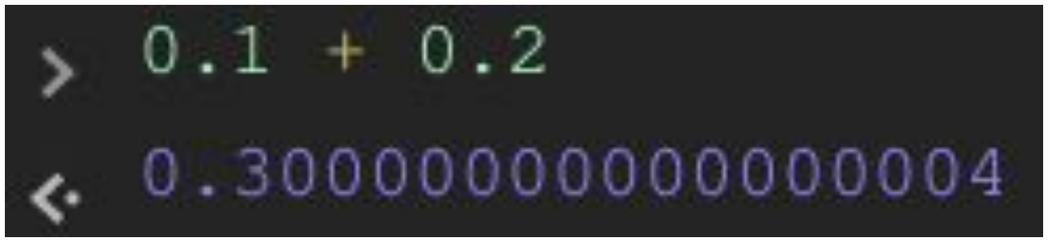
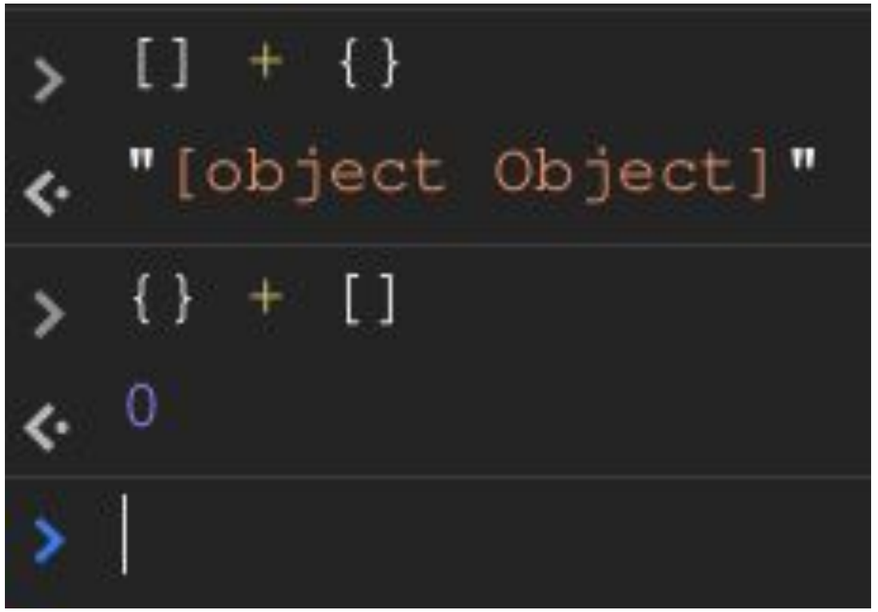
Introducción a JavaScript

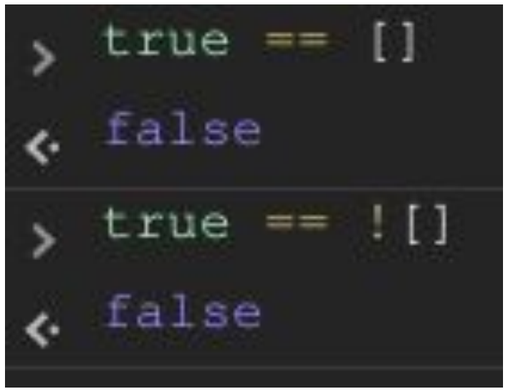
¿Qué es JavaScript?

