

ES 2015 +
IGNACIO ANAYA
IANAYA89.COM

QUE ES ES?



HISTORIA



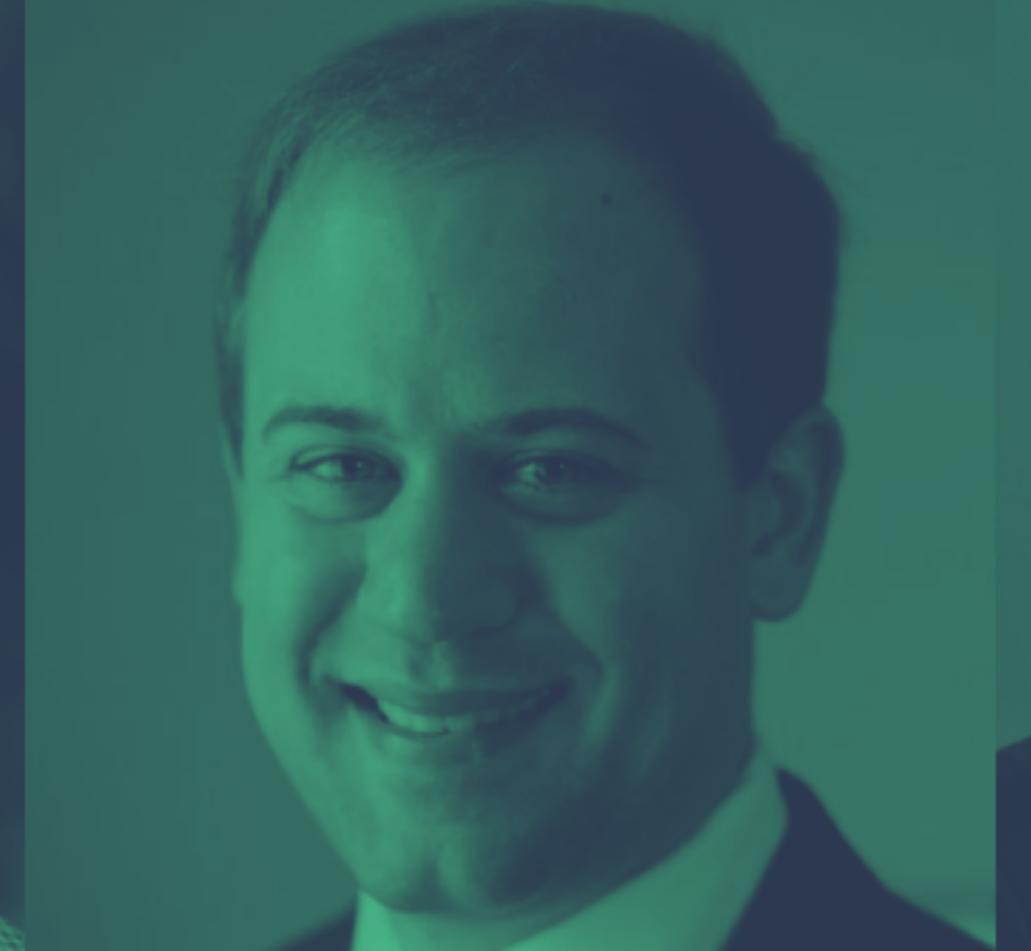
- » Mayo 1995 - JavaScript 🙌
- » Agosto 1996 - JScript 😈
- » Junio 1997 - EcmaScript 🥇
- » Julio 2008 - TC39 💯

ECMASCRIPT



- » Sintaxis
- » Semantica
- » Librerias

TC39



VERSIONES

12
34

» 1997 → v1.0

» 1998 → v2.0

» 1999 → v3.0

» ~~v4.0~~

» 2009 → v5.0

VERSIONES

12
34

» 2011 → 5.1

» 2015 → ~~v6.0~~ v2015

» 2016 → ~~v7.0~~ v2016

» 2017 → ~~v8.0~~ v2017

» ? → ES Next

NUEVAS FEATURES



TAIL CALLS

- » Mejora el uso de memoria en funciones recursivas.
- » Evita stack overflow tempranos.

TAIL CALLS

```
function foo(num) {  
  try { return foo((num || 0) + 1) }  
  catch(e) { return num }  
}
```

```
console.log(foo())
```

» Ejercicio: Probar el código en diferentes browsers

TAIL CALLS

EJERCICIOS

1. Probar el código en firefox.
2. Probar el código en chrome.
3. Probar el código en node.

LET & CONST

“Don't grep-replace "var" with "let" or you will break the internet.”

- » Scope
- » Hoisting
- » TDZ

LET & CONST BLOCK SCOPE

```
if (true) {  
    var a = 'a'  
    let b = 'b'  
  
    console.log(b) // ✓
```

}

```
console.log(a); // ✓  
console.log(b); // ✗ ReferenceError: b is not defined
```

LET & CONST

LET VS. CONST¹

```
const foo = 'foo'  
const bizz = { foo: 'bar' }
```

```
foo = 'bar' // ✗ TypeError: Assignment to constant variable.  
bizz.foo = 'foo' // ✓
```

¹ La única diferencia es que const no se puede reasignar.

