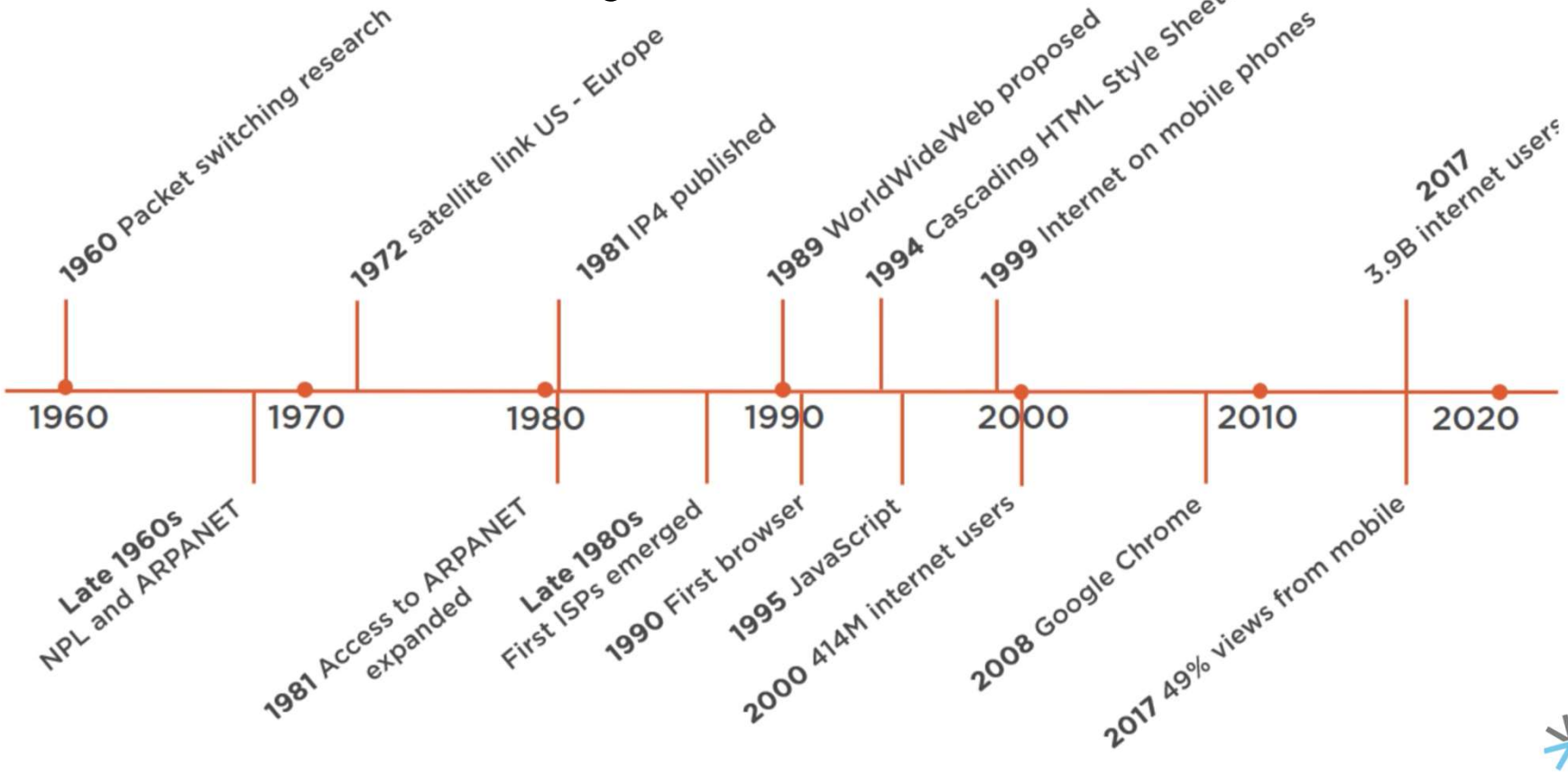
JavaScript

The Big Picture

History of the Internet



What is the World Wide Web

The **World Wide Web**

is an **information space**

where **documents** and **other web resources**

are identified by **Uniform Resource Locators (URLs),**

which may be interlinked by **hypertext**,

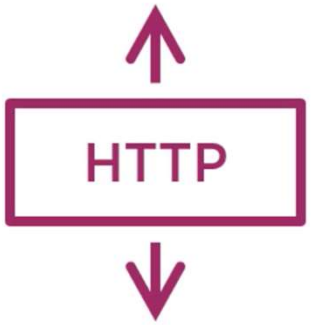
and are accessible over **the Internet**.

World Wide Web, Wikipedia, 2020
https://en.wikipedia.org/wiki/World_Wide_Web

Ingredients for the web

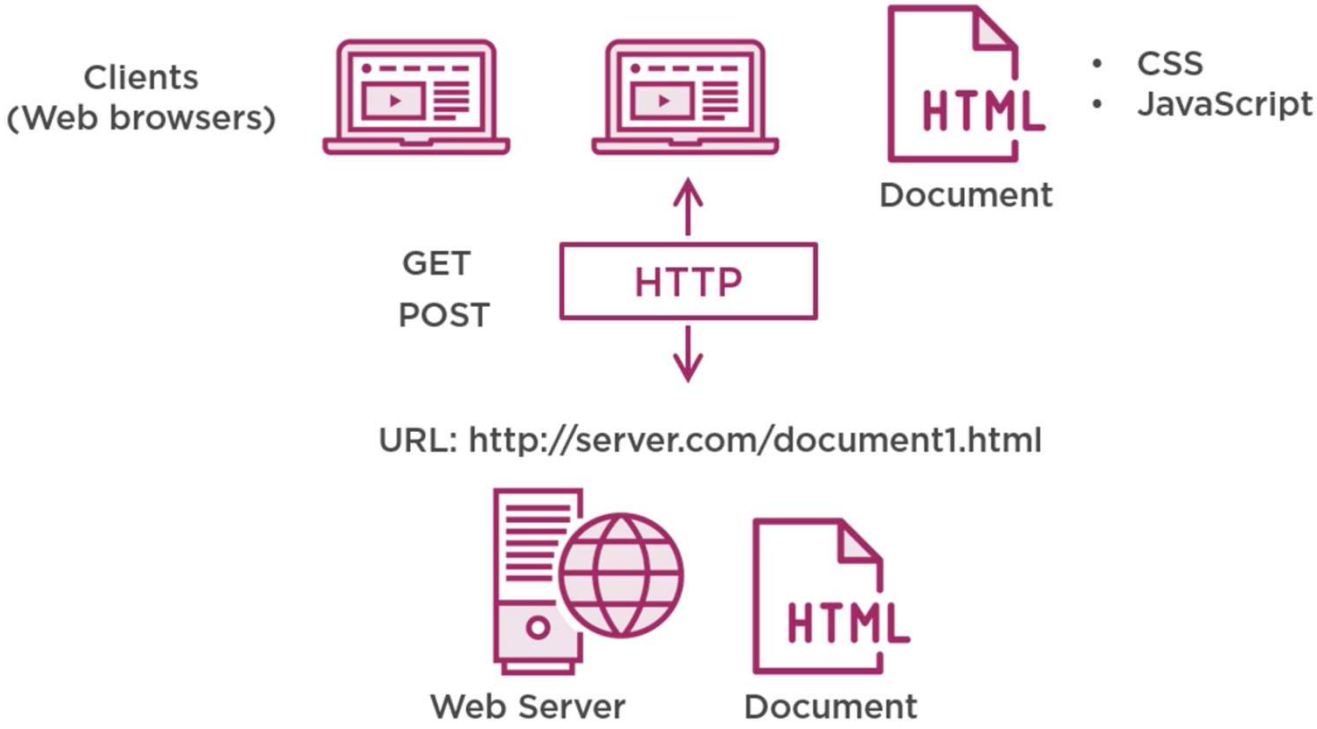
Resources (documents, images, other files)

Uniform Resource Locators (URLs)



Hypertext Transfer Protocol (HTTP)

Ingredients for the web



Thing to Remember

The web is possible because of the internet

- The internet has been around since the 80s
- The web has been around since the 90s

To use the web, you need:

- A web server, that hosts resources like documents
- URLs that identify those resources on that web server
- Protocols that can be used to transfer resources
- A web browser, to view documents and use resources

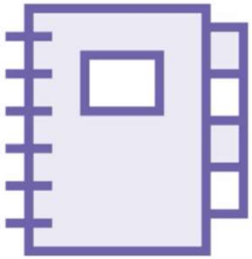
Why should you learn about HTML, CSS, and Javascript?