Un lenguaje para dominarlos a todos

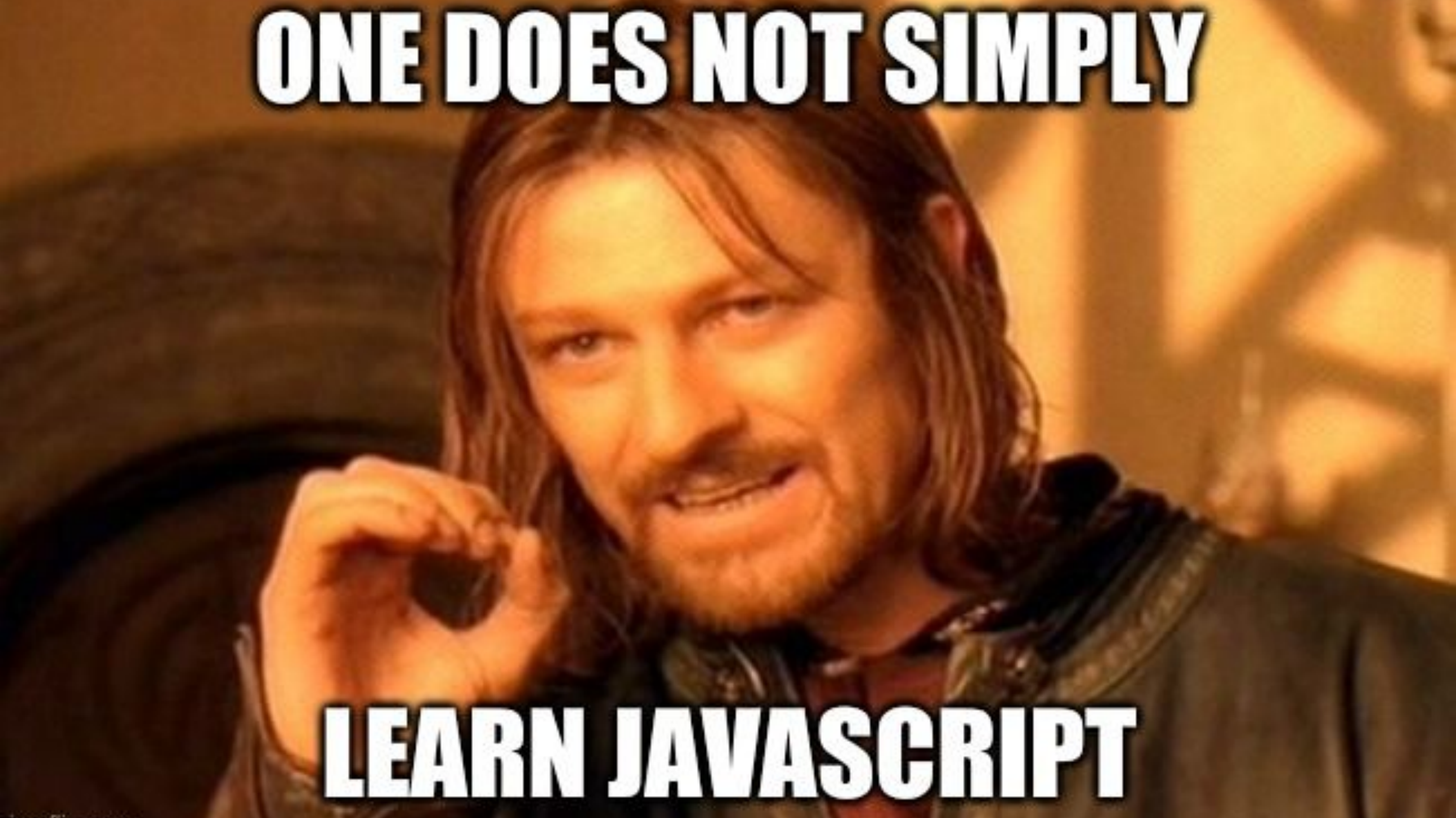
```
> true + true
< 2
```



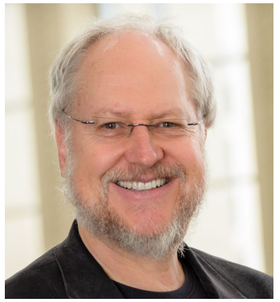





YO
VOSOTROS


ONE DOES NOT SIMPLY
LEARN JAVASCRIPT

Un poco de historia



1995 Brendan Eich crea JavaScript para NetScape

1998 Microsoft crea XMLHttpRequest, padre de AJAX

1999 Se empieza a hablar de Web 2.0

2000 Douglas Crockford inventa JSON

2004 Gmail utiliza JS incorporando AJAX masivamente

2006 John Resig crea JQuery

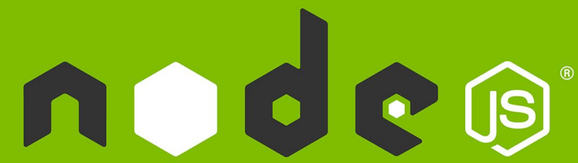
2009 Nace Google Chrome y su motor de JS: V8

2009 Ryan Dahl crea NodeJS usando el motor V8 de Chrome

2012 Windows 8 incorpora JS como lenguaje nativo

2014 Nuevo estándar ES6 / ES2015

¿Cómo funciona?



