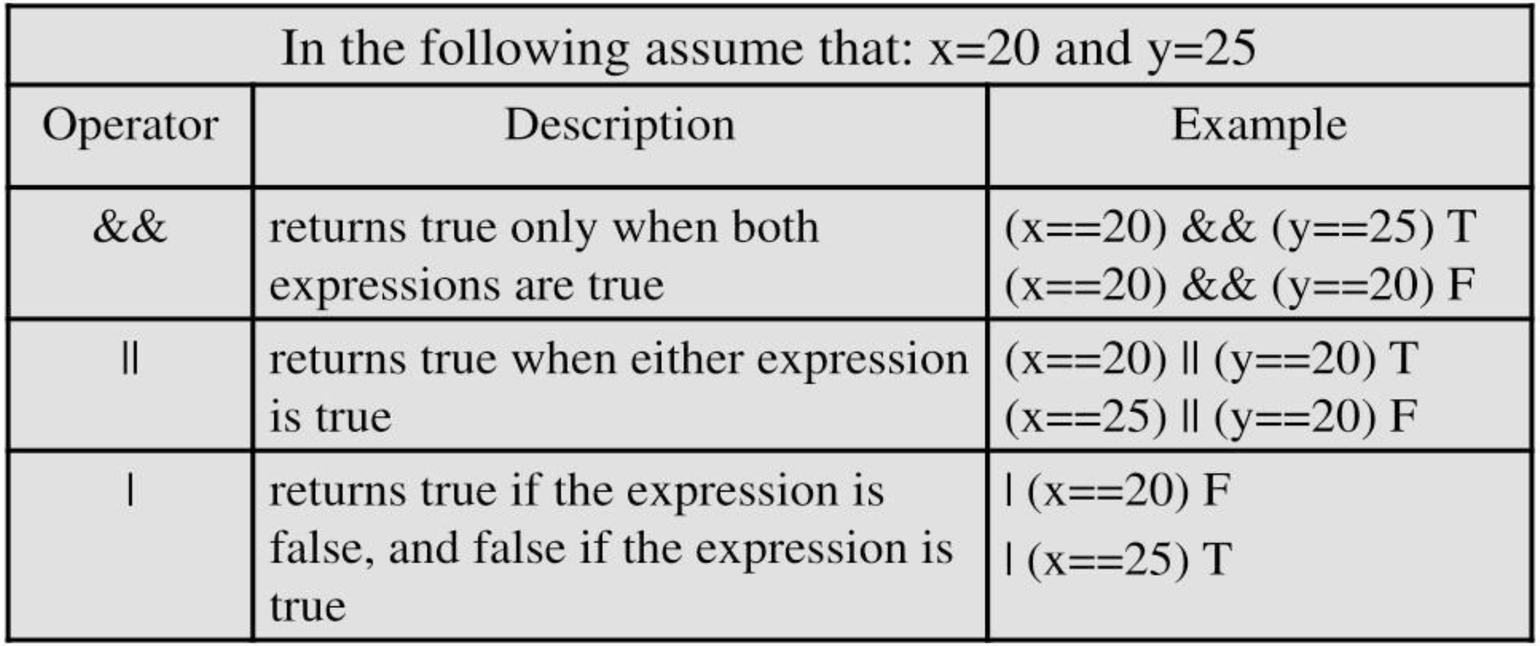
Logical operators are what we use to determine the logical relationship between values or variables. In basic terms, we use logical operators to make those options more complex. Instead of just looking at one bit of information, we can make more informed decisions using multiple bits of data.

- **&& - AND:** returns true if both arguments are either true or false.
- **|| - OR :** In case any of its arguments are true, it returns true, if not, it returns false.
- **! - NOT:** inverses the value upon which it is used.



```
// Using logical AND operator (&&)
if (x > 0 && y > 0) {
  console.log("Both x and y are positive.");
}
```

LOGICAL OPERATORS



In the following assume that: x=20 and y=25		
Operator	Description	Example
&&	returns true only when both expressions are true	(x==20) && (y==25) T (x==20) && (y==20) F
||	returns true when either expression is true	(x==20) || (y==20) T (x==25) || (y==20) F
|	returns true if the expression is false, and false if the expression is true	| (x==20) F | (x==25) T