- **What** the technologies are, **Why** they are relevant and **HOW** people use them

Introducción a JavaScript.

JavaScript is a high-level, interpreted programming language

How to Write JavaScript from Scratch



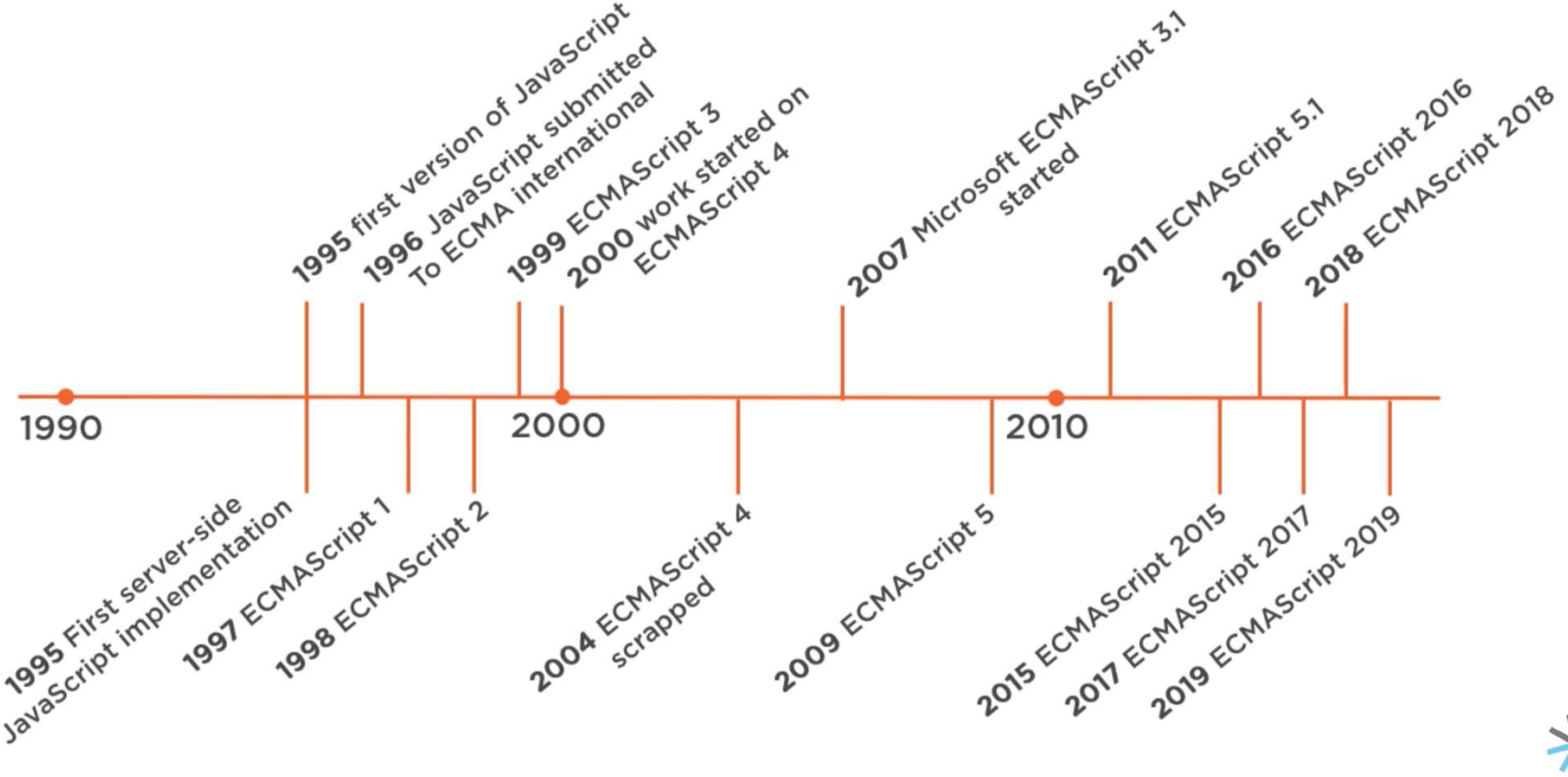
Code editor

- Notepad++
- Atom
- Visual Studio Code

Integrated Development Environment (IDE)

- Eclipse
- Visual Studio
- Webstorm

History of JavaScript



Historia – Origen del lenguaje JS

1995 - Netscape incorpora un JS desarrollado por Brendan Eich
- > Objetivo: validación de formularios en cliente (Respuesta a la lentitud de las conexiones)

1996 – Microsoft desarrolla lenguajes de Script similares
- > VBScript
- > JScript, que incorpora a Internet Explorer

1997 – Aparece ECMAScript como estándar respaldado por la European Computer Manufacturers Association

Historia – Evolución del Lenguaje JS

1998

OWA incorpora XMLHttp (AJAX), desarrollado por Alex Hopmann

2004

Gmail utiliza JS incorporando AJAX masivamente

2006

John Resig crea JQuery

2009

Aparece NodeJS, JS en el servidor, de mano de Ryan Dahl

2012

Windows 8 incorpora JS como lenguaje nativo

2014

Nuevo estándar ES6 / ES2015

Creación de NodeJS

https://youtu.be/ztspvPYybIY

- Tiene su origen en un proyecto de Ryan Dahl y sus colaboradores en la empresa Joyent,
  que fue presentado en una conferencia
  en la JSConf de 2009.

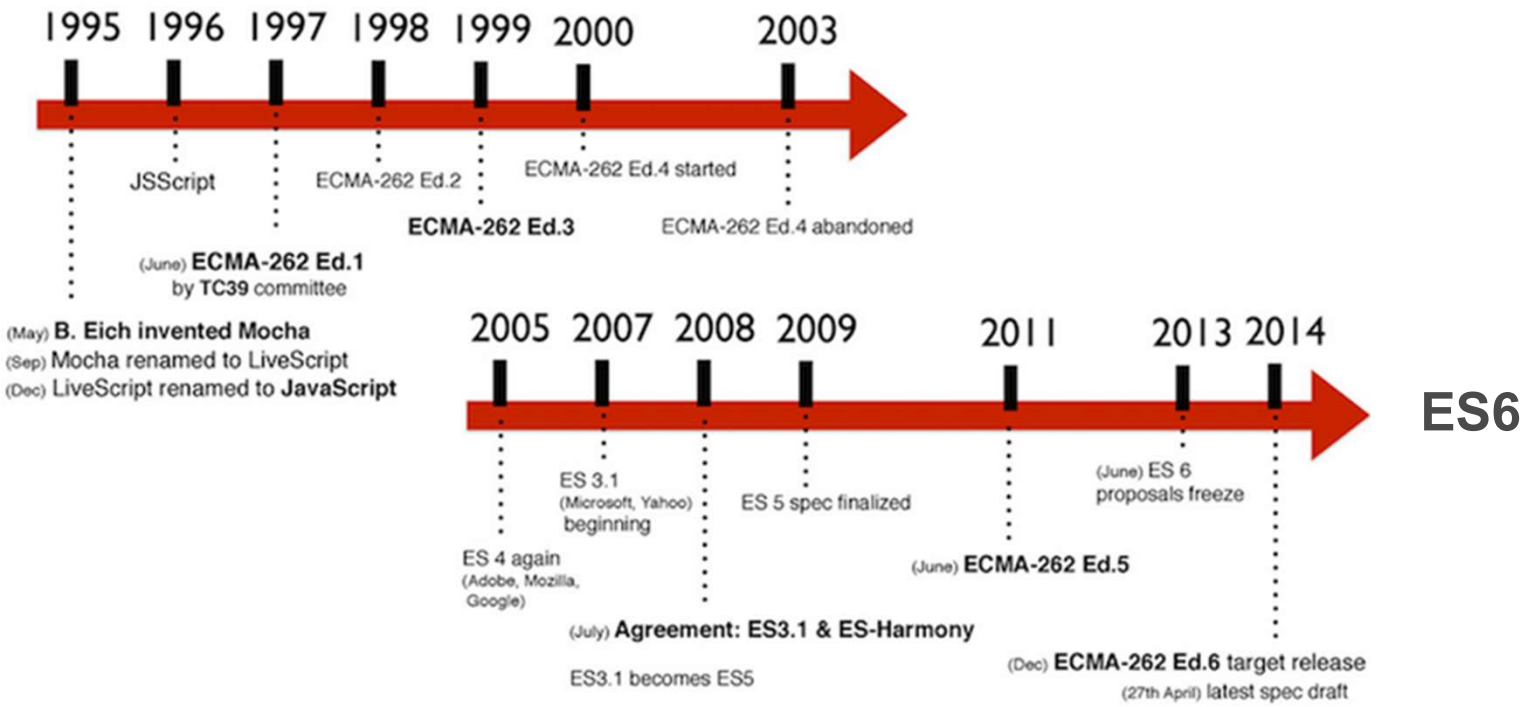
- Objetivo: escribir aplicaciones muy eficientes en E/S con el lenguaje dinámico más rápido (v8) para soportar miles de conexiones simultáneas

¿Por qué JavaScript?