JS Engine

JS

V8

Tipos primitivos

- **undefined:** valor indefinido
- **null:** valor nulo
- **Number:** 3, 3.1416, NaN, +/-Infinity, +-0
- **String:** “hola amigo”, ‘hola amigo’, `hola ${friend}`
- **Boolean:** true/false
- **Bigint:** para representar números muy grandes
- **Symbol:** para metaprogramar

Sintaxis 101


```
1  const name = 'Alberto Casero';
2  var email = "alberto@kasfactory.net";
3
4  let instructor = {
5      name: name,
6      email: email,
7      age: 35,
8      languages: ['Spanish', 'English', 'Galician', 'Python', 'JavaScript', 'PHP']
9  }
10
11 let run = true
12 while (run) {
13     for (let language of instructor.languages) {
14         if (language == 'Spanish') {
15             console.log('Hola amigo!')
16         } else {
17             switch (language) {
18                 case "English":
19                     console.log('Hello my friend');
20                     break;
21                 case 'Galician':
22                     console.log("Hola meu amigo")
23                     break
24                 default:
25                     // Here we are using a template string
26                     console.log(`Hello my friend using ${language}`)
27                     break;
28             }
29         }
30     }
31     run = false
32 }
```

CONSTANTE

VARIABLES

OBJETO LITERAL

VALOR ENTERO

ARRAY

BUCLE WHILE

BUCLE FOR

IF / ELSE

SWITCH

PINTAR POR PANTALLA

COMENTARIO

TEMPLATE STRING

VALOR BOOLEANO

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

Operaciones

```
// Operadores aritméticos
let a = 5;
let b = 2;

console.log('Suma', a + b);
console.log('Resta', a - b);
console.log('Multiplicación', a * b);
console.log('División', a / b);
console.log('Exponente', a ** b);  // a elevado a b
console.log('Módulo (resto)', a % b);
console.log('Negación', -a);
console.log('Incremento', a++);
console.log('Decremento', a--);
console.log('Pre-incremento', ++a);
console.log('Pre-decremento', --a);
```

```
// Operadores booleanos
console.log('And', a && b)
console.log('Or', a || b)
console.log('Not', !a)
console.log('Igual', a == b) // '4' == 4 -> true
console.log('Igual de verdad', a === b) // '4' === 4 -> false
console.log('Distinto', a !== b) // '4' != 4 -> false
console.log('Distinto de verdad', a !== b) // '4' !== 4 -> true
console.log('Mayor que', a > b)
console.log('Mayor o igual', a >= b)
console.log('Menor que', a < b)
console.log('Menor o igual', a <= b)
```

El bucle do...while

```
let i = 1;
do {
    console.log('Hello!', i)
    i++
} while(i < 10)
```

El bucle for

```
const fighters = ["Bud Spencer", "Chuck Norris", "Van Damme"];

for (let i = 0; i < fighters.length; i++) {
    const fighter = fighters[i];
    console.log(fighter);
}

for (let i in fighters) {
    const fighter = fighters[i]
    console.log(fighter);
}

for (let fighter of fighters) {
    console.log(fighter);
}
```

let, var y const

```
var x = 1;
let y = 1;

if (true) {
  var x = 2;
  let y = 2;
}

console.log(x); // devuelve 2
console.log(y); // devuelve 1
```

var es ***function scoped***
let es ***block scoped***
const es ***block scoped***

Bloques y variables

Bloques

En JavaScript, un bloque es cualquier conjunto instrucciones entre llaves **{ }.**

Las variables declaradas con **var**, son accesibles y se pueden modificar desde otro bloque.