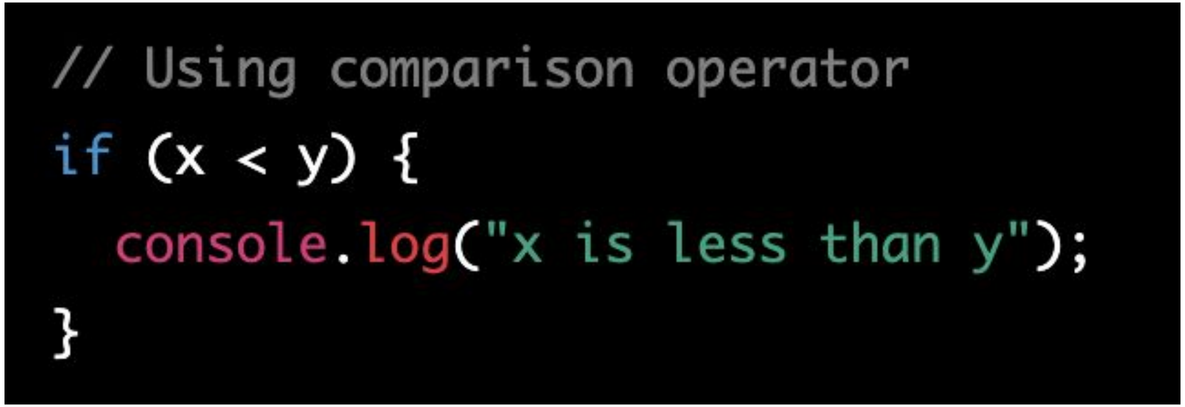
- ***Note:*** **Use logical and comparison operators when you want your program to act different under different conditions**

COMPARISON OPERATORS

In real life, you should never compare yourself to others. But in code? You should absolutely compare one value to another. This is how we gauge what's similar and different between them, helping us to make better decisions about how that information can work together.

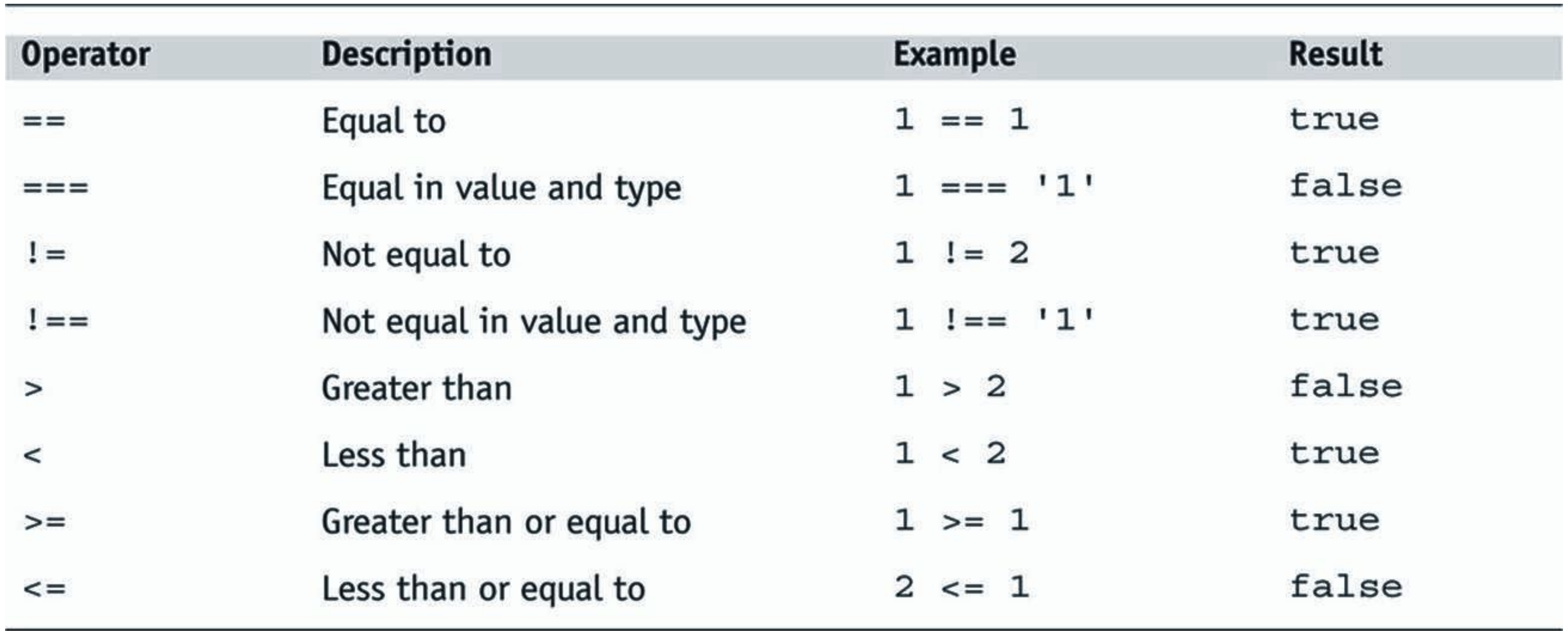
You already know a few comparison operators in everyday life - especially in the realm of mathematics. These core concepts carry over into code:

- **GREATER THAN**
- **LESSER THAN**
- **EQUAL TO**
- **NOT EQUAL TO**



```
// Using comparison operator
if (x < y) {
  console.log("x is less than y");
}
```

