ECMAScript

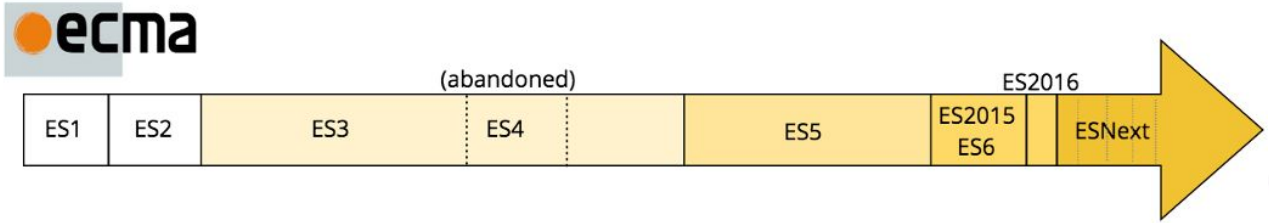
DEV.F:.
DESARROLLAMOS(PERSONAS);

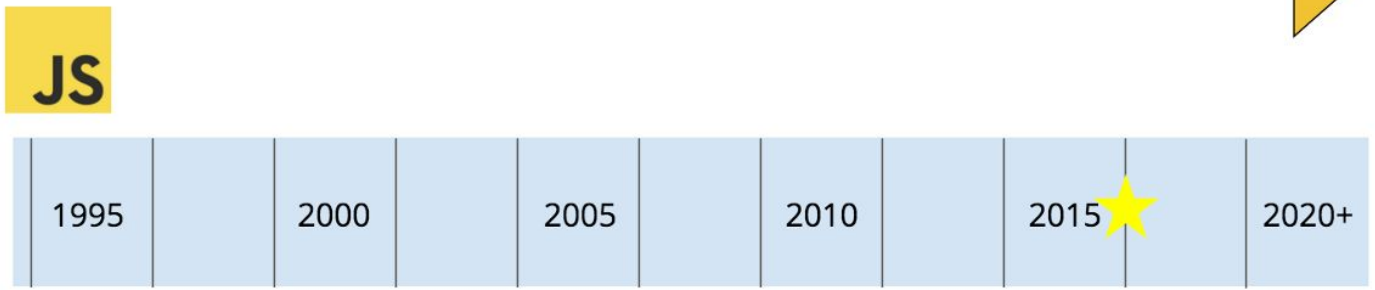
¿Qué es ECMA?

Ecma International es una organización internacional basada en membresías de estándares para la comunicación y la información. Adquirió el nombre Ecma International en 1994, cuando la European Computer Manufacturers Association (ECMA) cambió su nombre para expresar su alcance internacional.

Fue fundada en 1961 para estandarizar los sistemas informatizados en Europa.

ECMAScript


ecma
(abandoned)
ES2016
ES1
ES2
ES3
ES4
ES5
ES2015
ES6
ESNext


JS
1995
2000
2005
2010
2015
2020+


iOS

ECMAScript

Es el estándar que la empresa del mismo nombre definió para JavaScript en el año 2015 (ES6) y encarga de regir como debe ser interpretado y cómo debe funcionar el lenguaje JavaScript.

Versión	Nombre	Publicación	Guía
ES11	ES2020	Junio del 2020	-
ES10	ES2019	Junio del 2019	-
ES9	ES2018	Junio del 2018	-
ES8	ES2017	Junio del 2017	-
ES7	ES2016	Junio del 2016	-
ES6	ES2015	Junio del 2015	-
ES5.1	ES5.1	Junio del 2011	-

Características más utilizadas de ECMA

- Let y const.
- Arrow functions
- For/of y for/in.
- Map vs forEach.
- Clases.
- Promises.
- Default parameters.
- Spread operator.
- Rest operator.
- Object entries y values.
- JS Modules.
- Literal templates / Template strings.
- Map y Set. => Estructuras de datos.
- Async / await.
- Asynchronous Iteration.
- Promise Finally.
- Object Rest Properties.
- New RegExp Features (https://regexr.com/).

Scope
(Alcance)

DEV.F::
DESARROLLAMOS(PERSONAS);

Glosario

- **Scoping:** Cómo se organizan y accedemos a las variables. ¿Dónde viven? ¿Desde dónde puedo acceder a ellas?
- **Lexical Scoping:** Cuando el scoping está controlado por la ubicación de donde se declaran las funciones
- **Scope:** El Entorno en donde se declara una variable en específico.
    - Scope Global
    - Scope de Función
    - Scope de Bloque
- **Scope de una variable:** Región del código en donde se puede acceder a esa variable

Los 3 tipos de Scope

Scope Global

```
const firstName = 'Yaxche';
const job = 'sensei';
const year = 1993;
```

Fuera de cualquier función o bloque { }

Se puede acceder a las variables declaradas en el scope global desde donde queramos

Scope de función

```
function ageCalculator(birthYear) {
    const now = 2023;
    const age = now - birthYear;
    return age;
}
```