JavaScript has certain characteristics that make it very different than other dynamic languages, namely that it has no concept of threads. Its model of concurrency is completely based around events

Ryan Dahl

Estándares y versiones

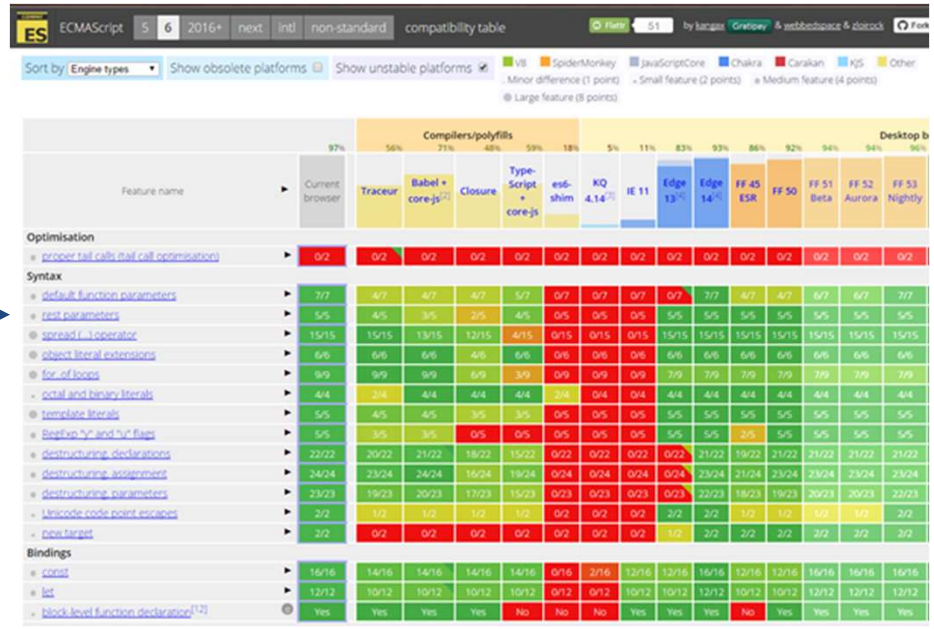


ES6: Nuevos elementos de código

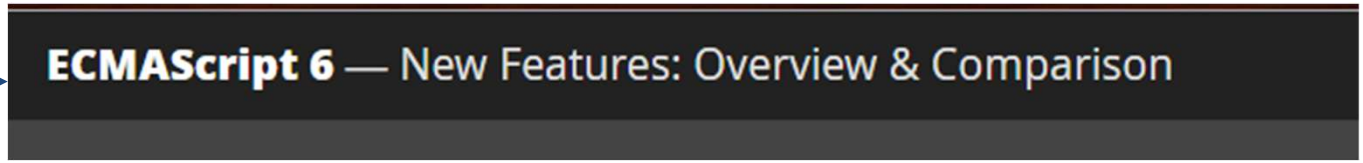
- Variables con ámbito (let) y Constantes (const).
- Template Strings: interpolación de variables
- Función Arrow. This "semántico"
- Valores por defecto
- De-structuring …
- Clases
- Promesas
- Módulos

Más información

http://kangax.github.io/compat-table/es6/



http://es6-features.org/

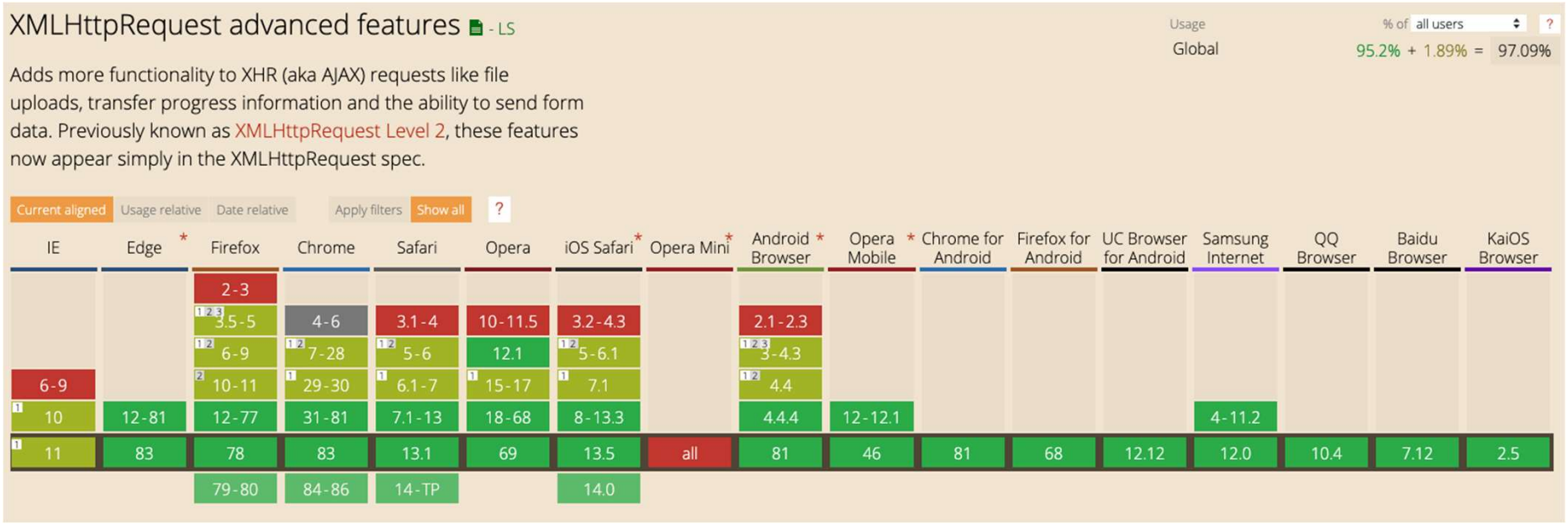


ES6+

6.0	ES 2015	Let, const, class, arrows, promesas…
7.0	ES 2016	Array.includes() / String.includes(), Operador exponencial (**)…
8.0	ES 2017	Async/Await Object.values(), Object.entries()…
9.0	ES 2018	Object Rest/Spread Iteración asíncrona, Promise.finally()
10.0	ES 2019	Array.flat() / Array.flatMap() Object. fromEntries()
11.0	ES2020	BigInt, globalThis, Importación dinámica (*lazy loading*)
12.0	ES2021	Operador de asignación lógica. String.replaceAll() Métodos & getters/setters private. Promise.any()

Browser Support of JavaScript

https://caniuse.co



JavaScript Libraries and Frameworks

- JQuery
- Dojo Toolkit
- Prototype.js
- Bootstrap
- Velocity.js
- D3.js
- Angular
- Backbone.js
- Ember.js
- Knockout
- Node.js
- Vue.js
- React

Angular Example

```
import { Component, OnInit } from
'@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit
{
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };

  constructor() { }

  ngOnInit() {
  }
}
```

```
<h2>{{hero.name | uppercase}} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div>
  <label>name:
    <input [(ngModel)]="hero.name"
placeholder="name"/>
  </label>
</div>
```