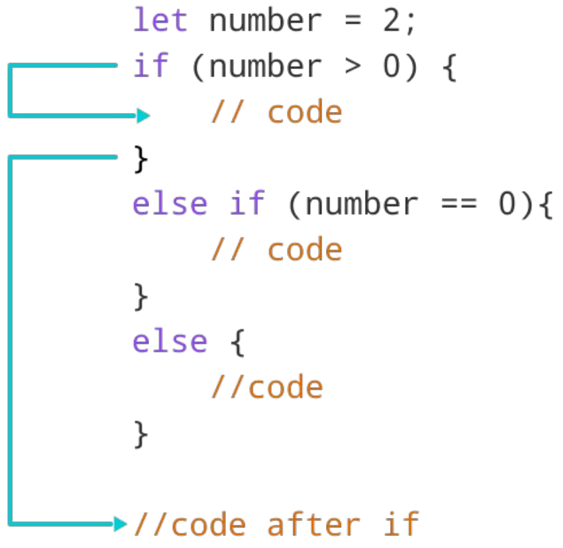
COMPARISON OPERATORS



Operator	Description	Example	Result
`==`	Equal to	`1 == 1`	`true`
`===`	Equal in value and type	`1 === '1'`	`false`
`!=`	Not equal to	`1 != 2`	`true`
`!==`	Not equal in value and type	`1 !== '1'`	`true`
`>`	Greater than	`1 > 2`	`false`
`<`	Less than	`1 < 2`	`true`
`>=`	Greater than or equal to	`1 >= 1`	`true`
`<=`	Less than or equal to	`2 <= 1`	`false`

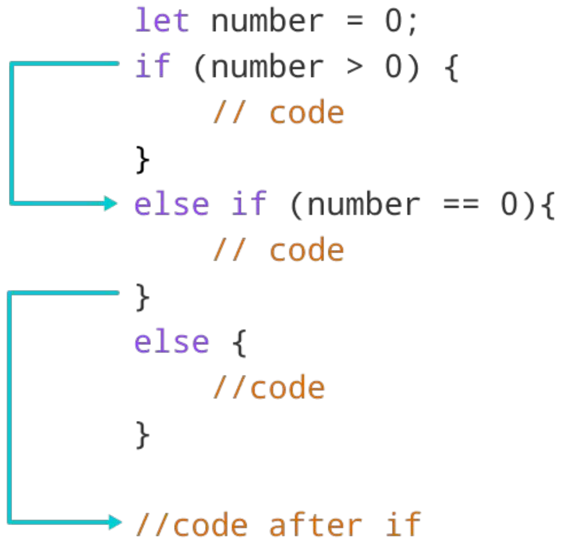
CONDITIONAL FLOW

1st Condition is true



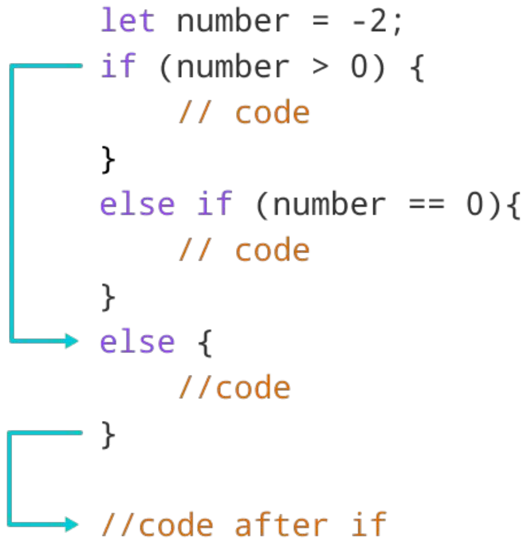
```
let number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

2nd Condition is true