Las declaradas con **let**, no.

```
const name = 'Alberto Casero';
var email = "alberto@kasfactory.net";

let instructor = {
    name: name,
    email: email,
    age: 35,
    languages: ['Spanish', 'English', 'Galician', 'Python', 'JavaScript', 'PHP']
}

let run = true
while (run) {
    for (let language of instructor.languages) {
        if (language == 'Spanish') {
            console.log('Hola amigo!')
        } else {
            switch (language) {
                case "English":
                    console.log('Hello my friend');
                    break;
                case 'Galician':
                    console.log("Hola meu amigo")
                    break
                default:
                    // Here we are using a template string
                    console.log(`Hello my friend using ${language}`)
                    break;
            }
        }
    }
    run = false
}
```

Valores de variables en memoria

```
let a = 1

const book = “When Mr.Bilbo Baggins...”


const members = [“Frodo”,“Sam”]

members.push(“Gandalf”)
```

RAM ASIGNADA AL PROGRAMA

a	1	*#2301*
book	“When Mr.Bilbo Baggins...	*#2302*
	...The end.”	*#2303*
members	=> #2305	*#2304*
	“Frodo”	*#2305*
	“Sam”	*#2306*
	“Gandalf”	*#2307*
		#2308

Funciones

Funciones

```
function hello(name, surname = 'Skywalker') {
    return `Hello ${name} ${surname}!`;
}
```

```
const hello2 = function(name, surname = 'Skywalker') {
    return `Hello ${name} ${surname}!`;
}
```

Arrow => functions

```
const hello3 = (name, surname = 'Skywalker') => {
    return `Hello ${name} ${surname}!`;
}
```

```
const hello4 = (name, surname = 'Skywalker') => `Hello ${name} ${surname}!`
```

```
const hello5 = name => `Hello ${name}!`
```

⚠️ **Las arrow functions mantienen el scope del bloque anterior** ⚠️

Clases, objetos, prototipos...

JavaScript y sus objetos

En Javascript todo son objetos (y si no, se comportan como tal).

Cada objeto tiene una propiedad **prototype** que apunta su prototipo.

El prototipo tiene a su vez su propiedad **prototype** que apunta a otro prototipo, y así sucesivamente hasta llegar al objeto Object (prototipo padre), cuyo prototipo es null.

A esto se le llama cadena de prototipos.

Clases

```
function Team(name) {

    this.name = name;

    this.play = function() {
        console.log(`${this.name} is playing!`);
    }
}

Team.prototype.train = function () {
    console.log(`${this.name} is training!`);
}

const bulls = new Team('Chicago Bulls');
bulls.play()
bulls.train()
```

```
class Team {
    constructor(name) {
        this.name = name;
        this.players = []
    }

    play() {
        console.log(`${this.name} is playing!`);
    }

    train () {
        console.log(`${this.name} is training!`);
    }

    static info() {
        console.log('This is an static method')
    }

    get squad() {
        return this.players.join(',')
    }

    set squad(players) {
        this.players = []
        for (const player of players) {
            this.players.push(player.name)
        }
    }
}
```

Método estático

Getters y Setters

Esto no cambia

Herencia

```
class BasketballTeam extends Team {
    constructor(name) {
        super(name)
        this.sport = 'Basketball'
    }

    getInitialSquad() {
        this.players.shuffle()  // shuffle no existe
        return this.players.slice(0, 4)
    }

    play() {
        const initialSquad = this.getInitialSquad().join(',')
        console.log(`${this.name} is playing with ${initialSquad}`)
    }
}
```

```
function BasketballTeam (name, players) {
    this.name = name;
    this.players = players;

    this.getInitialSquad = function() {
        this.players.shuffle();  // shuffle no existe
        return this.players.slice(0, 4);
    }

    this.play = function() {
        const initialSquad = this.getInitialSquad().join(',');
        console.log(`${this.name} is playing with ${initialSquad}`);
    }
}

BasketballTeam.prototype = new Team();
```

Herencia clásica

Herencia “prototipada”

...y objetos de tipos primitivos...

