ubiqum code academy

Introduction to JavaScript

JS

Let’s Go

What is JavaScript?

*JavaScript is a **programming language***
that can be included on web pages to make them more interactive.

You can use JavaScript to:

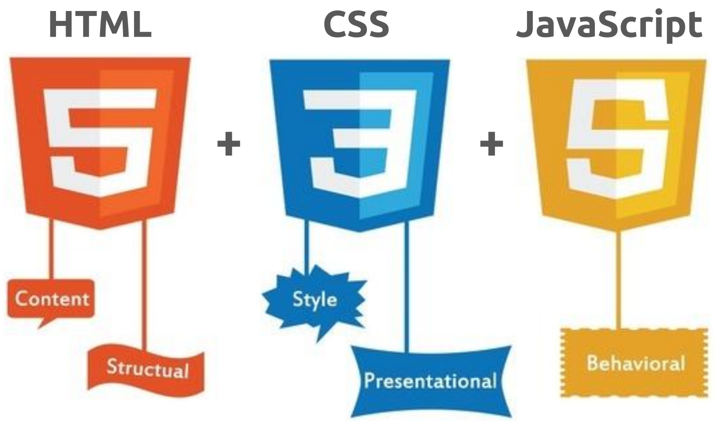
- check or modify the contents of forms
- change images, text, layout..
- open new windows
- handle and manage 'browser events'
- process data
- write dynamic page content
- **... and so many things!**

Taking control of our website **behaviour:**

“If we see HTML as the Skeleton of our webpage and CSS as the Skin that defines the appearance…

Now let’s take control of the behavior of our site! JAVASCRIPT is the brain! “

Pol Benedito, Full Stack Web Development Mentor at Ubiqum, Amsterdam



= Web Application!

JavaScript vs. **Java**

JavaScript has nothing to do with Java. JS and Java are completely different programming languages.

- **Client side**: Programs are passed to the computer that the browser is on, and that computer runs them.
- **Interpreted**: The program is passed as source code with all the programming language visible. It is then converted into machine code as it is being used.

- **Server side**: the program is run on the server and only the results are passed to the computer that the browser is on. (JAVA, PHP, Perl, ASP, JSP etc.)
- **Compiled**: languages are converted into machine code first then passed around, so you never get to see the original programming language.

Both JavaScript and Java are:

- **High level**: Written in words that are as close to English as possible. The contrast would be with assembly code, where each command can be directly translated into machine code.

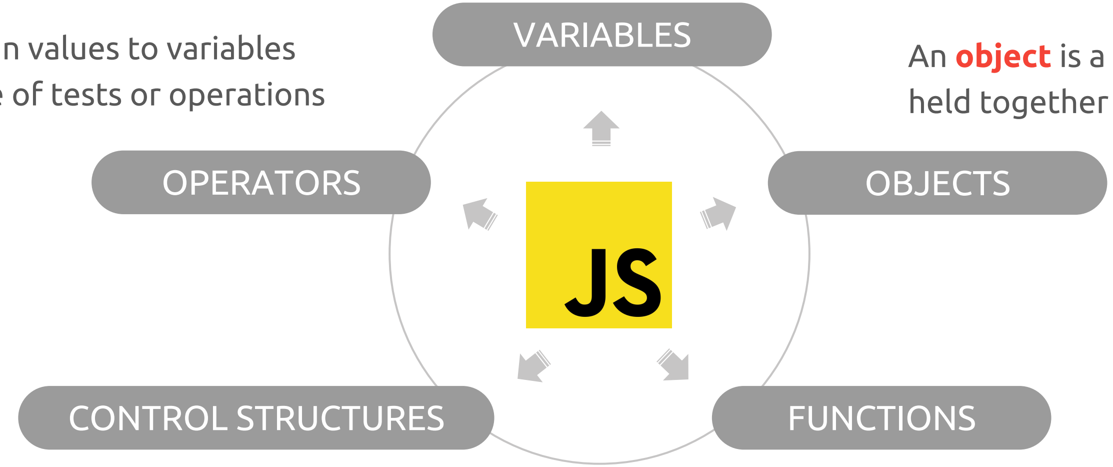
© 2018 Ubiqum Code Academy.

How is JavaScript constructed? **Basic elements**

A **variable** is a word that represents a piece of text, a number, a boolean true or false, a value or an object.

Operators assign values to variables or say what type of tests or operations to perform.

An **object** is a collection of variables held together by a parent variable.



Control structures say what scripts should be run if a test is satisfied.