```
let number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

All Conditions are false



```
let number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

NOW LET'S PRACTICE TOGETHER

USING CONDITIONAL STATEMENTS IN OUR CODE

5 MINS

Exercise 1.0 - LINKING JS
*Create an HTML file and a JS file and link them to each other

Exercise 1.1
Within the JS file, create three new variables called "genre" and "rating" and "pillows". Give these the values of "horror", "5" and "1", respectively.

9 MINS

Exercise 1.2
Imagine we are hosting a movie night. One of your friends is a huge scaredy cat, but the rest of you want to watch the new horror movie that's just come out. Create a conditional flow for your options.

- If the movie has a 5 star rating and is a horror AND you have a pillow, your friend will watch it.
- If the movie has a rating of 5 stars, is a horror but you have no pillows, your friend won't watch it.
- If the movie has a rubbish review, there's no point watching it.

GROUP EXERCISE

CONDITIONS AND LOGIC: LOOPS

We often need to repeat certain tasks while coding, whether it’s carrying out a function however many times or searching through bits of data one by one. This is why we use loops - just as the name suggests, they are contained parts of code that will repeat based on certain conditions.

There are two main types of loops:

- **FOR:** Used when we know **exactly how many times** we want a piece of code to run.
- **WHILE:** Used when we want to **break the loop when it meets a condition**, not a certain number of times.

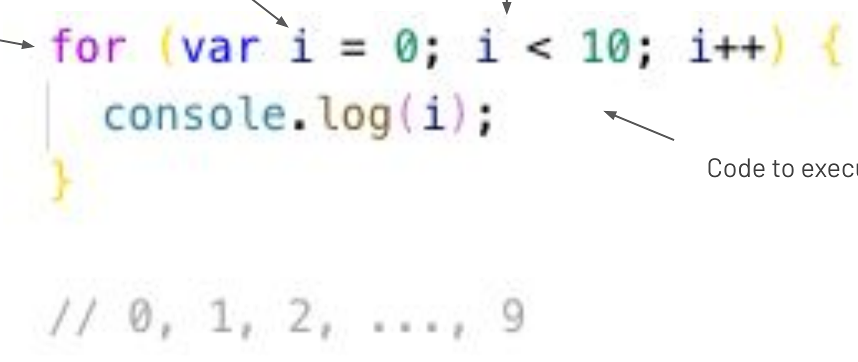
THE STRUCTURE OF A FOR LOOP

Condition
If this evaluates to true, the code in the block will run

Initialiser
Sets an initial value

Modifier
Changes the value for the next loop around

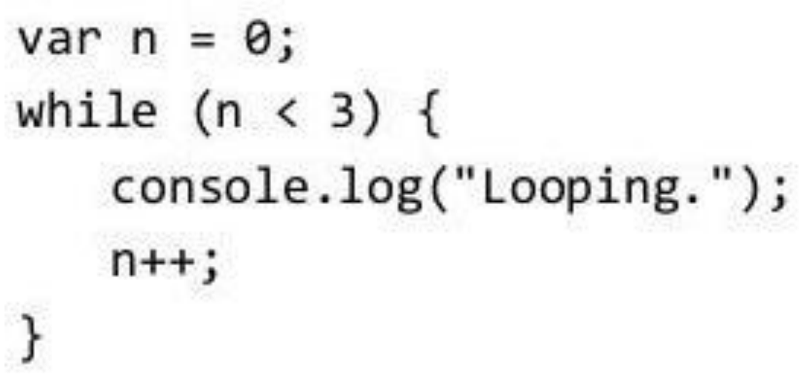
Keyword
Declares that we're building a for loop



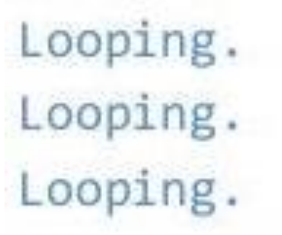
```
for (var i = 0; i < 10; i++) {
  console.log(i);
}