Referencia del lenguaje

- https://developer.mozilla.org/es/docs/Web/JavaScript

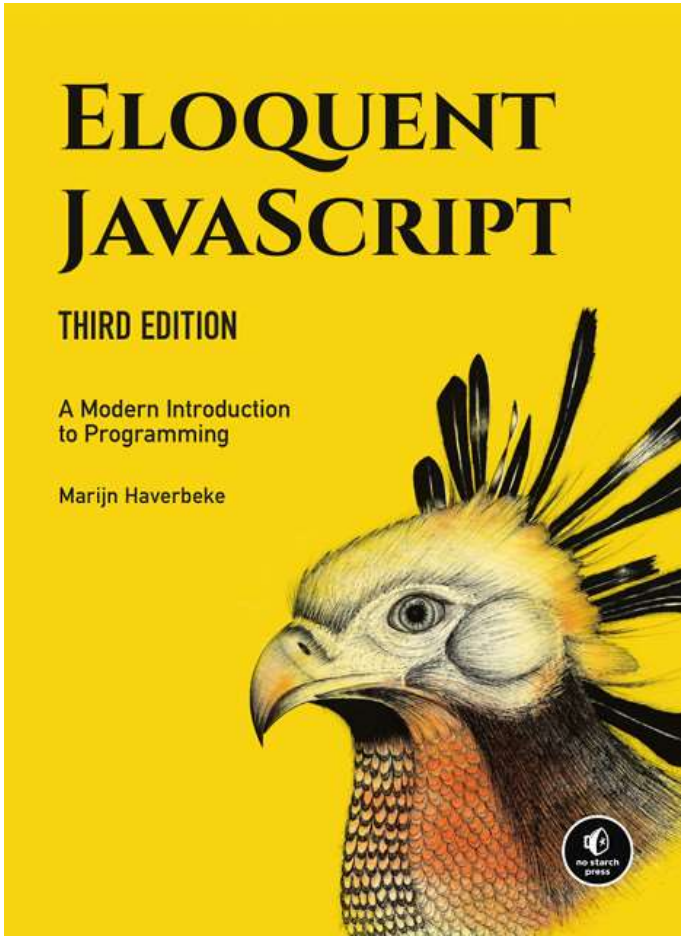
Links to Learning Resources

- General
  - https://www.w3schools.com/ - Free HTML, CSS, JavaScript (and more) tutorials
  - https://caniuse.com/ - Find out which browsers support which features
- HTML
  - https://app.pluralsight.com/paths/skills/html5 HTML skill path
- CSS
  - https://app.pluralsight.com/paths/skills/css CSS skill path
- JavaScript
  - https://app.pluralsight.com/paths/skill/javascript JavaScript skill path

Eloquent JS

Part 1: Language
Part 2: Browser
Part 3: Node

https://eloquentjavascript.net/

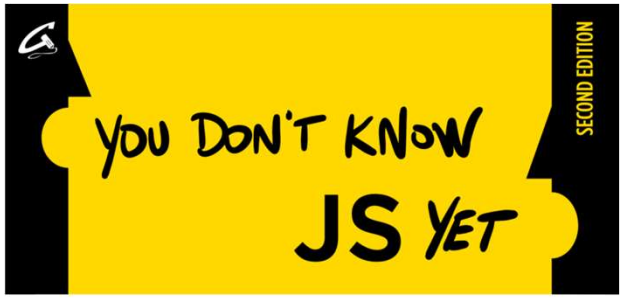


You-Dont-Know-JS

- Get Started
- Scope & Closures
- Objects & Classes
- Types & Grammar
- Sync & Async
- ES.Next & Beyond

https://github.com/getify/You-Dont-Know-JS

Kyle Simpson.



GET STARTED

Thing to Remember

JavaScript is the “glue” of the web

JavaScript is not derived from Java

JavaScript is a language that you can use to

- Interact with HTML
- Interact with the web browser
- Interact with other systems (like APIs)

The language is a standard (ECMAScript)

- Interpreted by web browsers

You can use JavaScript

- By writing it from scratch
- By using a library (React)
- By using a framework (Angular)

Elementos del entorno

Git y una cuenta en un hosting de repositorios (GitHub)
https://git-scm.com/
https://github.com/

Editor de código: Visual Studio Code
https://code.visualstudio.com/

Terminal (Linea de commandos)

Node.js y el gestor de paquetes npm
https://nodejs.org/es/

Entornos de ejecución

- El núcleo (core) del lenguaje JavaScript API mínima
  - Manejo de datos, textos, arrays, y expresiones regulares
  - No hay ninguna funcionalidad de las entradas y salidas
  - No hay funciones más sofisticadas: redes, almacenamiento, gráficos...
- Navegadores. Soporte de ES6.
  - Transpilación (babel)
- NodeJS

Entorno de trabajo: NodeJS

- Ventajas
  - Permite conocer el core del lenguaje sin prestar atención a los elementos específicos del navegador
  - Salida de datos mediante console.log()

- Desventajas
  - No hay definido un mecanismo nativo de entrada de datos (origen como entorno de servidor)
  - El sistema de Módulos de ES6 no esta implementado inicialmente

Creación de un proyecto

```
npm init
npm init -y

{
  "name": "bootcamp_IX",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Creamos .gitignore
Incluimos la línea node_modules/

El primer ejemplo.

Hola mundos

Hola Mundo. Sintaxis básica del lenguaje

- Variables y palabras reservadas. Let
- Caso (Mayúsculas y minúsculas).

- Salida de información: console. Objetos y métodos
- Funciones. No Repitiendo el código
  (DRY – Don’t Repeat Yourself)

- ‘use strict’ (a partir de ES5)

Buenas prácticas

- camelCase
- Líneas. El uso del “;”
- Comillas simples y dobles
- Comentarios

- Herramientas de estilo (Linters) – ESLint

Código. Entornos de ejecución de JS

Hola Mundo en NodeJS

La estructura de Node.js es completamente modular, incluso en los elementos nativos que componen su núcleo

Cada modulo es un fichero diferente con un espacio de nombres local

Para disponer de la funcionalidad de un módulo lo asignamos a una variable mediante el comando ***require***

```
var <nombre> = require('<nombre módulo>);
```

Si el módulo no es del sistema se indicará su *path* según la notación UNIX

Los módulos son **closures** creadas por que exponen un interfaz con una serie de funciones

Un módulo tiene → una parte pública (interfaz) y otra privada (implementación)

Módulos y CommonJS

CommonJS es una colección de estándares que surgió de la necesidad de completar aspectos que se echaban en falta en la especificación de JavaScript, especialmente en un "ecosistema" ajeno al navegador

NodeJS soporta e implementa el sistema que CommonJS define para esta gestión de módulos

- La función ***require***() para acceder a los módulos
- La variable ***exports*** dentro de los módulos.
- La variable ***module***, alternativa a la anterior

El API de CommonJS también se emplea el la implemetación de JS realizada por Microsoft para el desarrollo nativo de apps de Windows

http://www.commonjs.org/

Interfaz → parte visible en el exterior que permite utilizar el módulo a otros

Se define exportando los métodos individualmente o el objeto interfaz

métodos individualmente

