# JavaScript

* JavaScript is a high level, object oriented, multi paradigm programming language.
* High-level: Resources (like memory) are managed by the language itself, resources are not managed manually as in low level languages, as C.
* Garbage-collected: Algorithm inside the JavaScript engine which automatically removes all the unused objects from the computer memory.
* Interpreted or just-in-time compiled: PENDING
* Multi-paradigm: Supports procedural programming, object-oriented programming and functional programming.
* Prototype-based object-oriented: Almost everything in JavaScript are objects, even arrays are objects and those have access to its methods (push, shift, etc.) due to the prototype inheritance.
* First class functions: Functions are treated as variables, we can pass them into other functions as parameters or return them as the output of a function.
* Dynamic: Dynamically typed language. No data definitions, types become known at runtime. Data type of variables are automatically changed on reassignment.
* Single-threaded: JavaScript runs in one single thread, so it can only do one thing at a time.
* Non-blocking event loop: Takes long running tasks, executes them in the “background” and puts them back in the main thread once they are finished.
* Variable names can only contain numbers, letters, underscores, or the dollar sign.
* As a convention, write constant variable names with uppercase, like const PI = 3.14
* == operator (loose) can introduce bugs to the code, it is best to ALWAYS use the === operator (strict) and if type coercion is needed, then do it manually instead of expecting the == operator to do it automatically.
* Expression: produces a value (1 + 2) or (2 < 3)
* Statement: Full sentence (const a = ‘sale’;)
* Ternary operator (? :) is an expression, different to an if else statement, which is a statement.
* JavaScript updates will never break old revisions, NEVER BREAK THE WEB, all updates are backwards compatible.
* Strict mode: Type ‘use strict’; in the first line of the script, helps to prevent issues that can appear if no strict mode is activated, recommended to always use it.
* Functions inside an object are called methods and methods inside an object have access to the ‘this’ keyword, ONLY METHODS. ‘this’ is writable and readable, FYI.
* Arrow functions do not have access to its own ‘this’ keyword, only normal functions ‘function() {}’.
* Arrays are also objects, that is why they have their own methods.
* ‘continue’ advances to the next iteration and ‘break’ gets you out of the loop.
* Besides from ‘console.log’, we also have ‘console.warn’ and ‘console.error’.
* ‘debugger;’ can be used to open the debugger tool in the browser, being that line the breakpoint.
* The CSS ‘box-sizing’ property allows us to include the padding and border in an element's total width and height.
* Document Object Model ‘DOM’: Structured representation of html documents. Allows JavaScript to access html elements and styles to manipulate them.
* Top level code is the code that is NOT inside a function and will be executed in the global execution context.
* One execution context is created per function, and those contexts together with the global execution context make the ‘call stack’.
* Inside an execution context you can find
* Variable environment: let, const and var declarations | Functions | arguments object
* Scope chain: References to variables that are located outside the current function.
* ‘this’ keyword: Special variable PENDING
* The JavaScript engine is formed by the call stack and the memory heap.
* Block scope applies only for variables declared with let or const. Variables declared with var are function scope (or global). Functions are also block scoped (only in strict mode).
* Hoisting: Makes some types of variables accessible/usable in the code before they are actually declared. “Variables lifted to the top of their scope”. Before execution, code is scanned for variable declarations, and for each variable, a new property is created in the variable environment object.
* ‘this’ keyword/variable: Special variable that is created for every execution context (every function). Takes the value of (points to) the “owner” of the function in which the ‘this’ keyword is used.
* ‘this’ is NOT static. It depends on how the function is called, and its value is only assigned when the function is actually called. ‘this’ does NOT point to the function itself and also NOT its variable environment.
* Method: this = <Object that is calling the method>
* Simple function call: this = undefined
* Arrow functions: this = <this of surrounding fuction (lexical this)>
* Event listener: this = <DOM element that the handler is attached to>
* new, call, apply, bind: <PENDING>
* Only primitive values are stored directly in the call stack, that is why a variable declared with const cannot be changed. On the other hand, objects and functions are stored in the memory heap and only a reference to that value is stored in the call stack, that is why if you declare a variable to be equal to another object variable, the new variable points to the same reference in the memory heap.
* Deep copies of objects can be made with external libraries.
* The spread operator works on all iterables, NOT in objects.
* Spread operator can only be used when building an array or when we pass values into a function. BUTTT since es2018, spread operator can also be used for objects.
* The spread and the rest syntax both look exactly the same, but they work in opposite ways. The spread operator is used where we would otherwise write values separated by a comma, on the other hand, the rest pattern is used where we would otherwise write variables separated by a comma.
* The Nullish Coalescing operator (??) introduced in es2020 evaluates only the nullish (null, undefined, ) values, different to the || that evaluates falsy values.
* Logical assignment operators were introduced in es2021 (||=) (&&= ) (??=)