// 0, 1, 2, ..., 9
```

Code to execute

THE STRUCTURE OF A WHILE LOOP



```
var n = 0;
while (n < 3) {
    console.log("Looping.");
    n++;
}
```



```
Looping.
Looping.
Looping.
```

Here, instead of specifying the number of loops needed in the conditional, we instead give it a rule.

While n is less than 3, keep looping. So we rely on conditions only - **it keeps running so long as we haven't hit 3.**

When we we meet this condition, the loop ends.

LOOPS: POINTS TO CONSIDER

Both for loops and while loops rely on some kind of initial index. For a for loop, this will be specified within the loop parameters, whereas a while loop will often take it from an outside variable.

While loops are best used when **SEARCHING** for something.

For loops are best for **REPEATING** what you already know.

Remember that while loops go on as long as the condition is true, so they can potentially be infinite! This is why it’s so important to know how to write them, to **prevent an inescapable loop**.

NOW LET'S PRACTICE TOGETHER

USING LOOPS IN OUR CODE

5 MINS

Exercise 2.1 - We have a lot of students in this course! Let's create a for loop to create a student tracking system so we know How many students we have. In your js file, create a list variable called **students**. Fill it with the names "Sofia", "Elizabeth", "Sasha", "Samantha", "Abigail", "Lorena", "Ayesha", "Adeyo", "Emil"

9 MINS

Exercise 2.2 - Create a for loop to count through the number of students within the students array. *Hint:* Use the .length method instead of counting the number of students manually!