LET & CONST

EJERCICIOS

1. Usar let dentro de un if.
2. Usar let dentro de un for.
3. Crear block scopes solo con llaves.

REST PARAMETERS

```
function foo( ...bar) {  
  console.log(bar.join(''))  
}
```

```
foo( 'Estos' , 'parámetros' , 'se' , 'van' , 'a' , 'juntar' )
```

REST PARAMETERS

CONSIDERACIONES

- » Inyecta todos los parámetros de una función dentro de un Array
- » 1 por función
- » Siempre ultimo parametro
- » Se pierde el valor de arguments
- » No se puede usar con default

REST PARAMETERS

EJERCICIOS

1. Probar el código en el browser.
2. Probar el código agregando otro rest parameter.
3. Probar el código agregando otro parámetro después de ...bar.

SPREAD OPERATOR

- » Permite descomponer los elementos de un Array en valores individuales.

SPREAD OPERATOR

ASIGNAR PARAMETROS

```
function add(x, y) {  
    return x + y  
}  
  
const values = [1, 2]  
console.log(add(...values)) // 3
```

SPREAD OPERATOR

CONCATENAR ARRAYS

```
const arr1 = [1, 2]
```

```
const arr2 = [3, 4]
```

```
const arr3 = [...arr1, ...arr2]
```

```
console.log(arr3) // [1, 2, 3, 4]
```

SPREAD OPERATOR

EJERCICIOS

1. Probar ambos códigos en el browser.

DESTRUCTURING

- » Permite asignar un grupo de variable/s en base a los valores de un Object o un Array.

DESTRUCTURING OBJETOS

```
const person = { name: 'Ignacio', age: 27 }
const { name, age } = person
```

```
console.log(name, age) // Ignacio 27
```

DESTRUCTURING ARRAY

```
const date = [12, 31, 2016]
const [m, d, y] = date

console.log(m, d, y) // 12 31 2016
```

DESTRUCTURING

EJERCICIOS

1. Probar el código en el browser.
2. Probar que pasa cuando no existe la propiedad en el objeto.
3. Probar como ignorar posiciones en el array.
4. Probar como asignar alias a las variables (objetos).

ARROW FUNCTIONS

- » Sintaxis Diferente.
- » Lexical this.
- » () => {}

ARROW FUNCTION VS. FUNCIONES TRADICIONALES

```
const fn = function () { return 2 }
const fnArrow = () => { return 2 }
const fnTinyArrow = () => 2
```

ARROW FUNCTIONS

PARENTESIS

`() => {}` // sin parametros, parentesis obligatorios

`x => {}` // un parametro, parentesis opcionales

`(x) => {}` // un parametro, parentesis opcionales

`(x, y) => {}` // 2 o + parametros, parentesis obligatorios

ARROW FUNCTIONS

LLAVES

```
x => x * x // una linea, llaves y return opcionales  
x => {  
    return x * x  
} // mas de una linea, llaves y return obligatorios
```

ARROW FUNCTIONS

EJERCICIOS

1. Usar un array y un forEach con arrow function para imprimir los elementos en la consola.
2. Usar un array y un map con arrow function para crear un nuevo array de objetos.

ARROW FUNCTIONS

this BINDING

```
const Widget = {  
  init: function() {  
    document.addEventListener('click', function(e) {  
      this.do(event.type) // ✗ !?  
    })  
  },  
  
  do: function(type) {  
    console.log('Evento', type)  
  }  
}
```

Widget.init() // ✗ this.do is not a function

ARROW FUNCTIONS

ARREGLANDO this

```
const Widget = {  
  init: function() {  
    const _this = this // 🤝  
    document.addEventListener('click', function(e) {  
      _this.do(event.type) // ✅  
    })  
  },  
  
  do: function(type) {  
    console.log('Evento', type)  
  }  
}  
  
Widget.init() // ✅
```

ARROW FUNCTIONS

LEXICAL this

```
const Widget = {  
  init: function() {  
    document.addEventListener('click', (e) => {  
      this.do(event.type) // 💪  
    })  
  },  
  
  do: function(type) {  
    console.log('Evento', type)  
  }  
}  
  
Widget.init() // 🙌
```

ARROW FUNCTION

CONSIDERACIONES

- » Seguimos usando funciones normales.
- » Siempre son anónimas.
- » No puedo cambiar el valor de this.
- » `typeof () => {} // 'function'`

ARROW FUNCTION

EJERCICIOS II

1. Probar los 3 ejemplos en el browser

DEFAULT PARAMETERS

- » Permite establecer valores por defecto en los argumentos de las funciones

```
function add(x = 1, y = 2) {  
    return x + y  
}
```

```
foo() // 3
```

DEFAULT PARAMETERS

EJERCICIOS

1. Probar el ejemplo en el browser.
2. Ejecutar la función de esta manera: `foo(10)`.
3. Ejecutar la función de esta manera: `foo(10, 20)`.
4. Ejecutar la función de esta manera: `foo(0, 20)`.

