👌🏽

JS Recap

> Finally, a programming language!

JavaScript is a programming language that works with HTML when it's attached to it.

In this recap I will go over the most important blocks in writing JS code (I will not talk about programming languages, interpretation vs compilation etc)

Starting with JS

- Create a separate file with the file extension **.js**
- Insert in the <head> tag of an HTML document with

```
<script src="file.js"></script>
```

JS Output

> console.log() will write in your console which is accessible using F12 in the Console tab

Comments

Need a quick comment? In VS Code hold **CTRL** and tap **/**

Variables

Think of variables as a portion of memory with a unique name and a value.

Some variables can be uninitialized and without a value at first.

To declare a variable in JS there are 3 ways (types of variables)

- **var** keyword → older version of JS which defines variables
- new version
  - **let** → this will define a variable that can change it's value
  - **const** → this keyword will define a variable that should have a constant value remaining.

```
var code;
let code_2;
```

```
const code_3;
```

A few points → **var**, **const** and **let** are keywords that define variables. **code**, **code_2**, **code_3** are variable names.

Variable should have unique names, cannot start with a number and cannot use some restricted words.

Mathematical operations

In JavaScript there are many mathematical operations, most important:

- Sum +
- Difference -
- Multiplication *
- Division /
- Modulo % (division reminder)

Shorthand operators

These mathematical operators also have a equivalent shorthand for example:

- **a = a + b** → shorthand is **a += b**
- **a = a * b →** shorthand is **a *= b;**

and others. check this article → it shows all of the possible shorthands which can make you write less code!

25+ JavaScript Shorthand Coding Techniques - SitePoint
This really is a must read for any JavaScript developer. I have written this guide to shorthand JavaScript coding techniques that I have picked up over the years. To help you understand what is going on, I have included the
https://www.sitepoint.com/shorthand-javascript-techniques/

Other mathematical operations

- Incrementation → when we want to increase a value by 1

  ```
  a++; // this is a post increment (it will add one after a main action)
  ++a; // it will put the incrementation as the first action before doing anything else
  ```

- Decrementation → when we want to decrease a value by 1

  ```
  a--;
  ```

Types

Variables have their names but also their underlying type. Think of a types as a piece of information indicating the browser what that variable is (maybe it's a string, maybe it's a number etc).

In **JavaScript we do not need to declare types** but they are assigned **automatically**. This is both useful but also has its disadvantages. To debug types we can use command:

```
console.log(typeof something); // this will print in the console the type of variable named something
```

Main Types

- Boolean → true or false → logical type that only has 2 states.
- Number → In JS a number is an integer (2, 4, 5, 6) and or float (11.22, 10.5) → Represents a numeric value
- String → values that are specified using brackets → used to represent text variable values
  - ''
  - ""
  - ``
- Array → represents a list of values in one variable
- Object → a variable that has a set of variables inside of it, including functions.
- null → there is no value (but there is a variable like this)
- undefined → when a variable has not been declared or initialized
- NaN → Not A Number → invalid number (for example when trying to parse a string into a number type)

Comparison Operators

Used to compare values usually in places where a program must perform some operations only if certain conditions are met. Main comparisons are:

- a < b
- a > b
- a ≥ b
- a ≤ b
- ! → NOT or opposite value
- a === b → compares variables and their types (three equal sign is strong comparison)
- a == b → compares only variable values (but not the types) (two equal signs is weak comparison)
- a ≠= b → if variable value and type is not equal
- a ≠ b → if variable value is not equal

IF Statements

Think of an IF statement as an instruction for the program to do something only if a certain requirement is met.