Exercise 2.3 - Let's say we can only run a small class. Create variables for **class size** (0), **maxClassSize** (7) and an empty **classList**. Use a while loop to add random students to the classList so long as the number of students in the new class is less than the maximum size.

GROUP EXERCISE

PUTTING IT ALL TOGETHER

Now that we're creating more complex programmes, let's talk about how we can use them repeatedly, without having to write them out again and again.

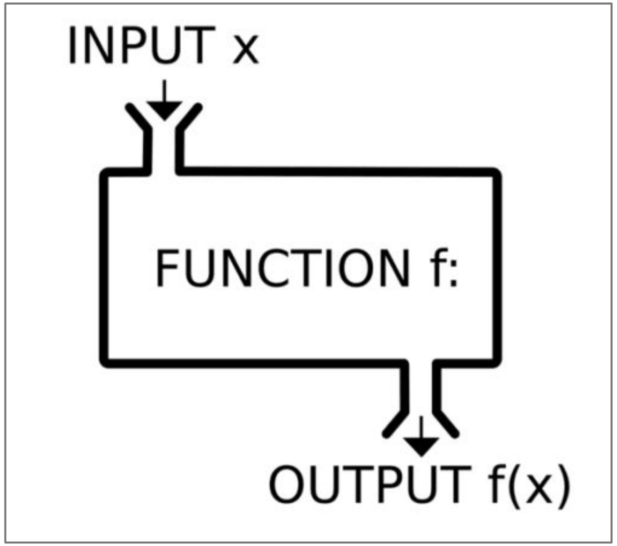
We do this with something called a **function**, which allows us to take all of what we've learned so far and **store it as a kind of tool**, **which we can call throughout the course of our code**.

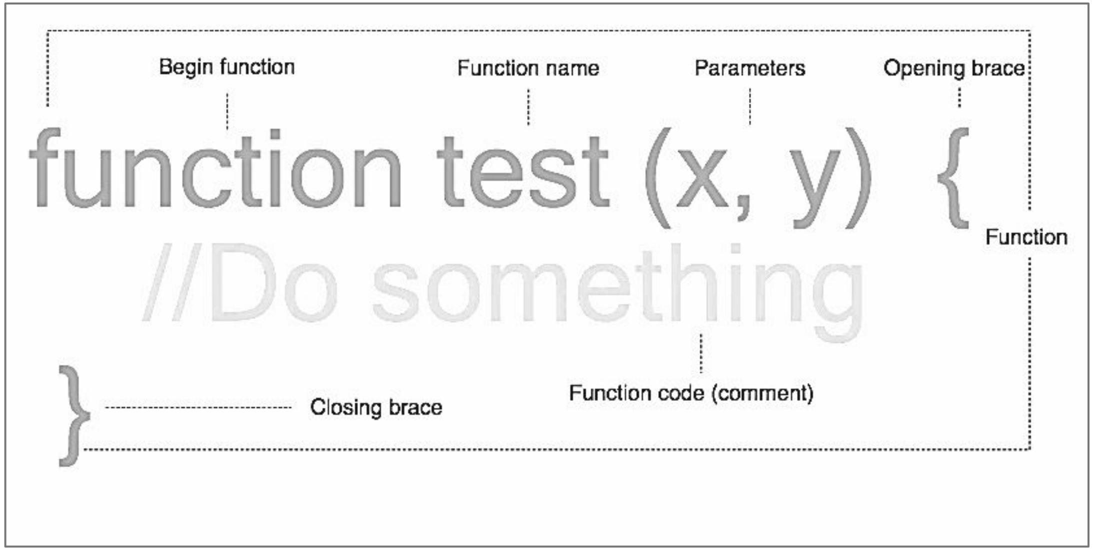
Functions allow us to...:

- Do more with our code.
- Carry out tasks faster
- Reuse different tools with new information

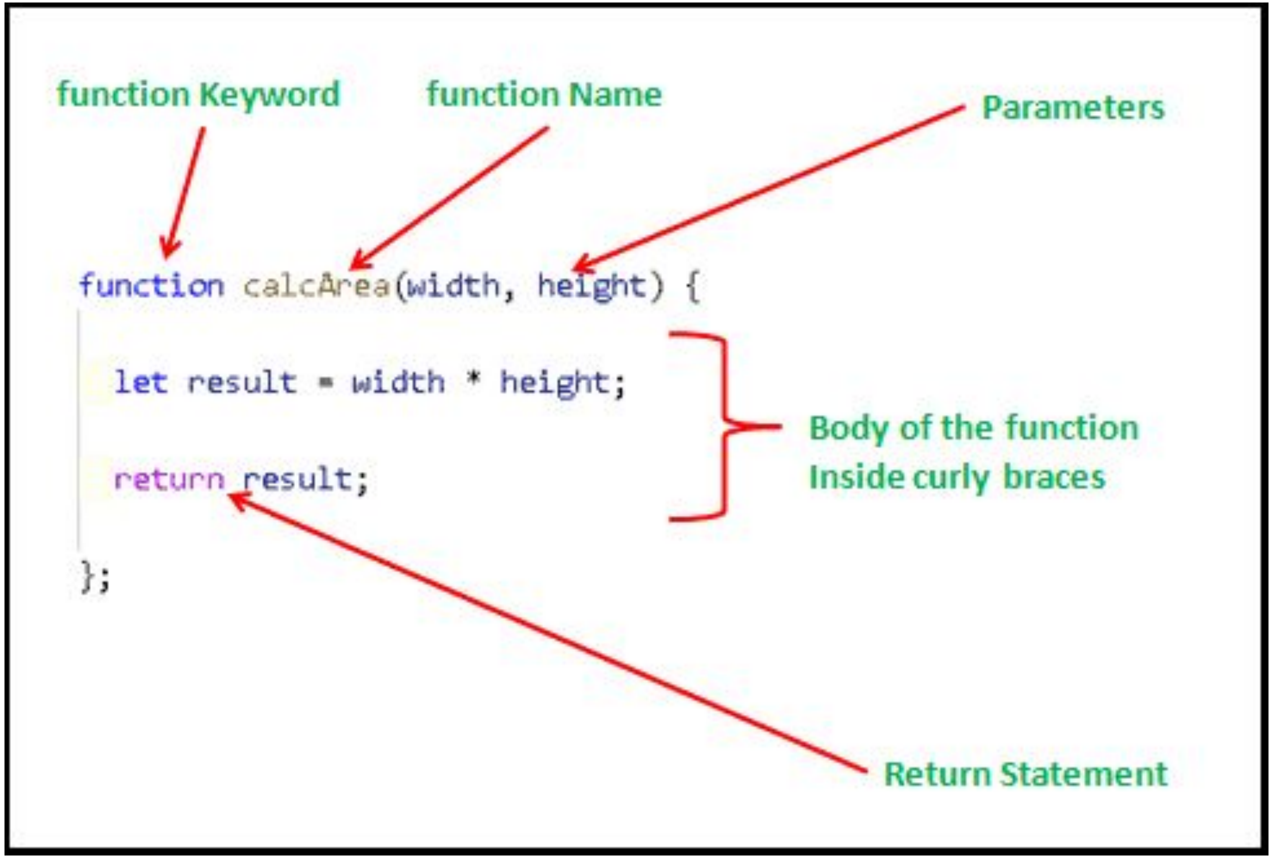
JAVASCRIPT BASICS: FUNCTIONS

A function is basically a reusable block of code. It can include its own variables and operations, and often has parameters so that you can feed it information and get new information in return. Functions are some of the most powerful elements in any programming language. Console.log() is one such function we've seen already. You can spot a function by the use of these brackets. In its simplest terms, think of a function as a reusable tool to perform an action or operation.



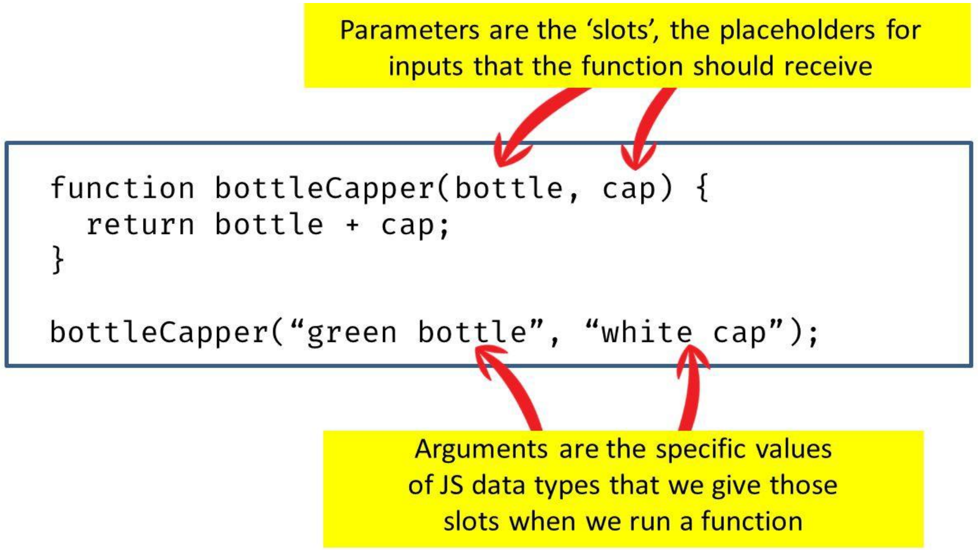


Function Syntax & Structure