CLASS

- » Syntax sugar para orientación a objetos.
- » JavaScript sigue utilizando prototypes.

CLASS SINTAXIS

```
class Person {  
    constructor (name, age) {  
        this.name = name  
        this.age = age  
    }  
  
    hi () {  
        return '👋 ' + this.name  
    }  
}  
  
const p = new Person('Ignacio', 27)  
p.hi() // 👋 Ignacio
```

CLASS HERENCIA

```
class Baby extends Person {  
    constructor (name, age) {  
        super(name, age) // llama al método de la clase padre  
    }  
  
    cry () {  
        return '😭'  
    }  
}  
  
const b = new Baby('Ana', 1)  
b.cry() // 😭
```

CLASS

EJERCICIOS

1. Probar el primer ejemplo en el browser y acceder métodos y propiedades de p.
2. Probar el segundo ejemplo en el browser y acceder métodos y propiedades de b.

TEMPLATE LITERALS

- » Mejora la concatenación de string literals con variables
- » Se utilizan los backtick ``.

```
const foo = 'foo'  
const bar = 'bar'
```

```
console.log(`This is ${foo} and this is ${bar} 😊`)
```

TEMPLATE LITERALS

EJERCICIOS

1. Probar el ejemplo en el browser.
2. Usar dentro de un template literal una llamada a una función.

OBJECT LITERAL EXTENSIONS

» Syntax sugar para manipulación de objetos.

OBJECT LITERAL EXTENSIONS

PROPIEDADES

- » Si el nombre de la propiedad es igual al nombre la variable no tengo que repetirlo

```
const name = 'Ignacio'
```

```
const age = 27
```

```
const person = { name, age }
```

OBJECT LITERAL EXTENSIONS

METODOS

- » Puedo escribir métodos sin escribir el carácter ":" y la palabra function².

```
const dog = {  
  bark () {  
    console.log('🐶')  
  }  
}
```

² Misma sintaxis que con Class

OBJECT LITERAL EXTENSIONS

EJERCICIOS

1. Probar los ejemplos en el browser.

PROMISES

- » El código async asusta .
- » Callback Hell .
- » Mejor organización .
- » Código mas mantenible .

```
1 function hell (win) {
2   // for listener purpose
3   return function () {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function () {
5       loadScript(win, REMOTE_SRC+'/lib/async.js', function () {
6         loadScript(win, REMOTE_SRC+'/lib/easyXDM.js', function () {
7           loadScript(win, REMOTE_SRC+'/lib/json2.js', function () {
8             loadScript(win, REMOTE_SRC+'/lib/underscore.min.js', function () {
9               loadScript(win, REMOTE_SRC+'/lib/backbone.min.js', function () {
0                 loadScript(win, REMOTE_SRC+'/dev/base_dev.js', function () {
1                   loadScript(win, REMOTE_SRC+'/assets/js/deps.js', function () {
2                     loadScript(win, REMOTE_SRC+'/src/'+win.loader_path+'/loader.js', function () {
3                       async.eachSeries(SCRIPTS, function (src, callback) {
4                         loadScript(win, BASE_URL+src, callback);
5                       });
6                     });
7                   });
8                 });
9               });
0             });
1           });
2         });
3       });
4     });
5   );
6 }
```

PROMISES

new Promise()

```
function doAsync() {  
  const p = new Promise(function(resolve, reject) {  
    if (/* todo sale bien */) {  
      return resolve('👍')  
    }  
  
    return reject('👎')  
  })  
  
  return p  
}
```

PROMISES

.then() & .catch()

doAsync()

```
.then(result => {  
    console.log(result)  
})  
.catch(err => {  
    console.log(err)  
})
```

PROMISES CHAIN

```
doAsync()
  .then(result => {
    return doAsync()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.log(err)
  })
}
```

PROMISES PARA QUE?

- » AJAX
- » Carga de Imágenes
- » Leer/Escribir Storage
- » Mutaciones en el DOM

PROMISES

ESTADOS

- » fulfilled 
- » rejected 
- » pending 

ES MODULES

- » Permiten separar nuestro aplicacion en diferentes archivos.
- » Mejor calidad de código .
- » No hay soporte nativo

ES MODULES

export

```
// archivo person.js
const person = { name: 'Ignacio', age: 27 }
```

```
export default person
```

ES MODULES

import

```
// archivo app.js  
import person from './person.js'
```

```
console.log(person)
```

MAS!!!



» Generators

» Proxies

» String API

» Math API

» Object API

» Array API

» Number API

» Regex

TRANSPLING

FUUUUUSION... HAI!

TOOLS



- » Babel
- » Traceur
- » TypeScript
- » es6-shim

SHIMS & POLYFILLS

- » Proveen compatibilidad con motores de JavaScript legacy.
- » La sintaxis no es shimeable.

SOPORTE



» Can I Use?

» Kangax

RECURSOS



- » 2ality
- » NCZ
- » Lukehoban
- » ES6 & Beyond
- » ES6 Learning

PREGUNTAS