For now, let's see **functions** as "blocks of code". Functions collect control structures, actions and assignments together and can be told to run those pieces of script as and when necessary.

ubiqum code academy

© 2018 Ubiqum Code Academy.

Variables A **variable** is a **“named storage” for data**.

JS

To create a variable in JavaScript, we need to use the **var** keyword.

The **statement** on the right creates a variable with the name “message”:

```
var message;
```

Now we can put some data into it by using the ***assignment operator*** **=**

```
message = 'Hello'; // store the string
```

The string is now saved into the memory area associated with the variable. We can access it using the variable name:

```
console.log(message); // shows the variable content
```

To be concise we can merge the variable declaration and assignment into a single line:

```
var message = 'Hello!'; // define the variable and assign the value
```

The **console.log()** method writes a message to the browser’s console. The console is useful for testing purposes.
Tip: When testing this method, be sure to have the console view visible (press F12 to view the console).

ubiqum code academy

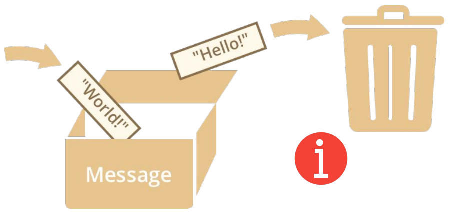
© 2018 Ubiqum Code Academy.

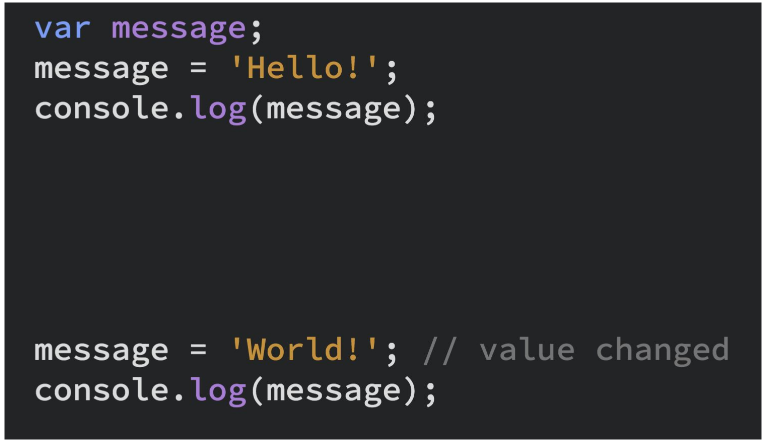
Variables: **a real-life analogy**

We can easily grasp the concept of a “***variable***” if we imagine it as a “box” for data, with a uniquely-named sticker on it. For instance, the variable message can be imagined as a box labeled "message" with the value "Hello!" in it:

We can put any **value** into the box. Also we can change it. **The value can be changed as many times as needed.**

"Hello!"
Message





```
var message;
message = 'Hello!';
console.log(message);



message = 'World!'; // value changed
console.log(message);
```

When the value is changed, the old data is **removed** from the variable.

Variable **naming**

There are two limitations for a variable name in JavaScript:

- The name must contain only letters, digits, symbols $ and _.
- The first character must not be a digit.

Valid names, for instance:

```
var userName;
var test123;
```

When the name contains multiple words, ***camelCase*** is commonly used. That is: words go one after another, each word starts with a capital letter:

```
var myVeryLongName; // camelCase example
```

Examples of incorrect variable names:

```
var 1a; // cannot start with a digit
var my-name; // a hyphen '-' is not allowed in the name
```

Case matters:
Variables named apple and Apple – are two different variables:

```
var apple;
var Apple;
```

Reserved names (keywords):
There is a list of reserved words, which cannot be used as variable names, because they are used by the language itself.
For example, words **var, let, class, return, function** are reserved.

© 2018 Ubiqum Code Academy.

Basic data types in JavaScript (I)

JS

A number

The number type serves both for integer and floating point numbers.

(There are many operations for numbers, e.g. multiplication *, division /, addition +, subtraction - and so on. We'll see that).

```
var number = 123;

number = 12.345; // value changed
```

A string

A string in JavaScript must be quoted.

Double and single quotes are “simple” quotes. There's no difference between them in JavaScript.

```
var string = "Hello";

var string2 = 'Single quotes are ok too';
```

© 2018 Ubiqum Code Academy.

Basic data types in JavaScript (II)

A boolean

The boolean type has only two values: **true** and **false**.
This type is commonly used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.

```
var nameFieldChecked = true; // yes, name
field is checked

var ageFieldChecked = false; // no, age
field is not checked
```

© 2018 Ubiqum Code Academy.