```
function calcArea(width, height) {

  let result = width * height;

  return result;

};
```

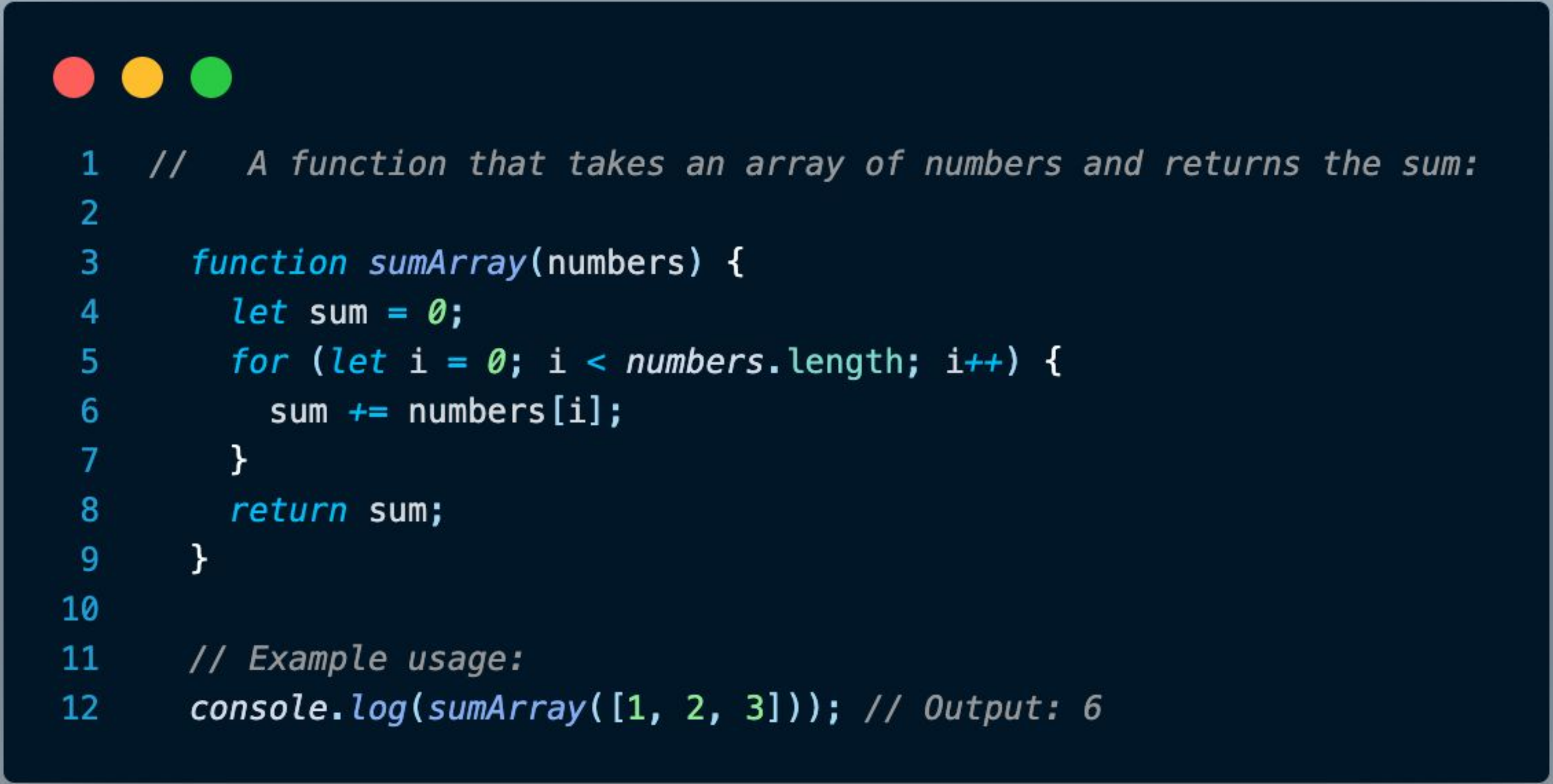
Function Syntax & Structure


Parameters are the 'slots', the placeholders for inputs that the function should receive
Arguments are the specific values of JS data types that we give those slots when we run a function

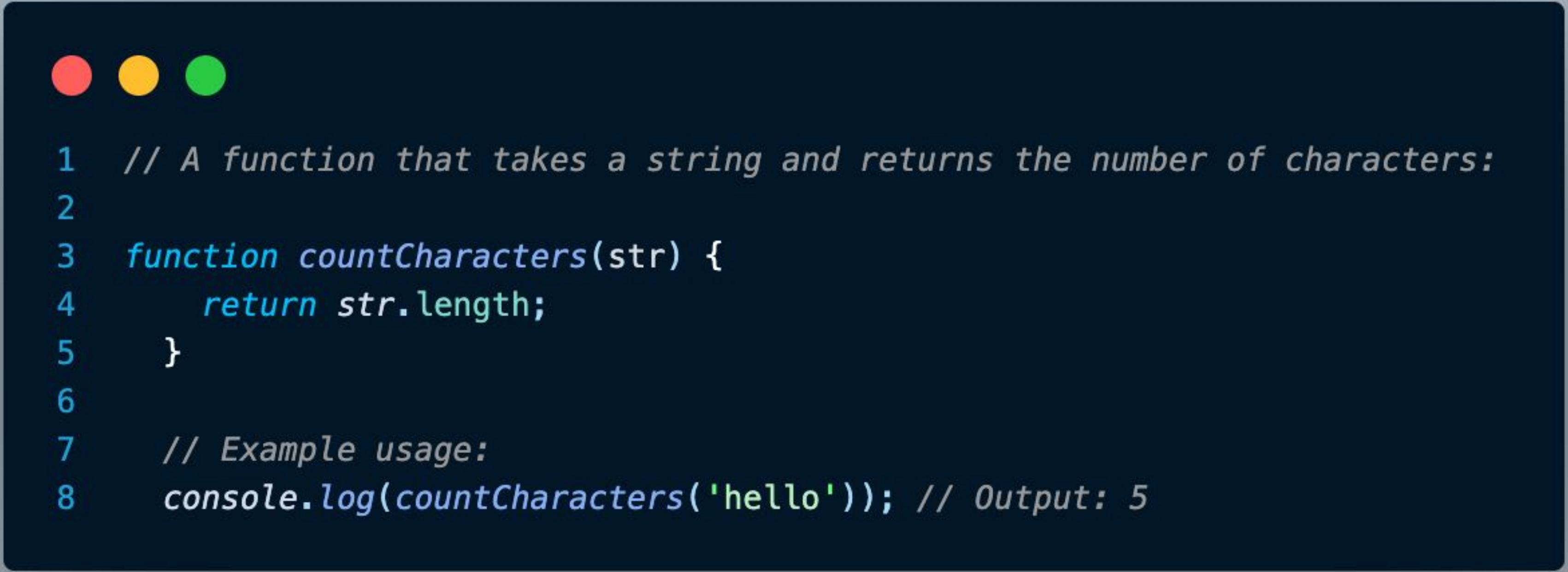
```
function bottleCapper(bottle, cap) {
  return bottle + cap;
}