```
exports.metodo_individual = <metodo>
```

objeto interfaz completo

```
module.exports =<objeto_interfaz>
```

Implementación → código del módulo que crea la funcionalidad

bloque de instrucciones del módulo del que NodeJS hará un cierre o *closure*

Código. Módulos de NodeJS

Hola mundo en NodeJS usando módulos de NodeJS
Creamos un modulo que contenga los cálculos geométricos de un círculo:

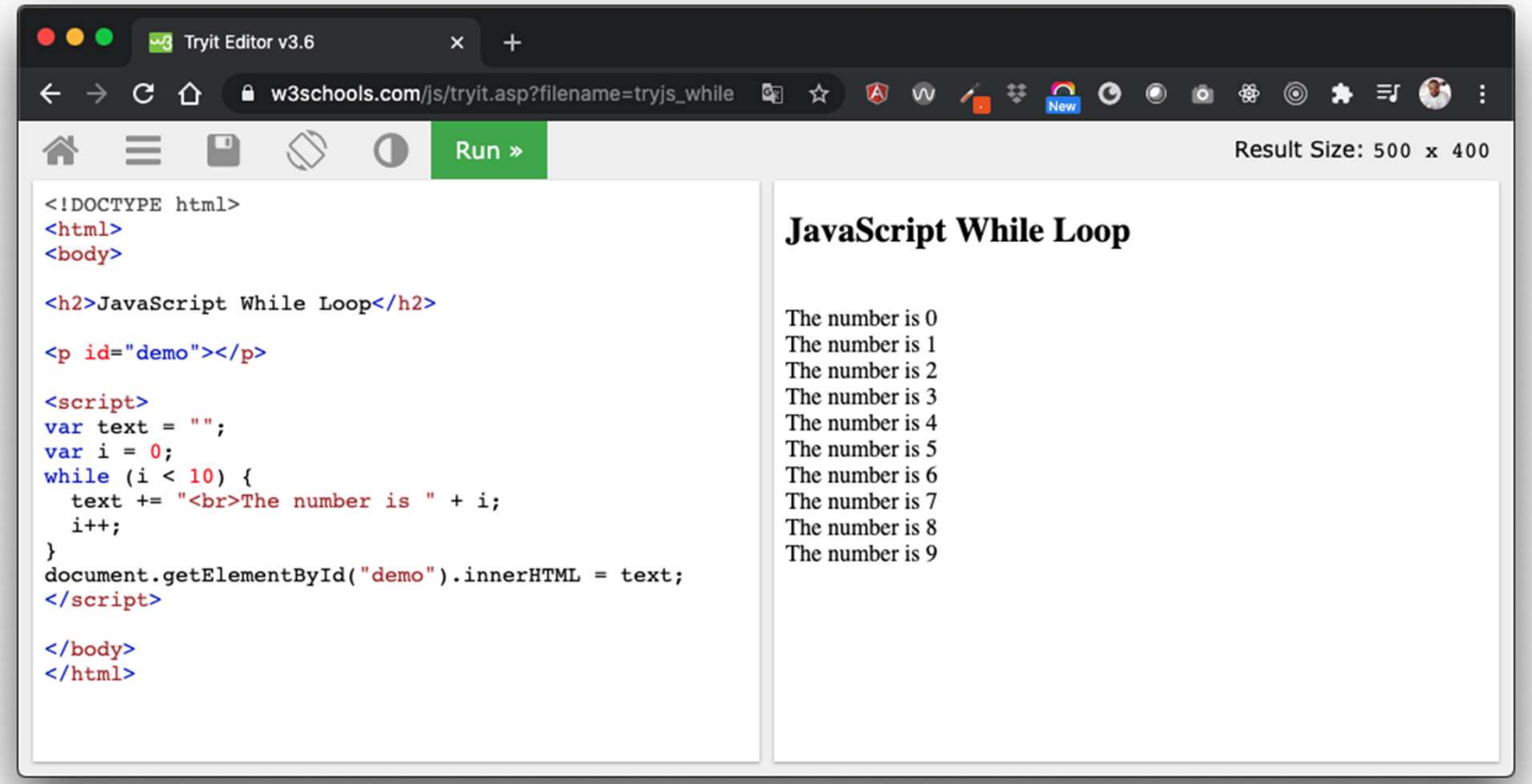
- Su circunferencia
- Su área

Entorno de trabajo: JavaScript en la Web

- JavaScript inline
- JavaScript en bloques
- JavaScript desde archivos externos
  - Posición de los scripts
  - Agrupación del código en funciones (encapsulación)
    Funciones autoinvocadas ->
    IIEF (Immediately-invoked function expression)
  - Espera a la carga completa del HTML antes de ejecutar JS
    eventos window.load / document.DOMContentLoaded (sin contar hojas de estilo, imágenes...)

Código. Entornos de ejecución de JS

Hola mundo en Chrome


Tryit Editor v3.6
w3schools.com/js/tryit.asp?filename=tryjs_while
Run »
Result Size: 500 x 400
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript While Loop</h2>

<script>
var text = "";
var i = 0;
while (i < 10) {
text += "
The number is " + i;
i++;
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
JavaScript While Loop
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

Módulos en ES6

- export -> crea automáticamente el módulo
- import -> permite importar los elementos necesarios

```
import {elementos} from 'url'
```

- en el navegador se indica que el script inicial utiliza módulos

```
<script src="url" type="module"></script>
```

Código. Módulos de ES6 en el navegador

Hola mundo en Chrome usando módulos de ES6

Módulos en ES6 en NodeJS

- export / import -> Cannot use import statement outside a module
    - la extensión de los ficheros cambia a mjs

```
main.mjs (import…)
lib.mjs (export…)
```

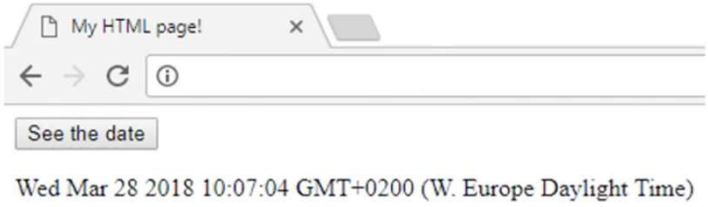
    - En package.json se define type: module

```
“type”: “module”
```

- Ya NO ES NECEARIO invocar node con el parámetro --experimental-modules; puede usarse --input-type

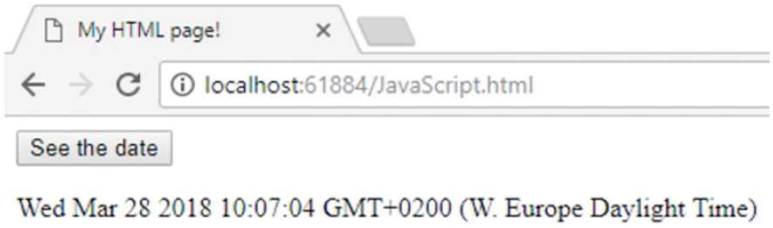
Dynamic Values with JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <title>My HTML page!</title>
</head>
<body>
    <button type="button"
            onclick="document.getElementById('demo').innerHTML = Date()">
        See the date
    </button>
    <p id="demo"></p>
</body>
</html>
```



Dynamic Values with JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <title>My HTML page!</title>
    <script>
        function ShowDate() {
            document.getElementById('demo').innerHTML = Date();
        }
    </script>
</head>
<body>
    <button type="button"
            onclick="ShowDate();">
        See the date
    </button>
    <p id="demo"></p>
</body>
</html>
```

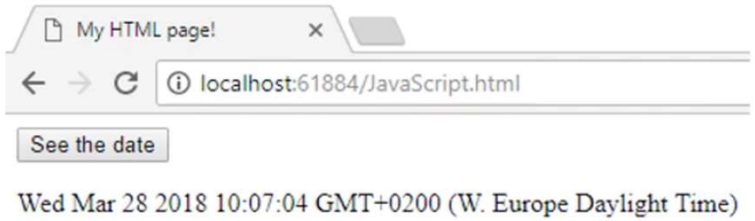


Dynamic Values with JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <title>My HTML page!</title>
</head>
<body>
    <button type="button"
            onclick="ShowDate();">
        See the date
    </button>
    <p id="demo"></p>

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

</body>
</html>
```

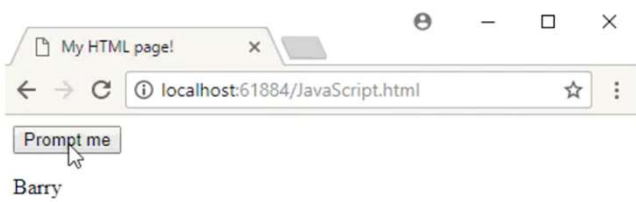
ScriptFile.js

```
function ShowDate() {
   document.getElementById('demo')
           .innerHTML = Date();
}
```