Basic data types in JavaScript (III)

An object

The object type is special. All other previous types are called **“primitive”**, because their values can contain only a single thing (be it a string or a number or whatever). **In contrast, objects are used to store collections of data** and more complex entities.

The values are written as **name:value** pairs (name and value separated by a colon).

```
var myDog = {
    name: "Lassie",
    age: 8,
    color: "brown",
    isDangerous: false
};

var myCar = {
    model: "Fiat500",
    maxSpeed: 200
};
```

JavaScript objects are containers for **named values** called ***properties***.
So, an object can contain several ‘**property : property value**’ pairs.
(e.g. ***model : “Fiat500”, maxSpeed : 200***)

Basic data types in JavaScript (IV)

An array

JavaScript arrays are used to **store multiple values in a single variable**.

An array is a special variable, which can hold more than one value at a time.

An array can hold many values under a single name, and you can **access the values by referring to an index number**.

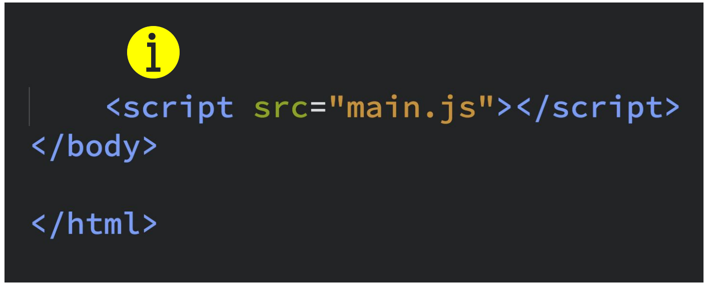
```
var fruits = ["Banana", "Apple", "Melon",
"Lemon"];

var favouriteNumbers = [7, 13, 43, 2, 45, 20,
80, 100];


console.log(fruits[0]); // will print "Banana"

console.log(favouriteNumbers[2]); // will print
43
```

You refer to an array element by referring to the **index number**.
[0] is the first element in an array, [1] is the second, ...
Array indexes start with **0**.

Adding JavaScript to a web page

You can add a script anywhere inside the head or body sections of your document. However, to **keep your project well structured there are some basic guidelines:**

- To insert JavaScript into a web page, use the **<script>** tag.
- **Use external script files:** We suggest to use a separate file to contain the JavaScript (for instance, **main.js**) making it easy to share across pages:
- **Load your javascript code just before the </body> closing tag** in your HTML file. This is called **bottom-loading**.



```
    <script src="main.js"></script>
</body>

</html>
```

Placing scripts in external files has some advantages:

➔ It separates HTML and code.
➔ It makes HTML and JavaScript easier to read and maintain.
➔ Cached JavaScript files can speed up page loads.

Let's code! Exercise 1

Let's start with our first lines of JS code!

- Create an empty HTML page (**index.html**).
- Create your Javascript file (**main.js**) in the same folder of your index.html file.
- Put a **<script>** tag in index.html to load your Javascript code.

In your main.js file:

- Declare two **variables**: admin and name.
- Assign the **value** "John" to name.
- **Copy** the value from name to admin.
- **Show** the value of admin using console.log (**must output "John"**).

© 2018 Ubiqum Code Academy.

JavaScript (basic) **Operators** (I)

JS

ASSIGNMENT OR 'ACTION' OPERATORS

are used to **assign values** to JavaScript variables.

```
var x = 10;

x += 5; // x is now 15

var name = "John";
```

ARITHMETIC OPERATORS

are used to **perform arithmetic** operations between variables and/or values (add, subtract, divide, multiply, for example).

```
var sum = x + 3;

sum = sum / 2;

sum = sum * 3;

var surname = "Doe";

var fullName = name + " " + surname;
```

The **+** operator can also be used to **add (concatenate) strings.**

ubiqum code academy

JavaScript (basic) **Operators** (II)

JS

COMPARISON OPERATORS

are used to **determine equality or difference** between variables or values. **These operators returns true/false values**.

```
(3 < 5) // true

(3 > 10) // false

var number1 = 1;

var number2 = 10;

(number1 < number2) // true

(number1 == number2) // false
```

LOGICAL OPERATORS

are used to **combine multiple boolean expressions** and **provide a single boolean output (true/false).**

```
(3 < 5) && (number1 < number2) // true

(3 > 10) || (number1 == 1) // true

(number > 20 ) && (number2 > 20) // false

!(number1 == 5) // true
```

&& = AND, **||** = OR, **!** = NOT

ubiqum code academy

© 2018 Ubiqum Code Academy.