> Inside the IF statement there needs to be a variable or a comparison which results in a Boolean value → true or false

There are a few types of if statements:

- Regular if statement

```
if (a > b) {
  // do something
}

// we can also write this as
const comparison = a > b;
if (comparison) {
  // do something
}
```

- if … else

```
if (a < b) {
  // first case
} else {
  // everything else or if a < b  is not true
}
```

- if else if

```
if (a < b) {
  // first case
} else if (a > b) {
  // second case
} else {
  // everything else
}
```

- Ternary operator → this is a shorthand operator for an if else statement.

```
let difference;
if (a === b) {
  difference= 0;
} else {
  difference= a - b;
}
// it can be written as
difference = (a === b) ? 0 : a - b;
```

- Switch → When a IF statement has more than 3 options we use switch for readability an efficiency (we can write many **else if** statements but with a switch it's easier and more readable). Example:

```
switch(a) {
  case 1:
    // if a === 1
    // do something
    break;
  case 2:
    // if a === 2
    // do something
    break;
  default
    // do something when the cases are not met
    break;
}
```

Remember to use a **break;** in a switch and pass a variable like **a** here on which a different case will be executed.

Functions

Functions are a name and limited fragment of code that can be created to perform a specific task and reuse. They are similar to mathematical functions where they get arguments and return a value. Although in programming functions **can** receive no parameters and return no value. A few types of functions:

- Functions without any parameters

```
function myFunction() {
  console.log('My first function!');
}
```

- Functions with parameters

```
function myFunction(a, b) {
  console.log('My first function with parameters ' + a + ' and ' + b);
}
```

- Functions without any return value (the one's above)
- Functions with return value

```
function sum(a, b) {
  return a + b;
}
```

- All of the upper combinations 🙂 (with parameters and return, without parameters and a return and so on...)

Things to remember:

- function keyword
- { } → inside the function block
- naming functions is similar to naming variables (can't start with a number, some keywords are prohibited, must be unique and so on)
- once you **define** a function you have to **call** it to use it

---

Loops

Loop is a way to do things many times over and over again until a specified end.

There are many ways to loop in JS, firstly I'll show the vanilla JS ways.

For

```
for (var i=0; i<100; i++) {
  // do something
  // this will execute 100 times
}
```

While

```
var i = 0;
while (i < 100) {
  i++;
}
// this will also execute a 100 times
```

Do While loop (it guarantees that it will run AT LEAST ONE TIME)

```
var i = 0;
do {
  //something
} while (i < 100);
// this will run a 100 times
```

Additional keywords for loops

Break

This keyword will exit out of a loop. We can use it once a certain criteria is met to stop the loop (maybe if we were searching for something in an array and found it)

```
for (var i = 0; i < 100; i++) {
  if (i === 4) {
    console.log('found it!');
    break;
  }
}
```

Continue

Continue allows us to skip the current iteration of a loop without skipping it. We can use it in cases where we don't need to perform the computations inside the loop for this iteration and we can just skip to the next one.

```
for (var i = 0; i < 100; i++) {
  if (i !=== 4) {
    continue;
  }
  // some code if we don't skip...
}
```

Arrays

Arrays is a variable that can store many values (think of it as a list). They can be **referred by their indexes which start from 0.**

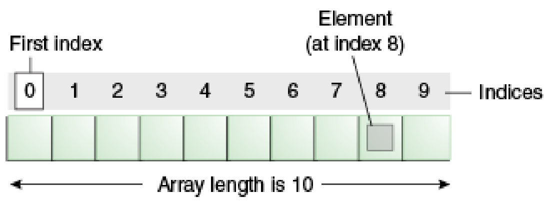
Important:

- Arrays have indexes starting from 0 and incrementing by one (0, 1, 2, ...)
- Arrays have values in their specified index positions, so the value in an index 0 of array is in array[0].
- Arrays can story all types of variables (Booleans, objects, strings, numbers even undefined, null etc)

Example array initialization

```
var a = [1, 5, 16, 7]
```

> A little bit more about this array and indexes + values.



In the picture above we can see that indexes are always starting from 0. In the array we have written in the code block above we can see that **the value of element at index 2 is 16.** This seeing of arrays is important.

