**Introduction to JavaScript**

JavaScript is one of the most popular programming languages in the world, and now widely used also outside of the browser. The rise of Node.js in the last few years unlocked backend development, once the domain of Java, Ruby, Python, PHP, and more traditional server-side languages.

Introduction

JavaScript is one of the most popular programming languages in the world. Created 20 years ago, it's gone a very long way since its humble beginnings. Being the first - and the only - scripting language that was supported natively by web browsers, it simply stuck. In the beginnings, it was not nearly powerful as it is today, and it was mainly used for fancy animations and the marvel known at the time as DHTML. With the growing needs that the web platform demands, JavaScript had the responsibility to grow as well, to accommodate the needs of one of the most widely used ecosystems of the world. Many things were introduced in the platform, with browser APIs, but the language grew quite a lot as well. JavaScript is now widely used also outside of the browser. The rise of Node.js in the last few years unlocked backend development, once the domain of Java, Ruby, Python and PHP and more traditional server-side languages. JavaScript is now also the language powering databases and many more applications, and it's even possible to develop embedded applications, mobile apps, TV sets apps and much more. What started as a tiny language inside the browser is now the most popular language in the world.

A basic definition of JavaScript

JavaScript is a programming language that is: high level: it provides abstractions that allow you to ignore the details of the machine where it's running on. It manages memory automatically with a garbage collector, so you can focus on the code instead of managing memory locations, and provides many constructs which allow you to deal with highly powerful variables and objects.

dynamic: opposed to static programming languages, a dynamic language executes at runtime many of the things that a static language does at compile time. This has pros and cons, and it gives us powerful features like dynamic typing, late binding, reflection, functional programming, object runtime alteration, closures and much more.

dynamically typed: a variable does not enforce a type. You can reassign any type to a variable, for example assigning an integer to a variable that holds a string.

weakly typed: as opposed to strong typing, weakly (or loosely) typed languages do not enforce the type of an object, allowing more flexibility but denying us type safety and type checking (something that TypeScript and Flow aim to improve)

interpreted: it's commonly known as an interpreted language, which means that it does not need a compilation stage before a program can run, as opposed to C, Java or Go for example. In practice, browsers do compile JavaScript before executing it, for performance reasons, but this is transparent to you: there is no additional step involved.

multi-paradigm: the language does not enforce any particular programming paradigm, unlike Java for example which forces the use of object oriented programming, or C that forces imperative programming. You can write JavaScript using an object-oriented paradigm, using prototypes and the new (as of ES6) classes syntax. You can write JavaScript in functional programming style, with its first class functions, or even in an imperative style (C-like).

In case you're wondering, JavaScript has nothing to do with Java, it's a poor name choice but we have to live with it.

**Coding style**

This JavaScript Coding Style is the set of conventions I use every day when using JavaScript. It's a live document, with the main set of rules I follow A coding style is an agreement with yourself and your team, to keep consistency on a project. An if you don't have a team, it's an agreement with you, to always keep your code up to your standards. Having fixed rules on your code writing format helps a lot in order to have a more readable and managed code.

**Be consistent with the project you work on**

Even if you prefer a set of styles, when working on a project you should use that project style. An Open Source project on GitHub might follow a set of rules, another project you work on with a team might follow an entirely different one. Prettier is an awesome tool that enforces code formatting, use it.

**My own preferences**

My own take on JavaScript style is:

Always use the latest ES version. Use Babel if old browser support is necessary.

Indentation: use spaces instead of tabs, indent using 2 spaces.

Semicolons: don't use semicolons.

Line length: try to cut lines at 80 chars, if possible.

Inline Comments: use inline comments in your code. Use block comments only to document.

No dead code: Don't leave old code commented, "just in case" it will be useful later. Keep only the code you need now, version control/your notes app is meant for this.

Only comment when useful: Don't add comments that don't help understand what the code is doing. If the code is self-explaining through the use of good variable and function naming, and JSDoc function comments, don't add a comment.

Variable declarations: always declare variables to avoid polluting the global object. Never use var . Default to const , only use let if you reassign the variable. Constants: declare all constants in CAPS. Use \_ to separate words in a VARIABLE\_NAME .

Functions: use arrow functions unless you have a specific reason to use regular functions, like in object methods or constructors, due to how this works. Declare them as const, and use implicit returns if possible.

const test = (a, b) => a + b

const another = a => a + 2

Names: function names, variable names and method names always start with a lowercase letter (unless you identify them as private, read below), and are camelCased. Only constructor functions and class names should start capitalized. If you use a framework that requires specific conventions, change your habits accordingly. File names should all be lowercase, with words separated by - .

**Statement-specific formats and rules:**

If

if (condition) { statements }

if (condition) { statements }

else { statements }

if (condition) { statements }

else if (condition) {

statements }

else { statements }

for

Always initialize the length in the initialization to cache it, don't insert it in the condition. Avoid using for in except with used in conjunction with .hasOwnProperty() . Prefer for of (see JavaScript Loops)

for (initialization; condition; update) { statements }

while

while (condition) { statements }

do

do { statements }

while (condition);

switch

switch (expression) {

case expression: statements

default: statements }

try

try { statements }

catch (variable) { statements }

try { statements }

catch (variable) {

statements } finally { statements }

Whitespace: use whitespace wisely to improve readability: put a whitespace after a keyword followed by a ( ; before & after a binary operation ( + , - , / , \* , && ..); inside the for statement, after each ; to separate each part of the statement; after each , .

New lines: use new lines to separate blocks of code that perform logically related operations.

Quotes favor single quotes ' instead of double quotes " . Double quotes are a standard in HTML attributes, so using single quotes helps remove problems when dealing with HTML strings. Use template literals when appropriate instead of variable interpolation.

**Lexical Structure**

A deep dive into the building blocks of JavaScript: unicode, semicolons, white space, case sensitivity, comments, literals, identifiers and reserved words

**Unicode**

JavaScript is written in Unicode. This means you can use Emojis as variable names, but more importantly, you can write identifiers in any language, for example Japanese or Chinese, with some rules. Semicolons JavaScript has a very C-like syntax, and you might see lots of code samples that feature semicolons at the end of each line. Semicolons aren't mandatory, and JavaScript does not have any problem in code that does not use them, and lately many developers, especially those coming from languages that do not have semicolons, started avoiding using them. You just need to avoid doing strange things like typing statements on multiple lines return variable or starting a line with parentheses ( [ or ( ) and you'll be safe 99.9% of the times (and your linter will warn you). It goes to personal preference, and lately I have decided to never add useless semicolons.

White space

JavaScript does not consider white space meaningful. Spaces and line breaks can be added in any fashion you might like, even though this is in theory. In practice, you will most likely keep a well defined style and adhere to what people commonly use, and enforce this using a linter or a style tool such as Prettier.