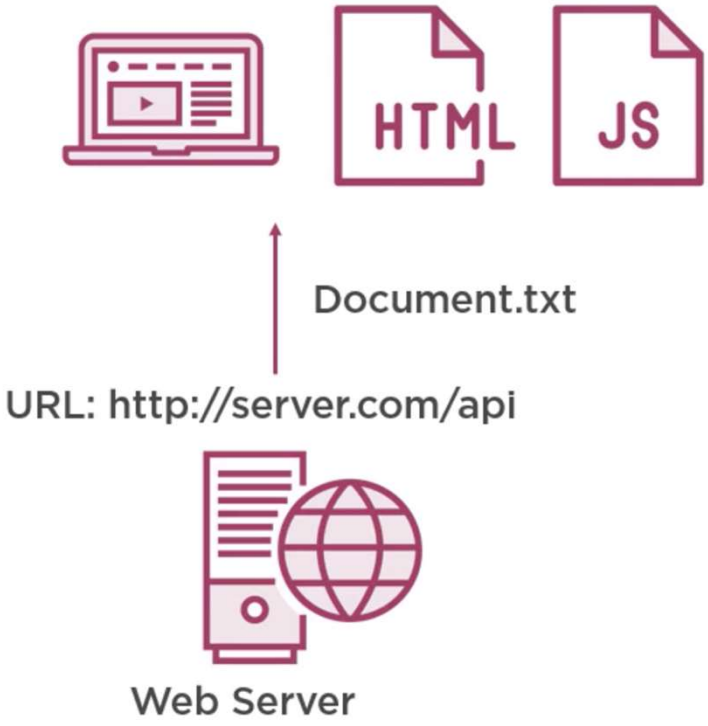


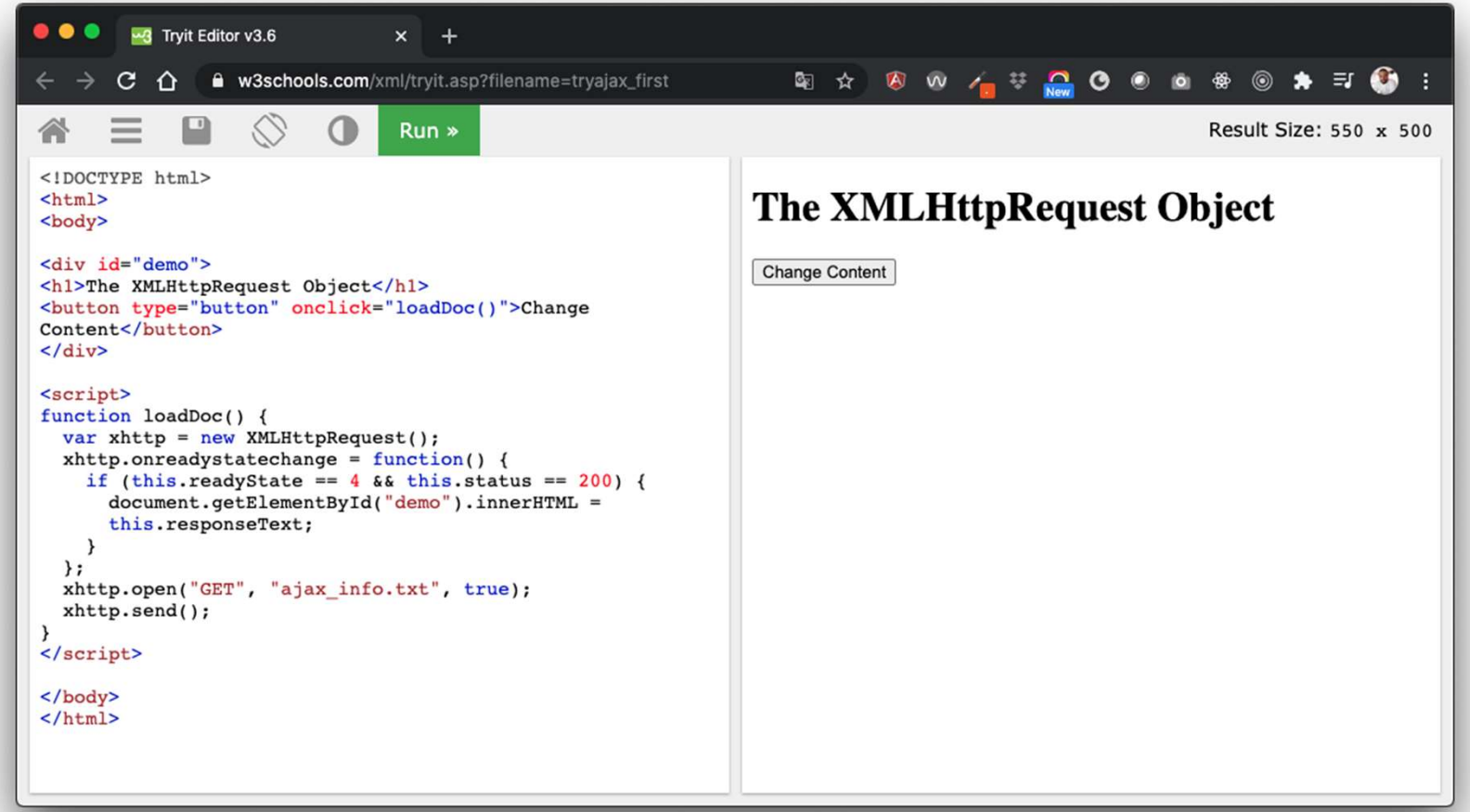
Interacting with the Browser

```
<script>

 function PromptUser() {
     var txt;
     var name = window.prompt("Enter your name");
     if (name != null || name != "") {
         txt = "Hello " + name;
     }

     document.getElementById("name").innerHTML = txt;
 }

</script>
```



Making AJAX Calls

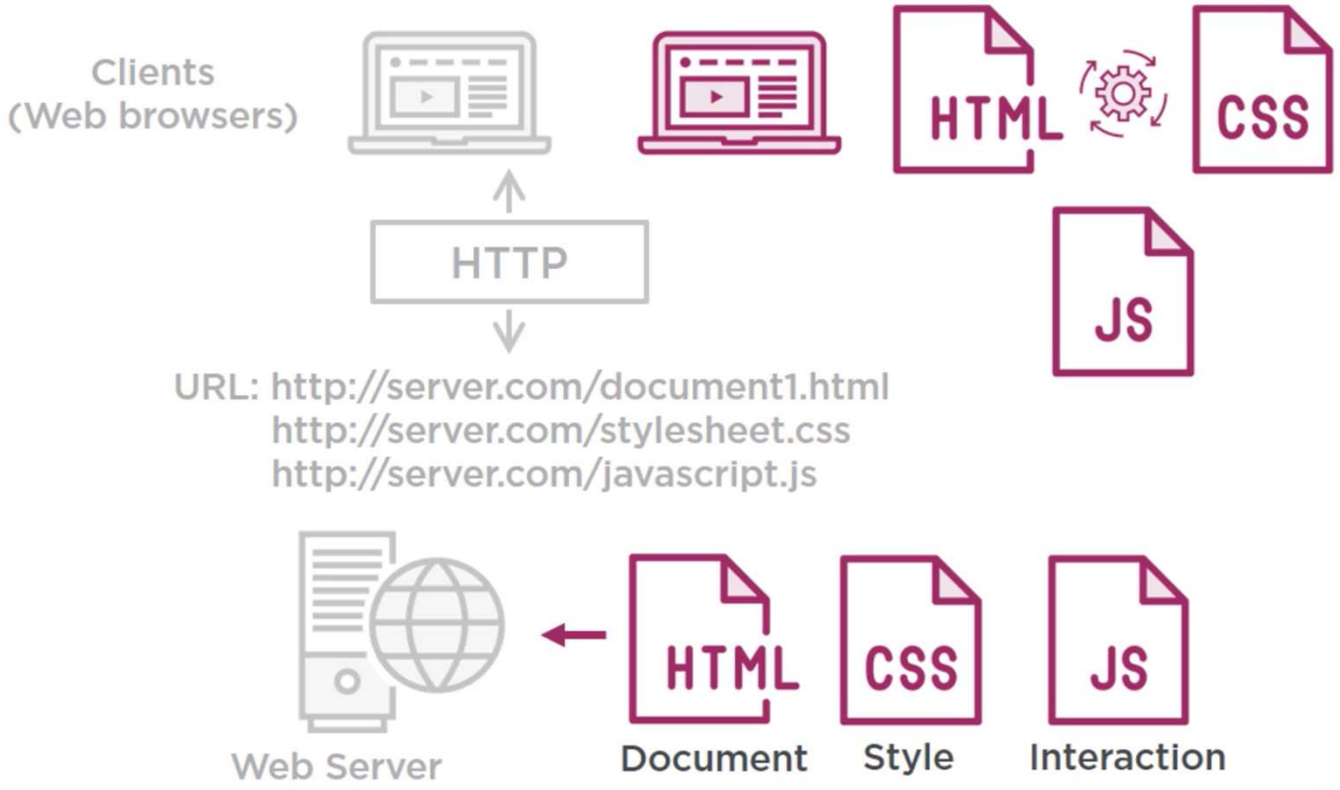



Tryit Editor v3.6
w3schools.com/xml/tryit.asp?filename=tryajax_first
Run »
Result Size: 550 x 500
<!DOCTYPE html>
<html>
<body>

<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change Content</button>

<script>
function loadDoc() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("demo").innerHTML =
this.responseText;
}
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
}
</script>
</body>
</html>
The XMLHttpRequest Object
Change Content

Ingredients for the web



JavaScript Básico. ES6.