JavaScript **Operators / Use** Table

JS

Operator	Uses
+	adds two numbers or appends two strings - if more than one type of variable is appended, including a string appended to a number or vice-versa, the result will be a string
-	subtracts the second number from the first
/	divides the first number by the second
*	multiplies two numbers
%	divide the first number by the second and return the remainder
=	assigns the value on the right to the object on the left
+=	the object on the left = the object on the left + the value on the right - this also works when appending strings
-=	the object on the left = the object on the left - the value on the right
>	number on the left must be greater than the number on the right - this also works with strings and values
<	number on the left must be less than the number on the right - this also works with strings and values
>=	number on the left must be greater than or equal to the number on the right - this also works with strings and values
<=	number on the left must be less than or equal to the number on the right - this also works with strings and values

Operator	Uses
++	increment the number
--	decrement the number
==	the numbers or objects or values must be equal
!=	the numbers or objects or values must not be equal
!	logical NOT (the statement must not be true)
&&	logical AND (both statements must be true)
||	logical OR (either statement must be true)
===	the numbers or objects or values must be equal, and must be the same variable type
!==	the numbers or objects or values must not be equal, or must not be the same variable type

ubiqum code academy

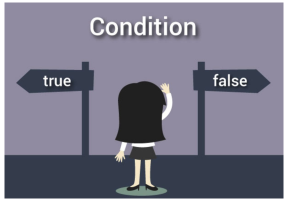
JS Control structures **Making decisions!** (I)

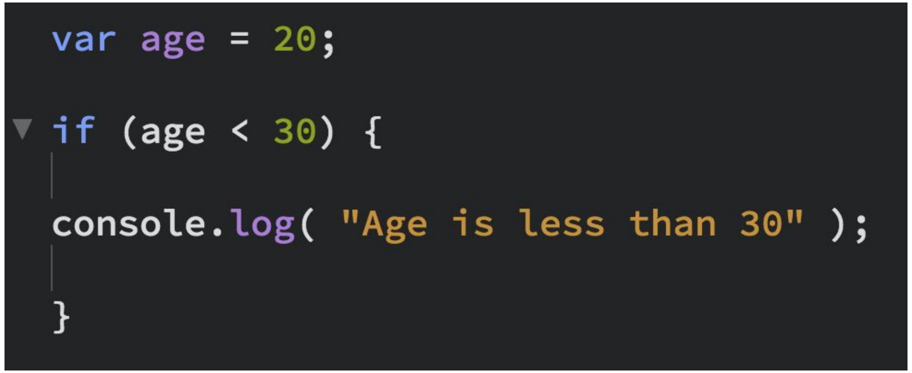
Conditional operators:

if statement

Sometimes we need to perform different actions based on a condition.
There is the ***if statement*** for that.

The if statement gets a condition, evaluates it and, if the result is true, executes the code.





```
var age = 20;

if (age < 30) {

console.log( "Age is less than 30" );

}
```

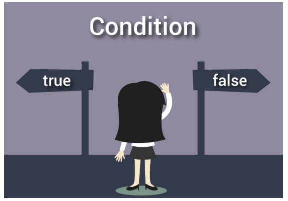
JS Control structures **Making decisions!** (II)

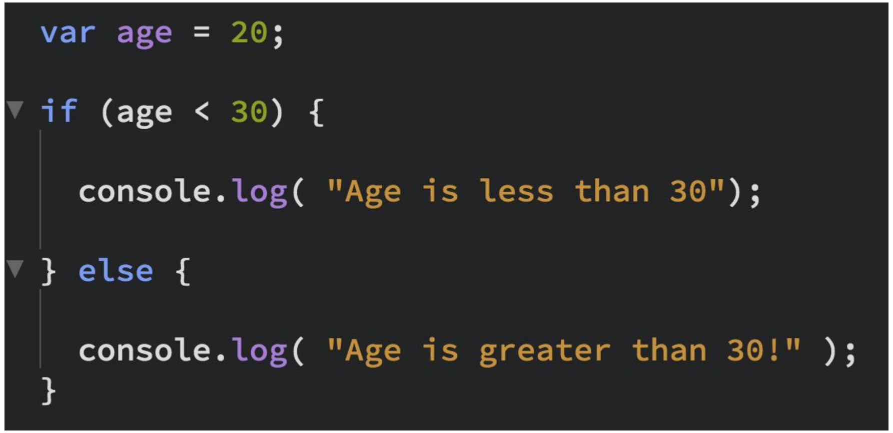
Conditional operators:

the '**else**' clause

The if statement may contain an optional "**else**" block.