Estas variables solo son accesibles **dentro de la función**

También llamado scope local

Scope de bloque (ES06)

```
if (year ⩾ 1981 && year ⩽ 1986) {
    const millennial = true;
    const food = 'Aguacate 🥑';
}
```

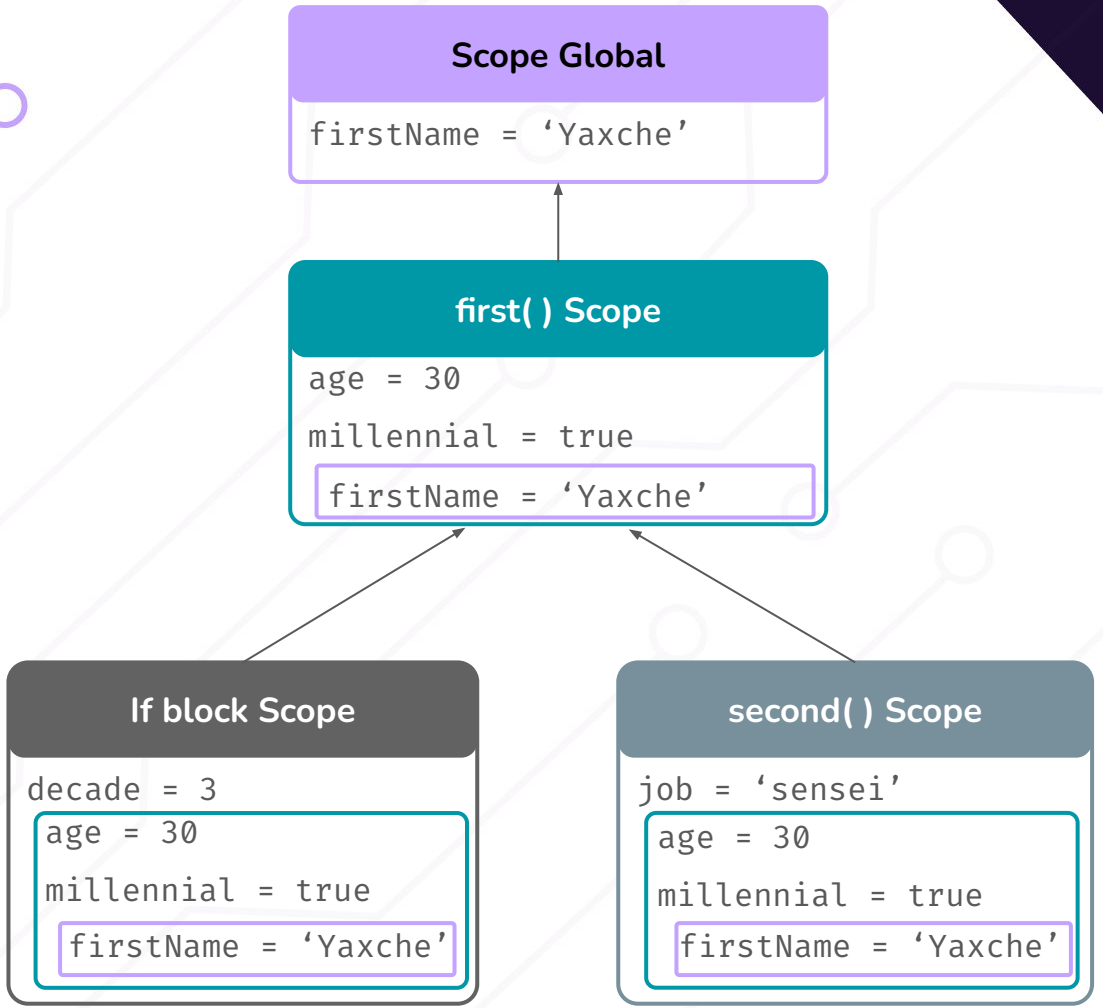
Solo se pueden acceder desde **dentro del bloque**

Solo aplica a variables declaradas con `let` y `const`.

`var` puede ser consultada fuera del bloque, se 'sale' al scope de función más cercano

Cómo se ve el scope?

```
const firstName = 'Yaxche';

function first() {
    const age = 30;

    if (age ⩾ 30) {
        const decade = 3;
        var millenial = true;
    }

    function second() {
        const job = 'sensei';

        console.log
            (`${firstName} tiene ${age} y es ${job}`);
            // Yaxche tiene 30 años y es sensei
    }

    second();
}

first();
```



`var` vs. `let` vs. `const`

	var	let	const
Stored in Global Scope	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can Be Reassigned?	✓	✓	✗
Can Be Redeclared?	✓	✗	✗
Can Be Hoisted?	✓	✗	✗

Strings

DEV.F:
DESARROLLAMOS(PERSONAS);

.endsWith() & .startsWith()

```
let saludo = 'Hola Mundo!';

console.log(salud.substr(0,1) === 'H');
// true

saludo.startsWith('Hola');
// true
saludo.startsWith('H');
// true
saludo.endsWith('Mundo!');
// true
saludo.endsWith('!');
// true

saludo.includes('a');
// true

saludo.startsWith('H', 5);
// false
// empieza a buscar despues de la posición 5
saludo.includes('H', 5);
// false
// empieza a buscar despues de la posición 5
```