Variables y ámbitos. Funciones.

JavaScript Core

- variables y scopes
- funciones
- control de flujo
- objetos

- types
  - Primitive Types
  - Coercion
  - Equality
- scopes (funciones)
  - Hoisting
  - Closure / Modules
- objects
  - Prototypes
  - this
  - class
  - OO vs OLOO

Kyle Simpson.

Variables y scopes

Datos

~~En JS todo ES un objeto~~

En JS todo **se comporta** como un OBJETO

Los Datos tienen TIPOS

Las variables toman el tipo de los datos

Tipado débil y dinámico

Distinguimos datos y variables

Tipos primitivos

- undefined (null -> ¿object?).
- number
  - Valores ‘especiales’
    - NaN
    - Infinity / - Infinity
    - 0 / -0
- string
  - ES6: string template
- boolean
- *symbol*
- *bigint*

- Elementales y referencias
- Mutabilidad e inmutabilidad

El operador typeof

Casting de tipos primitivos

- Casting automático o implícito (coerción)
  - A number
  - A string
  - A boolean: valores truthy / falsy
- Casting forzado
  - Number()
  - parseInt() / parseFloat()
  - +
  - String()
  - Boolean() / !!

toString()

null	"null"
undefined	"undefined"
true	"true"
false	"false"
3.14159	"3.14159"
0	"0"
-0	"0”

[]	""
[1,2,3]	"1,2,3"
[null,undefined]	","
[[[],[],[]],[]]	",,,"
[,,,,]	",,,"
{}	"[object Object]"
{a:2}	"[object Object]"

toNumber()

""	0
"0"	0
"-0"	-0
" 009 "	9
"3.14159"	3.14159
"0."	0
".0"	0
"."	NaN
"0xaf"	175

false	0
true	1
null	0
undefined	NaN

toNumber()

[""]	0
["0"]	0
["-0"]	-0
[null]	0
[undefined]	0
[1,2,3]	NaN
[[[[]]]]	0
{ .. }	NaN

toBoolean()

Falsy
false
“”
0, -0
null
NaN
undefined

Truthy
true
“foo”
23
{}, { a:1 }
[], [1,3]
function(){..}

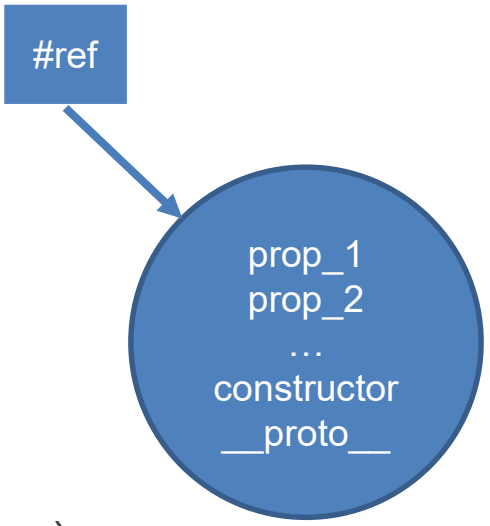
Tipos, coerción … dificultades en JS

- La coerción es inevitable. Es imprescindible. Puede ser causa de errores
- Hay que adoptar un estilo de código que haga evidentes los tipos de los valores

> A quality JS program embraces coercions, making sure the types involved in every operation are clear. Thus, corner cases are safely managed.

Tipos referenciados

- Object: tipos referenciados.
  - Literales v constructores
  - Objetos
  - Arrays
- Casting automático (coerción)
  - Objetos / Arrays
- Otros Objetos
  - Date, RegExp, Error
  - Math, JSON (Prototipos no instanciables)

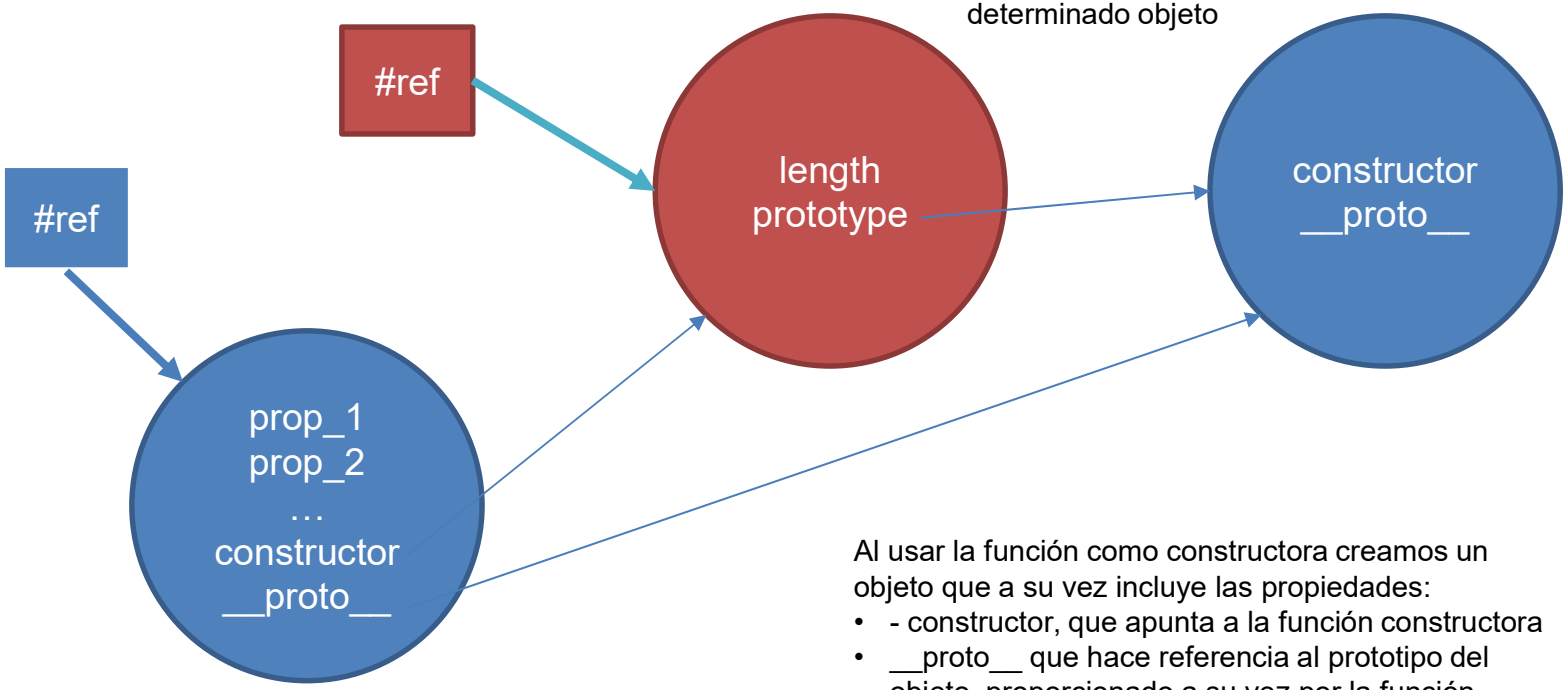


Objetos y prototipos

- En JS, casi todo, excepto los String, Number y Boolean, es un objeto, incluyendo arrays, funciones y por supuesto objetos
- Simplificando, cada objeto tiene una propiedad interna llamada prototype, que apunta a otro objeto
- Como esto se repite sucesivamente se conoce como cadena de prototipos
- Siguiendo esta cadena siempre se llega finalmente al objeto Object, cuyo prototipo es el objeto null

El prototipo (prototype)

Al crear una función sus propiedades incluye prototype que apunta a un determinado objeto



Al usar la función como constructora creamos un objeto que a su vez incluye las propiedades:

- - constructor, que apunta a la función constructora
- __proto__ que hace referencia al prototipo del objeto, proporcionado a su vez por la función constructora

Objetos envolventes (wrapper)

String()
Number()
Boolean()

En JS todo **se comporta** como un OBJETO

coerción (casting) primitivo -> object

number.length
sting.toUpperCase()

Funciones

En JS son un tipo más de Objeto -> Programación Funcional

Formas de crearlas
- Declaración
- Asignación a variables

Llamada a funciones.
  - Parámetros
  - Llamadas y referencias

Variables y scopes

- Declaración. Inicialización o asignación
  - Var. Hoisting (‘alzamiento’)
  - Variables globales
  - ES6: let, const
- Ámbito o alcance (scope):
  - funciones y bloques.