It executes when the condition is wrong.





```
var age = 20;

if (age < 30) {

  console.log( "Age is less than 30");

} else {

  console.log( "Age is greater than 30!" );
}
```

Do you need more conditions? Sometimes we'd like to test several variants of a condition. There is an **else if** clause for that. Search for information in Javascript ebook!!

© 2018 Ubiqum Code Academy.

Javascript **Functions** (I): *The core of your code!*

JS

Functions group together script code; control structures, operations, method calls, etc. in the same way as a normal script. These functions can then be **called when needed**, and the **code contained within them will be run.**

This makes it very easy to reuse code without having to repeat it within your script. DRY!

Function declaration:

```
function nameOfFunction(parameters) {

  // function code should be written here

}
```

Function call:

```
nameOfFunction(arguments);
```

Function example: *Sum two given values*

```
var myNum = 10;
var YourNum = 5;

//function declaration

function sumNumbers (num1, num2) {
var sum = num1 + num2;
console.log(sum);
}

// function calls
sumNumbers(myNum, YourNum);
sumNumbers(3, 8);
```

What would happen if we pass two 'strings' to the function?

© 2018 Ubiqum Code Academy.

JavaScript **Loops** (I)

JS

'for' loop

We often have a need to *perform similar actions many times in a row.*

For example, when we need to output goods from a list one after another. Or just run the same code for each number from 1 to 10.

Loops are a way to repeat the same part of code multiple times.

The **for** loop is the most often used one.

```
for (initialization; condition; step) {

  // ... loop body ...   i

}
```

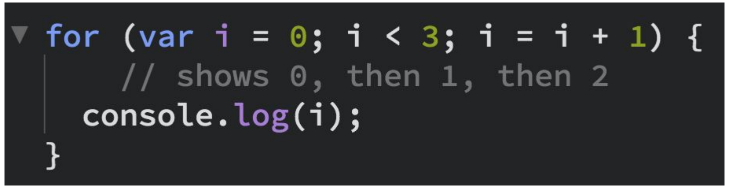
A single execution of the loop body is called an **iteration**.

© 2018 Ubiqum Code Academy.

JS Loops: **'for' loop explained**

Parts of a '**for**' loop: ***initialization, condition, step, body.***

Let's learn the meaning of these parts by example. The loop on the right runs **console.log(i)** for **i** from 0 up to (but not including) 3:



```
for (var i = 0; i < 3; i = i + 1) {
    // shows 0, then 1, then 2
  console.log(i);
}
```

Let's examine the for statement part by part:

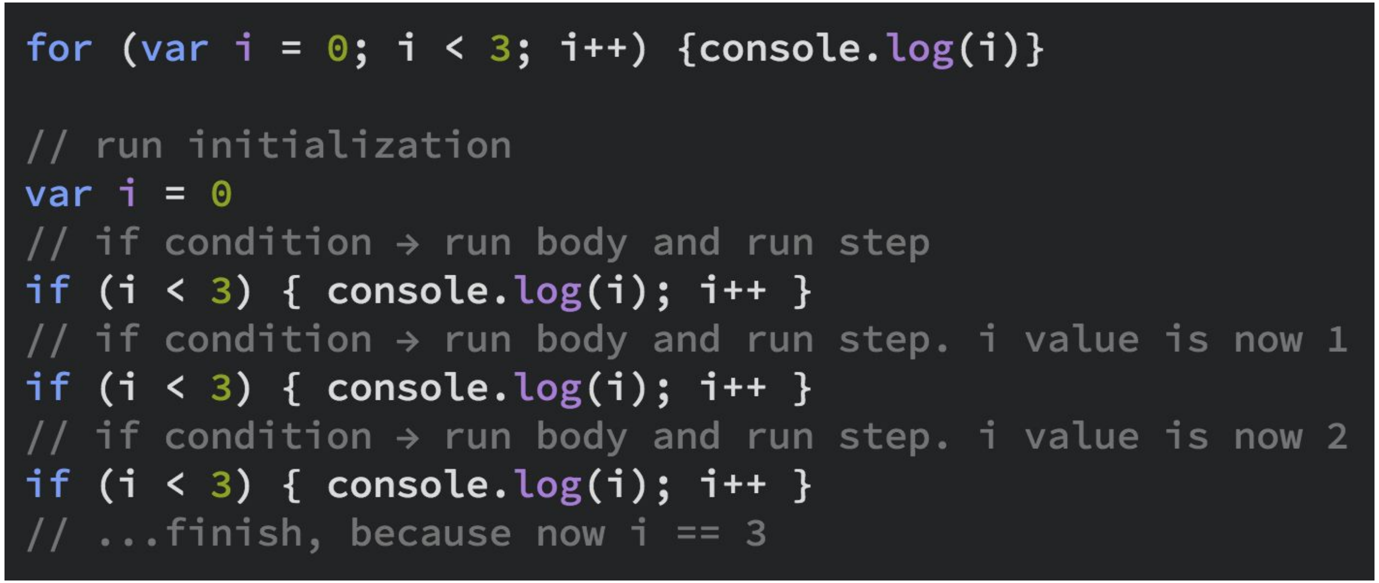
part	code	action
initialization	var i = 0	**Executes once** upon entering the loop.
condition	i < 3	Checked before every loop iteration, **if fails the loop stops**.
step	i = i + 1 (or i++)	Executes after the body on each iteration, but before the condition check.
body	console.log(i)	Runs again and again while the condition is truthy