Objetos de tipos primitivos

En JavaScript definen objetos para cada uno de los tipos de datos primitivos:

objetos String, objetos Number, objetos Boolean, objetos Array

Éstos objetos almacenan los mismos valores de los tipos de datos primitivos y añaden propiedades y métodos para manipular sus valores.

Todos éstos objetos, heredan de (o su prototipo es) Object.

...y objetos literales

Objetos literales

```
const goat = {
    name: 'Michael Jordan',
    team: 'Chicago Bulls',
    number: 23,
    shoot: function() {
        return 3 // always in
    }
}
```

```
const goat = new Object();
goat.name = 'Michael Jordan';
goat.team = 'Chicago Bulls';
goat.number = 23;
goat.shoot = function() {
    return 3; // always in
}
```

Son instancias de la clase Object, que se crean con una sintaxis especial.

Vienen a ser los “diccionarios” en JavaScript.

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

Spreading

Spread operator

```
// Spread Operator ...

//Combine arrays
let mid = [3, 4];
let arr = [1, 2, ...mid, 5, 6]; //[1, 2, 3, 4, 5, 6]

//Copy arrays
let arr = [1,2,3];
let arr2 = [...arr]; // like arr.slice()
arr2.push(4) //[1, 2, 3, 4]

//Calling functions Apply
const arr = [2, 4, 8, 6, 0];
const max = Math.max(...arr);

console.log(max); //8
```

https://medium.com/@serbanmihai/javascript-es6-cheatsheet-spread-operator-d39089ae24a1

Modules

“Los buenos escritores, dividen sus libros en capítulos.

Los buenos programadores, dividen su código en módulos.”


CommonJS
Modules
ES6
Modules
AMD
Modules

CommonJS Modules

```
// square.js
module.exports = class Square {
  constructor(width) {
    this.width = width;
  }
  area() {
    return this.width ** 2;
  }
};
```

```
const Square = require('./square.js');
const mySquare = new Square(2);
console.log(`The area of mySquare is ${mySquare.area()}`);
```

```
// circle.js
const { PI } = Math;
exports.area = (r) => PI * r ** 2;
exports.circumference = (r) => 2 * PI * r;
```

```
const circle = require('./circle.js');
console.log(`The area of a circle of radius 4 is ${circle.area(4)}`);
```

AMD Modules

- Iguales que los módulos CommonJS pero de naturaleza asíncrona.
- Pensados para ser utilizados en los navegadores web.

ES6 Modules

Los módulos JavaScript permiten partir nuestro código en diferentes archivos para poder re-utilizarlo en diferentes sitios.

Los módulos ES6 forman parte de la propia especificación del lenguaje.

Su carga es asíncrona.

Todo lo que hay en su interior es privado, salvo que se marque como exportable.

Module Cheatsheet

Name Export → Name Import

```
export const name = 'value'
```

```
import { name } from '...'
```

Default Export → Default Import

```
export default 'value'
```

```
import anyName from '...'
```

Rename Export → Name Import

```
export { name as newName }
```

```
import { newName } from '...'
```

Export List + Rename → Import List + Rename

```
export {
   name1,
   name2 as newName2
}
```

```
import {
   name1 as newName1,
   newName2
} from '...'
```

samanthaming samanthaming.com samantha_ming

Destructuring

```
// Destructuring objects
let person = {first_name: 'Joe', last_name: 'Appleseed'};
let {first_name, last_name} = person; //first_name = 'Joe', last_name = 'Appleseed'

// Destructuring arrays
const iterable = ['a', 'b'];
const [x, y] = iterable; // x = 'a'; y = 'b'

// Destructuring for-of loop
const arr = ['a', 'b'];
for (const [index, element] of arr.entries()) {
    console.log(index, element);
    // 0 a
    // 1 b
}
```

https://medium.com/@serbanmihai/javascript-es6-cheatsheet-spread-operator-d39089ae24a1