JavaScript Core: Operadores

- Operadores
  - Particularidades de la suma
  - Comparación (==) y comparación estricta (===)
  - Operador ternario

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

- Aritméticos
- Relacionales
- Lógicos o de combinación
- De asignación
- De concatenación
- Operador ternario o condicional
- Operador de tipo de datos typeof

Código. Conceptos de JS

Variables en ES6: var, let, const
Template string
Booleans en JS. Operador ternario
Primitivos (elementales) y referencias

JavaScript Core: control de flujo

- Condicionales
  - if / else if / else
  - switch / case
- Iteraciones
  - for / for in (objetos) / for of (ES6)
  - do / while
- Excepciones
  - try / catch
  - throw

Funciones

JavaScript 101: funciones

- Declaraciones v. asignación a variables de funciones anónimas
  - Hoisting
  - ES6: Array functions
- Ámbitos de las variables. Variables locales
- Argumentos y parámetros.
    - Declaración: **parámetros** (reales)
    - Invocación: parámetros formales o **argumentos**
  - Valores y referencias
  - ES6: valores por defecto
  - ES6: spread operator

Código. Funciones en JS

Creación de funciones.
Argumentos. Valores por defecto
Spread operator

JavaScript 101: funciones como objetos

- Callbacks
- Funciones anidadas
- Funciones autoinvocadas: patrón IIFE (Immediately-invoked function expression)

```
( function (parámetros) {
        código de la función
} (argumentos))
```

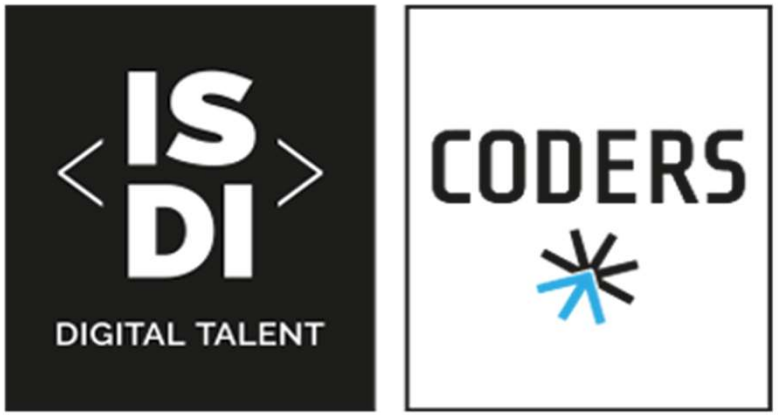
- Closures

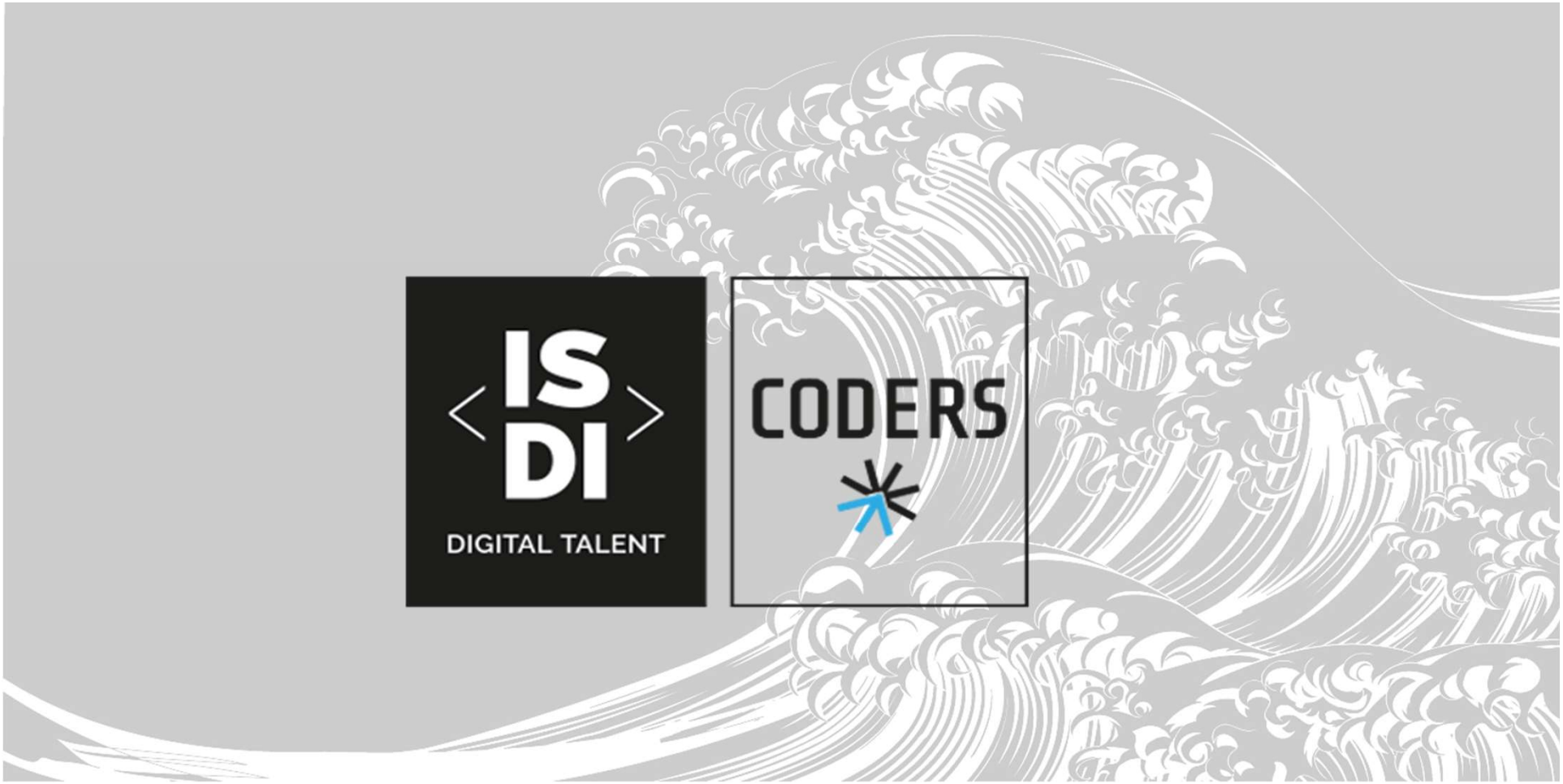
Callbacks

- Es una pieza de código ejecutable (función) que se pasa como argumento a otro código.
- Con la callback estamos logrando un comportamiento asincrónico, es decir, de no bloqueo
- Si tras una llamada a una función asíncrona queremos hacer algo, le pasaremos un argumento más correspondiente a una función callback, para que la invoque cuando termine.
- Pueden ser usados en funciones síncronas, como muchos de los métodos de arrays (foreach, map, filter…)

Código. Funciones en JS

Callbacks
Funciones anidadas
Funciones autoinvocadas


ISDI
DIGITAL TALENT
CODERS


ISDI
DIGITAL TALENT
CODERS