Intro

The Client-side Programming Language

What's with the name?

JavaScript != Java

Today's Goals

- Familiarise with the **Syntax**
- Practise the **basics** in JavaScript

Today we'll work in the **browser**

JavaScript Version

We are going to use **ES6**:

- Short for ECMAScript Edition 6
- Released in 2015 (ECMAScript 2015 Language)
- Supported by ~90% browsers

Run code on your browser

```
// in-browser dev tools
console.log("Hello Le Wagon");
➜
Hello Le Wagon
```

> Browser console is a REPL for JavaScript

Basic Types

```
"Hello Le Wagon"              // string

42                            // number
3.14                          // number

true                          // boolean
```

Checking types

```
typeof("Boris");
// => 'string'

typeof(42);
// => 'number'
```

Casting types

```
Number.parseInt('42', 10);
// => 42

(42).toString();
// => '42'
```

Data structures

```
[ 'Hello', 'Le', 'Wagon', 42 ]    // Array

{ name: 'bob', age: 42 }          // Object
{ 'name': 'bob', 'age': 42 }      // Object (the exact same)
```

Null & Undefined

```
let age; // undefined
let name = null;
```

Variables

Old JS uses `var`.

ES6 uses two new keywords in replacement.

`let`

For a variable you **will re-assign**

```
let counter = 1;
console.log(counter);

counter = counter + 1;
console.log(counter);
```

`const`

For a variable you **won't** re-assign

```
const firstName = "John";
console.log(firstName);

firstName = "Paul"; // TypeError: Assignment to constant variable.
```

Naming convention

```
const firstName = "Ringo";
// lowerCamelCase
```

Strings

Let's dive deeper into this type.

Reference: String on MDN web docs

Length

```
const firstName = "Paul";
firstName.length;
// => 4
```

Character access

```
const firstName = "John";
firstName[0];
// => "J"

// Print all characters starting at index 1
firstName.substring(1);
```

Case manipulation

```
const firstName = "Paul";
firstName.toUpperCase();
// => "PAUL"

firstName.toLowerCase();
// => "paul"
```

Split

```
const monthString = "Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec";

const months = monthString.split(",");
// => [ 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' ]
months.length;
// => 12
```

Interpolation

```
const firstName = "Ringo";
const lastName = "Starr";

const message = `${firstName} ${lastName} is a drummer`;
// => "Ringo Starr is a drummer";
```

Template literals on MDN

Arrays

Reference: Array on MDN web docs

CRUD

```
const fruits = [];
fruits.push("Apple"); // Create
fruits[0];            // Read
fruits[0] = "Banana"; // Update
fruits.splice(0, 1);  // Delete (1 item at index 0)
```

forEach

```
const beatles = ["paul", "john", "ringo", "george"];
beatles.forEach((beatle) => {
  console.log(beatle.toUpperCase());
});
```

Array.forEach

Control Flow

`if` / `else`

```
const age = 14;

if (age >= 18) {
  console.log("You can vote");
} else {
  console.log("You can't vote");
}
```

Falsy values

```
false
undefined
null
0
NaN
""
```

Ternary Operator

```
const raining = true;
const accessory = (raining ? "umbrella" : "sunglasses");
// => "umbrella"
```

```
if (digit === 0) {
  console.log('Zero');
} else if (digit === 1) {
  console.log('One');
} else {
  console.log("I don't know this digit, sorry!");
}
```

Read more about sameness in JS and the difference between `==` and `===`.

Objects

Guide: Working with Objects on MDN

Simple Object

```
const student = {
  firstName: "Boris",
  lastName: "Paillard"
};

console.log(typeof student);
// => "object"

console.log(student);
```

Reading/Setting a property

You can use dot-notation.

```
console.log(student.firstName);
// => "Boris"
console.log(student['firstName']); // Another way
// => "Boris"
student.firstName = "Romain";
console.log(student.firstName);
// => "Romain"
```

Functions

Read the Function Guide on MDN web docs

Defining

JavaScript (**old way**)

```
function square(x) {
  return x * x;
}
```

Note the **explicit** `return`

Calling

```
square(10);
// => 100
```

Arrow Function

```
const square = (x) => {
  return x * x;
};

// Or even shorter, with **implicit** return.
const square = x => x * x;
// Calling the function: same as before
square(10);
```

What should I use?

Arrow functions are a new way to write functions since ES6 and they are our preferred way of writing functions. When working with web JavaScript, always use arrow functions instead of ES5 `function` statements.

Capitalize example

Let's livecode an arrow function and store it into `capitalize`.

```
touch lib/capitalize.js
```

```
const capitalize = (word) => {
  const firstLetter = word[0].toUpperCase();
  const restOfTheWord = word.substring(1).toLowerCase();
  return `${firstLetter}${restOfTheWord}`;
};
```

Note the `return` on the last line

Inside multi-line functions, you must add the `return` keyword before the value you want to function to **output**.

```
const capitalize = (word) => {
  const firstLetter = word[0].toUpperCase();
  const restOfTheWord = word.substring(1).toLowerCase();
  firstLetter + restOfTheWord
};
```

```
// calling and loggin the function result
console.log(capitalize("hacker")) // => undefined
```

👍

```
const capitalize = (word) => {
  const firstLetter = word[0].toUpperCase();
  const restOfTheWord = word.substring(1).toLowerCase();
  return firstLetter + restOfTheWord
};
```

```
// calling and loggin the function result
console.log(capitalize("hacker")) // => Hacker
```

To sum up 🎯

JavaScript has built in `types` to help us work with different values

- `string` for any text - sentences, emails, even phone numbers!
- `number` for any numericals - whole numbers, numbers with decimals
- `array` (also `object` in JS) - for indexed lists of *any* values
- `object` - for collections of values *with labels*

We use `variables` to store values:

```
// 'let' variables can be changed
let name = "Alexander"
name = "Alex"

// 'const' variables are constant
const water = "H2O"
water = "CO2" // error: Assignment to constant variable.
```

We use `functions` to store blocks of code that perform an action, to avoid repetition

```
// ES5
function isEven(num) {
  return num % 2 === 0
}

// ES6
// for single-line functions
const isEven = (num) => num % 2 === 0

// for multi-line functions
const isEven = (num) => {
  return num % 2 === 0
}
```

After **defining/declaring** a function, we can **call it** (run it)

```
isEven(2) // function is called, but the result is not displayed
console.log(isEven(3)) // function is called and the result is displayed
```

Happy (Back-end) JavaScripting!