Arrays y programación funcional

Arrays

- Los arrays en JavaScript son objetos.
- Al ser objetos, tienen prototipo y ofrecen una serie de métodos
- Muchos de éstos métodos son comunes en programación funcional

  https://wweb.dev/resources/js-array-functions-cheatsheet

map

```
[ 2, 3, 4 ].map(val ⇒ val * 2)
// [ 4, 6, 8 ]
```

- Sirve para “mapear” un array
- Es decir, convertir un array en otro array con el mismo número de elementos pero diferente contenido

filter

```
[1, 10, 5, 6].filter(val ⇒ val > 5)
// [ 10, 6 ]
```

- Sirve para “filtrar” un array
- Devuelve una array sólo con los elementos que cumplen la condición

find

```
[1, 10, 5, 6].find(val ⇒ val > 5)
// 10
```

- Sirve para encontrar un elemento de un array
- Devuelve el primer elemento que cumpla la condición
- Si no lo encuentra, devuelve undefined

reduce

```
[ 5, 10, 15 ].reduce((accumulated, current) ⇒ accumulated + current, 0)
// 30
```

- Sirve para “reducir” un array a un valor
- Típico cuando queremos recorrer un array para hacer una suma del total

forEach

```
[ 1, 2, 3 ].forEach(val ⇒ console.log(val))
// 1
// 2
// 3
```

- Sirve para recorrer los elementos de un array de una forma “funcional”
- No devuelve ningún valor

