# Part 1

## Executing JavaScript

Websites use JavaScript to provide rich functionality, but for that to happen the browser needs to know what scripts to run.

The way we do this is using the **<script>** tag. This tag informs the browser that its contents are to be executed. Here’s an example of this in action:

<html>

<head>

<script>

console.log('Hello, World!');

</script>

</head>

<body>

</body>

</html>

Doing it this way with the JavaScript directly embedded in the HTML is fine for small things, but this quickly gets difficult to manage. To avoid this, break your code into separate **.js** files and reference them in your HTML like this:

<html>

<head>

<script src="mycode.js"></script>

</head>

<body>

</body>

</html>

Then your script in **mycode.js** can look like this:

console.log('Hello, World!');

## Best Practices

### Style

Because of the flexibility of JavaScript, following a style in your code is important. Programming practices aren’t about personal taste; you need to follow a standard too keep your code clean, clear, and readable.

There are a few style guides available and different frameworks built on JavaScript may have different guidelines. Here’s one of the most popular ones:

<https://github.com/airbnb/javascript>

There’s a lot to go through in there so this section will go through some of the highlights.

### Case Sensitivity & Syntax

The JavaScript Syntax is based on C.

Keywords are typically in lower-case and are case sensitive.

Variables, parameters, members, and function names tend to be in camelCase. Classes typically use pascal case.

### Brackets

Always put your first curly bracket on the same line as the declaration.

// GOOD

function funcName() {

// ...

}

// BAD

function funcName()

{

// ...

}

### Keywords

* JavaScript has a weird keyword policy. There’s a lot of reserved words, but the majority aren’t even used.
* Just avoid certain words – use a bit of common sense – similar to Java

https://www.w3schools.com/js/js\_reserved.asp

### OPERATORS

const a = 5;         // assign the value 5 to a

const b = 2;         // assign the value 2 to b

let c = a + b;     // assign the value 7 to c

let d = a \* b;

let e = a / b;

let f = a % b;

let c += a;

let d \*= b;

**&& || ! typeof instanceof**

The typeof operator returns a string identifying the type of the value.

It’s not great, no matter what type of object It is, it will return “object”, including arrays

and null objects. Unless it’s a string or a function, which will return “string” etc.

### STRING OPERATORS

txt1 = "John";

txt2 = "Doe";

txt3 = txt1 + " " + txt2; //John Doe

txt1 = "What a very ";

txt1 += "nice day"; //What a very nice day

x = 5 + 5; //10

y = "5" + 5; //55

z = "Hello" + 5; //Hello5

### Linting

Linting is an external program that will scan your code to detect errors. It helps you write better code because it:

* Enforces style rules
* Spots difficult to see errors
* Eliminate implied globals

There are a lot of different ones with their own benefits:

* JSHint – very configurable, built in package support, not extensible.
* ESLint – custom rule support, lots of configuration, hard to use.

If you are using VS Code you can find some good tutorials to follow to get JSHint set up in your environment.

### Additional Standards

* Break a line after any punctuator
* Don’t break a line after a name, string, number, or ) ] ++ --
* Avoid tricky expressions using comma operator
* Try to avoid using extra commas in array literals
* Opening bracket on the start of your function
* Always use brackets if you can

// GOOD

[1, 2, 3];

// BAD

[1, 2, 3,];

// GOOD

if (a) { /\*code \*/ }

// BAD

if (a) /\* code \*/

### Debugging

Chrome Developer Tools are incredibly powerful. To open them press **F12**, **CTRL + SHIFT + i** or use **Menu > More Tools > Developer Tools.**

We have features such as:

* Breakpoints
* Watchers
* Executing JS Directlyer
* Altering CSS/JS Directly
* DOM Manipulation
* And much more, read the Chrome devtools documentation for more!

## ASI ( Automatic Semicolon Insertion)

**Semicolons aren’t optional!**

If you don’t provide a semicolon, ASI will try to add one for you based on a set of rules.

### Rule 1

**When a script or module is parsed from left to right, if a token is encountered that is not allowed by any production of the grammar.**

var a = 12

var b = 13

if(a) { console.log(a) }

console.log(a+b)

var c = b + a

['menu', 'items', 'listed']

.forEach(function (element) {

console.log(element)

})

*In this case, the v of var b = 13 immediately after 12 doesn’t make sense, so it puts a* ***;*** *between them. If the offending token is separated from the previous token by at least one line terminator. If the offending token is a curly brace* ***}****, then we put a semi colon in between.*

***[]*** *is an allowed character after* ***b+a****, so it won’t put a SC there, but then we’re going to get an error since it’s looking for something that doesn’t exist.*

### Rule 2

**When a script or module is parsed from left to right, the end of the input stream of tokens is encountered.**

var a = 12

var b = 13

var c = b + a

(function () {

console.log('inside my life');

console.log('doing secret stuff...');

}())

***(*** *is an allowed character after a, so it won’t get a semi colon, but again we’ll get an error.*

*This is a common issue when loading in multiple scripts.*

### Rule 3

**When a token is encountered that is allowed by some production of the grammar but the production is a restricted production and the token would be the first token of a restricted production, and the restricted token is separated from the previous token by at least one line termination.**

function returnObject() {

if (true) {

return

{

hi: 'hello'

}

}

}

*This will put a semicolon after the return, meaning hi isn’t going to be executed or be returned, not good!*

### Summary

You can live without writing semicolons yourself, and some developers will tell you they are not necessary, but you’re prone to getting bugs if you do not write them yourself. Therefore, I’d highly recommend you write them yourself.