© 2018 Ubiqum Code Academy.

JS Loops: **‘for’ loop explained** (EXAMPLE)

If you are new to loops, then maybe it would help if you go back to the example and reproduce **how it runs step-by-step** on a piece of paper.

Here’s what exactly happens in our case (step-by-step) :



```
for (var i = 0; i < 3; i++) {console.log(i)}

// run initialization
var i = 0
// if condition → run body and run step
if (i < 3) { console.log(i); i++ }
// if condition → run body and run step. i value is now 1
if (i < 3) { console.log(i); i++ }
// if condition → run body and run step. i value is now 2
if (i < 3) { console.log(i); i++ }
// ...finish, because now i == 3
```

© 2018 Ubiqum Code Academy.

JavaScript **Loops** (II)

JS

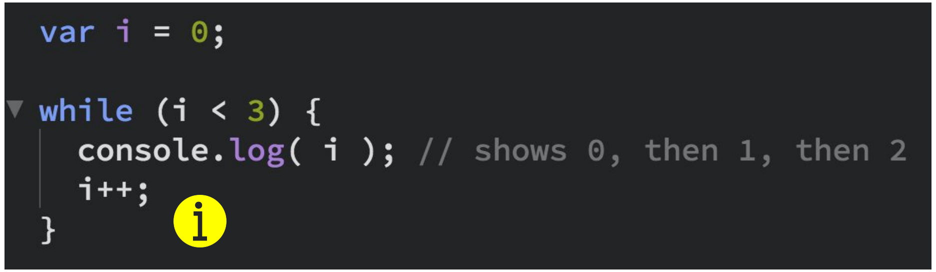
‘while’ loop

The **while** loop has the following syntax:

```
while (condition) {
  // code
  // so-called "loop body"
}
```

Let’s reproduce the ‘*while*’ version of the previous example that used ‘for’ loop:

While the condition is true, the code from the loop body is executed.
For instance, the loop on the right outputs **i** while **i < 3**:



```
var i = 0;

while (i < 3) {
   console.log( i ); // shows 0, then 1, then 2
   i++;
}
```

What would happen if we do not put **i++** within the loop?

JS Functions (II): parameters/arguments

JS

We **declare** two **arrays** like these…

```
var animals = ["Dog", "Cat", "Pig", "Bird"];

var developers = ["Jack", "Alex", "Mary"];
```

a **Function WITH NO** parameters…

```
function showAnimals () {

    for (var i = 0; i < animals.length; i = i + 1) {

        console.log(animals[i]);
    }
}
```

and finally a **Function WITH** parameters:

```
function showElementsInArray (array) {

    for (var i = 0; i < array.length; i = i + 1) {

        console.log(array[i]);
    }
}
```

Now, we **call** functions…

```
showAnimals();

// function calls passing ARGUMENTS

showElementsInArray(animals);

showElementsInArray(developers);

showElementsInArray([1,4,8]);

showElementsInArray(["Paul","Mike"]);
```

QUESTIONS:

- What **results** we expect for each function call?
- What is the main **difference** between the two functions? *Which one is most, let's say, 'useful'?*
- What about the last two statements?

ubiqum code academy

Parameters are variables listed as a part of the function definition.
Arguments are values passed to the function when it is invoked.

Thanks for Watching

Time to start coding in JavaScript!!