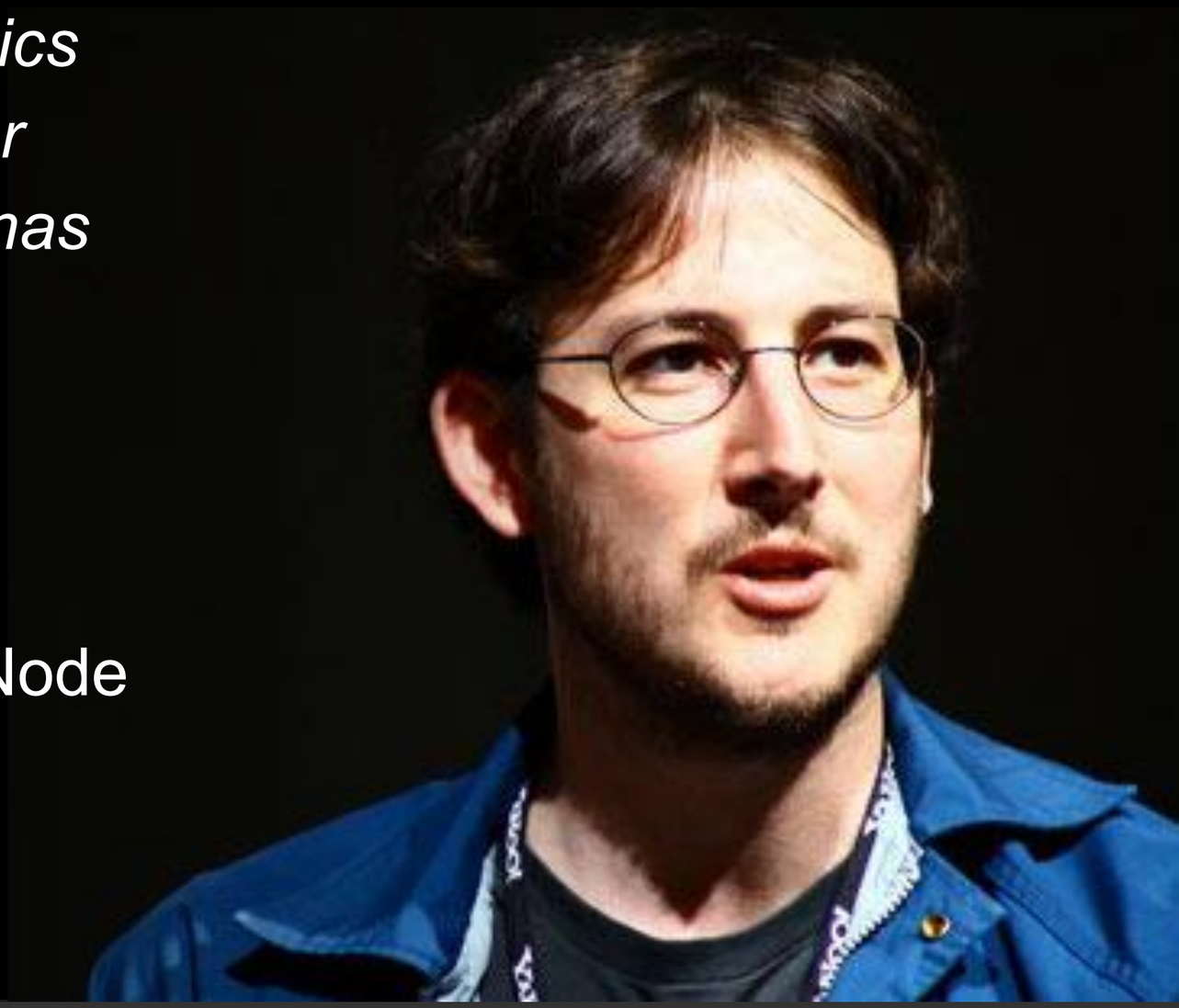
Asincronía


JBaños

Asincronía

*“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 - El de Node




Callbacks

Controlando el tiempo y callbacks

```
function callback() {
    console.log('Termine!');
}

console.log('Empiezo');
setTimeout(callback, 2000);
```

```
let times = 0
const intervalID = setInterval(() => {
    console.log('Beeeeeep! 🔔')
    times++
    if (times > 2500) {
        clearInterval(intervalID)  // finaliza el interval
    }
}, 1000)
```

setTimeout

Permite ejecutar una función pasados un tiempo que se indica en milisegundos.

setInterval

Permite ejecutar una función periódicamente cada un tiempo que se indica en milisegundos.

- A la función que se pasa como parámetro en ambos casos se le llama callback.
- La idea de **un callback es “dejar programado lo que que queremos que pase” cuando suceda algo**: que pase un tiempo, que el usuario haga click en un botón, que la base de datos me responda con los resultados, que me termine de descargar unos datos de internet….