.repeat(), .padStart() & .padEnd()

```
const mood = 'Happy! ';

console.log(`I feel ${mood.repeat(3)}`);
```

```
const str1 = 'Hello World';

console.log(str1.padEnd(25, '!'));
// Expected output: "Hello World!!!!!!!!!!!!!"
```

```
const fullNumber = '2034399002125581';
const last4Digits = fullNumber.slice(-4);
const maskedNumber = last4Digits.padStart(fullNumber.length, '*');

console.log(maskedNumber);
// Expected output: "************5581"
```

Template Literals | Template Strings

Podemos usar ‘backticks’ ``` para crear un valor de tipo string.

En qué nos ayudan?

Podemos imprimir los valores de variables sin tener que concatenar múltiples valores (string + números).

Dentro de ${ } podemos escribir expresiones

```
1  const firstName = 'Yaxche';
2
3  const greeting1 = 'Hola ' + firstName+ ', como estás?'
4  console.log(greeting1);
```

```
1  const firstName = 'Yaxche';
2
3  const greeting2 = `Hola, ${firstName}, como estás?`;
4  console.log(greeting2);
```

Parámetros por defecto

DEV.F::
DESARROLLAMOS(PERSONAS);

Parámetro Rest

DEV.F:
DESARROLLAMOS(PERSONAS);

Parámetro Rest

El parámetro Rest es indicado con tres puntos (...) seguido del nombre que le asignaremos a dicho parámetro.

Ese parámetro se convierte en un arreglo/objeto que contiene el “resto” de los parámetros pasados a la función.

De ahí el nombre REST

Restricciones del Parámetro Rest

1. Sólo puede existir un parámetro rest en la función
2. El parámetro rest debe de ir siempre como último parámetro

Destructuración de Objetos

DEV.F:
DESARROLLAMOS(PERSONAS);

Destructuración de Objetos

```
let user = {
    fullName: 'Yaxche Manrique',
    email: 'yaxchemanrique1@gmail.com',
    googleAccount: '214c4443-f959-4403-842c-1c24703e0936',
    isPremium: true,
};
```

```
let fullName = user.fullName,
    email = user.email,
    googleAccount = user.googleAccount;
```

DEV.F

Destructuración de Objetos

```
let { fullName, email, googleAccount } = user;
```

```
let { isPremium: payingClient } = user;
console.log(payingClient);
```

```
let { instagram: insta = '@yaxchemanrique' } = user;
console.log(insta);
```

Destructuración de Arreglos

DEV.F::
DESARROLLAMOS(PERSONAS);

Destructuración de Arreglos

```
let frutas = ['manzana', 'pera', 'fresa'];
let [fruta1, fruta2] = frutas;

console.log(fruta1); // manzana
console.log(fruta2); // pera
```

```
let frutas = ['manzana', 'pera', 'fresa'];
let [, , fruta3] = frutas;
console.log(fruta3); // fresa
```

Arrays

- Map y forEach(diferencias).
- Reduce.
- Push.
- Splice.
- Includes.
- Filter.
- Find.
- FindIndex.

```
[1, 2, 3].push(4) // [1,2,3,4]

[1, 2, 3].pop() // [1,2]

[1, 2, 3].shift() // [2,3]

[1, 2, 3].unshift(0) // [0,1,2,3]

['a', 'b'].concat('c') // ['a','b','c']

['a', 'b', 'c'].join('-') // a-b-c

['a', 'b', 'c'].slice(1) // ['a','b]

['a', 'b', 'c'].indexOf('b') // 1

;['a', 'b', 'c'].includes('c') // true

[3, 5, 6, 8].find((n) => n % 2 === 0) // 6

[2, 4, 3, 5].findIndex((n) => n % 2 !== 0) // 2

[3, 4, 8, 6].map((n) => n * 2) // [6,8,16,12]

[1, 4, 7, 8].filter((n) => n % 2 === 0) // [4,8]

[2, 4, 3, 7].reduce((acc, cur) => acc + cur) // 16

[2, 3, 4, 5].every((x) => x < 6) // true

[3, 5, 6, 8].some((n) => n > 6) // true

[1, 2, 3, 4].reverse() // [4,3,2,1]

[3, 5, 7, 8].at(-2) // 7
```

DEV.F

Funciones flecha

DEV.F:
DESARROLLAMOS(PERSONAS);

forOf

DEV.F::
DESARROLLAMOS(PERSONAS);

map

DEV.F::
DESARROLLAMOS(PERSONAS);

forOf

DEV.F::
DESARROLLAMOS(PERSONAS);

forIn

DEV.F::
DESARROLLAMOS(PERSONAS);

Bonus

- Paso por referencia y valor.
- Tipos de salida de una función.
- Valor por defecto de una variable (||).
- Encadenamiento opcional (cortocircuito ?).
- Nullish operator (??).