Accessing arrays in JS

We access an array in JS with **variable[index]** syntax, for example:

```
var a = [1, 15, 6];
console.log(a[2]); // this will log 6
a[1] = 4; // this will change element at index 1 value to 4 (from 15)
```

Multidimensional arrays

As stated in a chapter before arrays can hold different types and so they can also hold arrays. So in array element at index **x** we can have another array of size **m**. And so on, for example:

```
var a = [1, 2, 3, [ 'a', 'b' ] ]
// here we have an array ['a','b'] inside of array a
// to access it we could write
console.log(a[3][0]); // this would print 'a'
```

Iterating over arrays

A really common task in programming is to iterate over an array of any type of items and do some computation. There are many ways to do it some better than others. In my personal experience I prefer using **.map** or **.foreach** built-in functions for iterating. In some cases where I need to sort or find some element I use built-in functions **.sort** or **.find** but knowing how to do it with basic loops is also important. So some ways are:

- Standard for loop

```
for(var i = 0; i < myArray.length; i++){
  var currentValue = myArray[i]
  // do something
  // we access elements with myArray[i]
}
```

- Built-in for each function

```
myArray.forEach(function(val,key){
  var currentValue = val;
  // do something
});
```

- For .. of → allows quick access of element values → iterates over values

```
var tab = [10, 20, 50];

for(var i of tab){

    //outputs a value of an element
    console.log(i);
}
//this loop will output: 10, 20, 50
```

- For .. in → allows quick access of element indexes → iterates over indexes

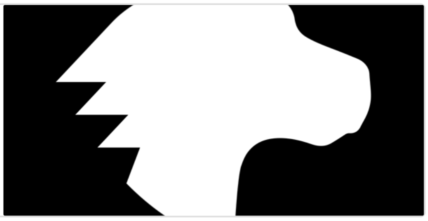
```
for(var i in tab){

    //outputs an index 0, 1, 2, 3 ....
    console.log(i);

    //outputs a value
    console.log(tab[i]);
}
//this loop will output: 0, 10, 1, 20, 2, 50
```

There are so many more functions to use for iterating and manipulating arrays. All of them are written in this holy reference:

Array

The JavaScript Array class is a global object that is used in the construction of arrays; which are high-level, list-like objects. Arrays are list-like objects whose prototype has methods to perform traversal and mutation operations.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array



JS Array reference

> If we wanted to find Index of something we could use **indexOf**, if we wanted to know if an array has some value we could use includes and many more, learn to search and use references and documentation to find what best fits your case.

Objects

Last but not least there is objects. Object can be thought of as an instance of some class. An object has different properties inside it (at least in JavaScript).

Think of an object like a car and inside there are properties like brand, model etc. Let's look at an example:

```
var car = {
    brand: "ford",
    model: "fiesta",
    year: 1999,
    mileage: 200000,
    launch: function() {
        console.log("works!");
    }
}
```

Important notes:

- Objects can hold almost all other types of variables:
    - Strings
    - Numbers

- Booleans
- undefined
- null
- functions
- arrays
- Objects

Accessing Objects in JS

Let's look at an example: (to access a key in JS we use a **.**)

```
console.log(car.brand); // this would log ford in our case
// we can also change object properties in JS
car.brand = "volkswagen"; // this would change brand from ford to volkswagen
// additionally if there's a function inside we can call it!
car.launch(); // this woould console.log() the string 'works'
```

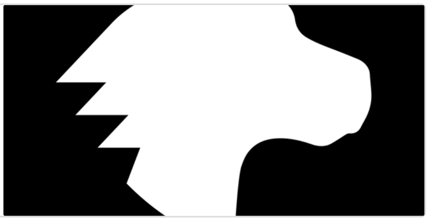
Due to the JS implementation we can change Object properties and Array elements to different values so be aware of that! (Even if you create **const somearray** or **const someObject** you can still change the values inside them!)

For more information about Objects read this holy grail that is MDN web docs:

Working with objects

JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects



Additional remarks

Hoisting

In general this topic is a little bit confusing at first but hoisting is a process when variable and function declarations are physically moved to the top of your code. (Technically, during the compile phase the function and variable declarations are put into memory to be used later).