Promesas

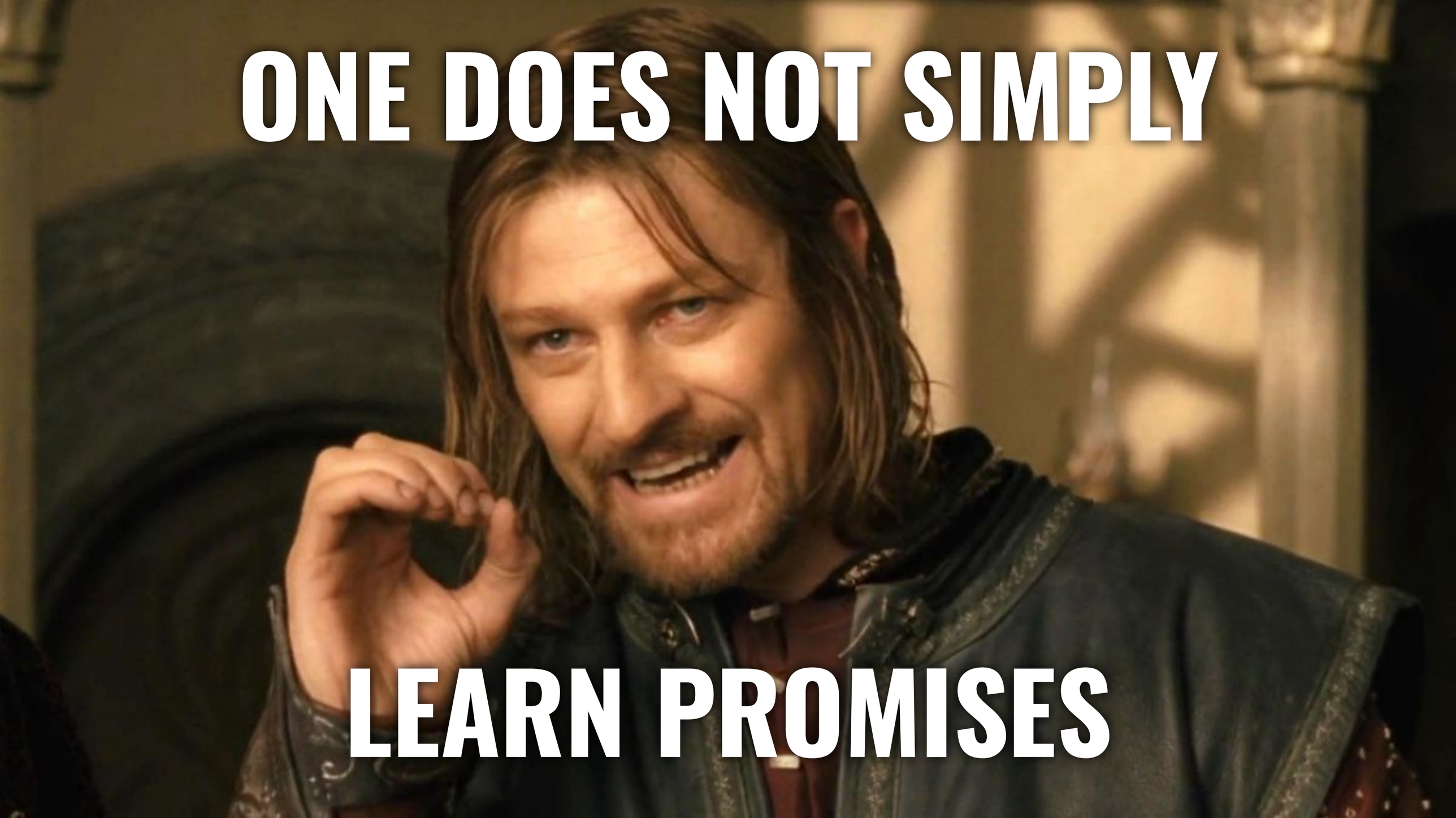
Promesas / Promises

```
function doYouLoveMe(name) {
    return new Promise(function(resolve, reject) {
        console.log('Let me think about it...')
        setTimeout(() => {
            if (name == 'Alberto') {
                resolve('Yes!')
            } else {
                reject('Sorry but no...')
            }
        }, 1000)
    })
}
```

```
const lovePromise = doYouLoveMe('Alberto')
lovePromise.then(response => {
    console.log(response)
}).catch(error => {
    console.error(error)
})
```

```
const lovePromise = doYouLoveMe('Alberto')
lovePromise.then(response => {
    console.log(response)
}, error => {
    console.error(error)
})
```

- Una promesa es un objeto que representa una operación que aún no se ha completado, pero que se completará más adelante.
- Tiene tres estados posibles:
  - Pendiente (pending)
  - Completada (settled)
    - Satisfactoriamente (fulfilled): cuando ha ido todo bien
    - Rechazada (rejected): cuando algo ha ido mal


ONE DOES NOT SIMPLY
LEARN PROMISES


ASYNC/AWAIT
YOU SHALL NOT SEEM ASYNCHRONOUS


Ignasi Marimon-Clos
@ignasi35
Async await
Async await
Async await
Async await
In the jungle
The mighty jungle
The lion sleeps tonight.

Async/await

```
function loveChecker() {
    doYouLoveMe('Alberto').then(response => {
        console.log('Alberto', response)
    }, error => {
        console.error('Alberto', error)
    })
    doYouLoveMe('Frodo').then(response => {
        console.log('Frodo', response)
    }, error => {
        console.error('Frodo', error)
    })
}
```

```
async function loveChecker() {
    try {
        let response = await doYouLoveMe('Alberto')
        console.log('Alberto', response)
    } catch (error) {
        console.error('Alberto', error)
    }
    try {
        response = await doYouLoveMe('Frodo')
        console.log('Frodo', response)
    } catch (error) {
        console.error('Frodo', error)
    }
}
```

- Con async/await conseguimos que nuestro código parezca síncrono.
- Al usar **async** en una función, JS **envuelve la función en una promesa**
- Al usar **await**, JS **maneja el callback de la promesa por nosotros**.
- A esto (que el lenguaje haga cosa por nosotros usando ciertas palabras) se le llama sugar syntax.


It's done.

A wizard is never late, nor is he early. He arrives precisely when ~~he means to~~ *the promise is resolved.*


¡GRACIAS!