bottleCapper("green bottle", "white cap");
```

Functions allow us to do a certain task or discover a value in a simple, repeatable way. As a programmer, you will regularly use functions - however, a lot of people don't fully understand a key point about them.

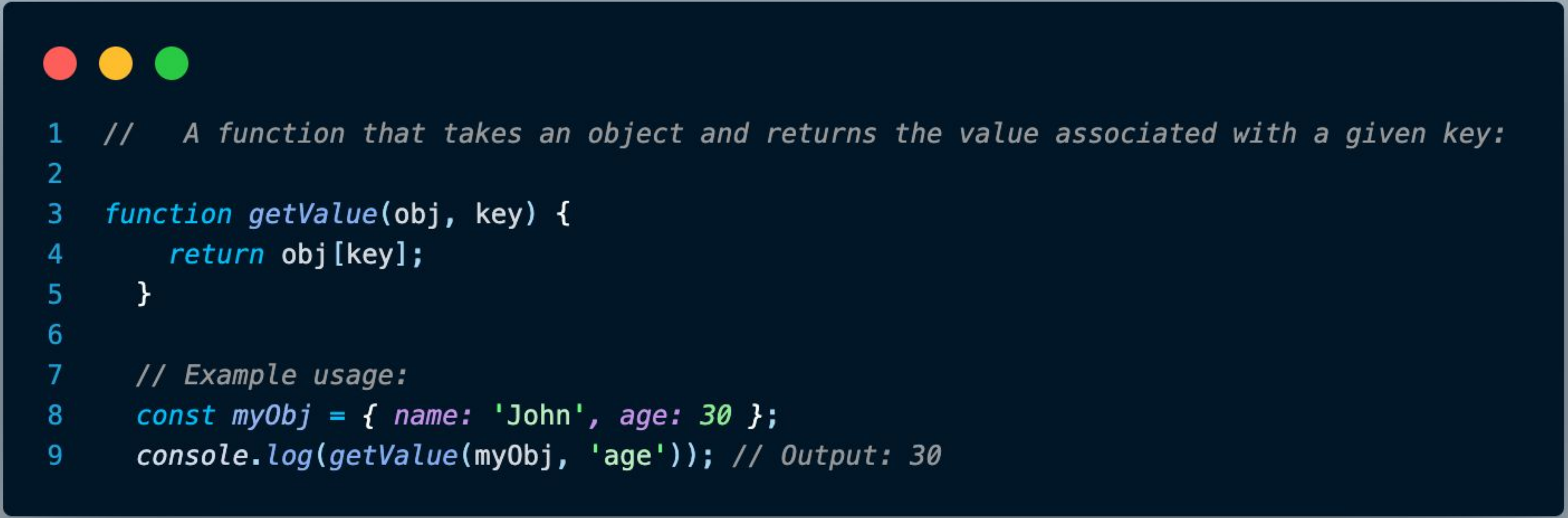
The input we give to the function is called a **parameter**. It's like a variable that's only meaningful inside the function. When calling this function, that's when we declare the value of the parameter. That is what we call an **argument**. These distinctions are important, as to refer to them as one in the same would be like thinking a variable name and value are one in the same!



```
//   A function that takes an array of numbers and returns the sum:

  function sumArray(numbers) {
    let sum = 0;
    for (let i = 0; i < numbers.length; i++) {
      sum += numbers[i];
    }
    return sum;
  }

  // Example usage:
  console.log(sumArray([1, 2, 3])); // Output: 6
```



```javascript
// A function that takes a string and returns the number of characters:

function countCharacters(str) {
    return str.length;
  }

  // Example usage:
  console.log(countCharacters('hello')); // Output: 5
```



```javascript
//   A function that takes an object and returns the value associated with a given key:

function getValue(obj, key) {
    return obj[key];
  }

  // Example usage:
  const myObj = { name: 'John', age: 30 };
  console.log(getValue(myObj, 'age')); // Output: 30
```

NOW LET'S PRACTICE TOGETHER

CREATING AND CALLING FUNCTIONS

5 MINS

Exercise 2.0
Let's create a function to add two numbers together. They can be
any two numbers you can think of. To demonstrate this, we'll start by writing a function
called **addTwoNumbers.**

Exercise 2.1
The function should have two parameters, num1 and num2. It should then return these two
numbers added together.

5 MINS

Exercise 2.2
Test this function with the following numbers:

- 2, 2
- 9, 3
- 2938, 2

The results should be 4, 12, and 2940.

GROUP EXERCISE

HOMEWORK

+ **Homework Task - Quitting the Loop**
We know that while loops can be infinite. Create a do while loop that takes in user input and returns different log statement options, but only if the input statement is not stop. If the statement is stop, break the loop.

THANK YOU
HAVE A GREAT WEEK!

CODE FIRST GIRLS