> Hoisting generally refers to declarations. As a good practice and habit you should always declare your variables or functions before using them

More about Hoisting can be read here:

Hoisting
Hoisting is a term you will not find used in any normative specification prose prior to ECMAScript® 2015 Language Specification. Hoisting was thought up as a general way of thinking about how execution contexts (specifically the
https://developer.mozilla.org/en-US/docs/Glossary/Hoisting

This guide will explain to you what errors you can get if you don't follow standard declarations (for example if you use a variable before it was declared etc.). It's mostly technical and just remember to declare things before using them 😉

Using Strings

I have seen some confusion involving strings. There are several ways to create a string literal:

```
const string1 = "A string primitive";
const string2 = 'Also a string primitive';
const string3 = `Yet another string primitive`;
```

All of these are valid but have different advantages:

```
// if you wanted to add a quation mark in your string you could use something like this
console.log(`"Quote by someone"`);
// combining these types you can create more things
```

Important: Remember that strings have to open or close so this syntax is incorrect

```
console.log(""something"");
```

Why? Because you open and close the string in the first 2 characters (""), then JS interprets **something** as a variable and then adds another string which opens and closes immediately ("").

In different projects, teams and companies you may have different standards of working with strings, I have seen 2 or 3 already and it's something that you learn and get used to quick.

String
The String object is used to represent and manipulate a sequence of characters. Strings are useful for holding data that can be represented in text form. Some of the most-used operations on strings are to check their , to build
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

More about strings

Use Math objects

If you're dealing with complex mathematical operations use the Math object.

Why? Because it ensures more safety then doing it for out of the box. For example if you wanted to find an exponent of a number you could use **Math.pow()** or if you'd want to round a number to the nearest lower value you could use **Math.ceil** or maybe the higher nearest value with **Math.floor** etc..

Doing it with regular JavaScript may lead to different unexpected errors, Math object ensures the safety of mathematical operations.

Note: You don't need to use this Object if you're doing simple multiplication, sum or something similar, but if you need to find a square root, a logarithm, abstract, even find cosine or create random numbers use the **Math** object.

Math
Math is a built-in object that has properties and methods for mathematical constants and functions. It's not a function object. Unlike many other global objects, Math is not a constructor. All properties and methods of Math are
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

More about this beautiful object

That's it!

Now go on and practice, practice, practice. Be sure to understand the basics very well and practice them with real code (not only theory).

Don't be scared to:

- Google basic things to understand them more clearly
- Search documentation
- Search others people's code (but don't copy, you won't learn anything)

Tips:

- Be patient, sometimes visualizing concepts and understanding them takes time and practice and it may be frustrating at first but you'll get there 😉
- Start a habit of learning to read references and guides and even forum answers (like StackOverflow) where the code is written, this will be useful in the future when learning different technologies
- Ask questions to other people in forums, Q&A platforms and even your colleagues, friends or acquaintances who could help
- Don't try to memorize things, just try to practice and it'll stay in your memory after some time
- Do a project for yourself! Maybe it's a e-shop, a clone of some other site, doing a real project is where you'll learn the most, trust me!

Some places to practice things for JS:

HackerRank

HackerRank is the market-leading technical assessment and remote interview solution for hiring developers. Learn how to hire technical talent from anywhere!

https://www.hackerrank.com/

Coding Tests and Assessments for Technical Hiring | CodeSignal

Many organizations today have implemented some form of technical skill assessment into their hiring process. This might look like a take-home test, a whiteboard assignment, or an engineering team crafting their own skill

https://codesignal.com/

Codewars: Achieve mastery through challenge

Codewars is where developers achieve code mastery through challenge. Train on kata in the dojo and reach your highest potential.

https://www.codewars.com/

I advise to use **CodeSignal** but getting used to these arcade type apps will take some time (you have to start thinking in a different way).

Reference

This is the best reference that there is out there. You can check things here to find some examples, explanations of some topics or built-in functions and many more.

JavaScript reference

This part of the JavaScript section on MDN serves as a repository of facts about the JavaScript language. Read more about this reference. JavaScript standard built-in objects, along with their methods and properties. JavaScript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference