

# ES6 Javascript



Para Aplicaciones Web  
Modernas

Fernando Saez  
saezfernando@gmail.com

# JAVASCRIPT

## 1- Vanilla JS (No Frameworks!)

- Datatypes
- Functions
- Conditionals
- Loops

## 2- DOM Manipulation & Events

## 3- Fetch API & JSON

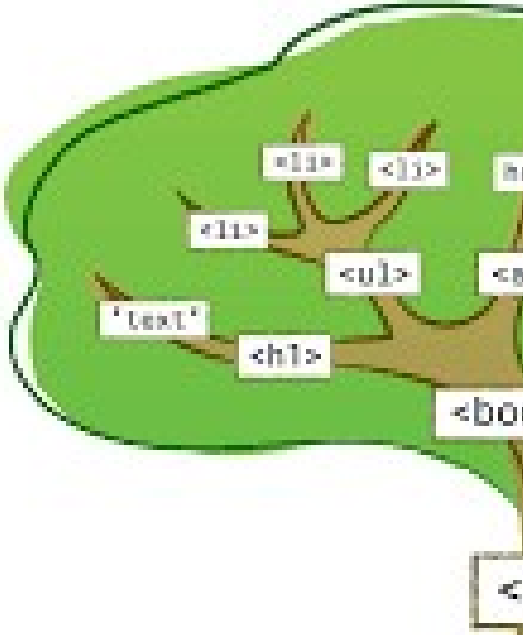
## 4- ES6+ Features

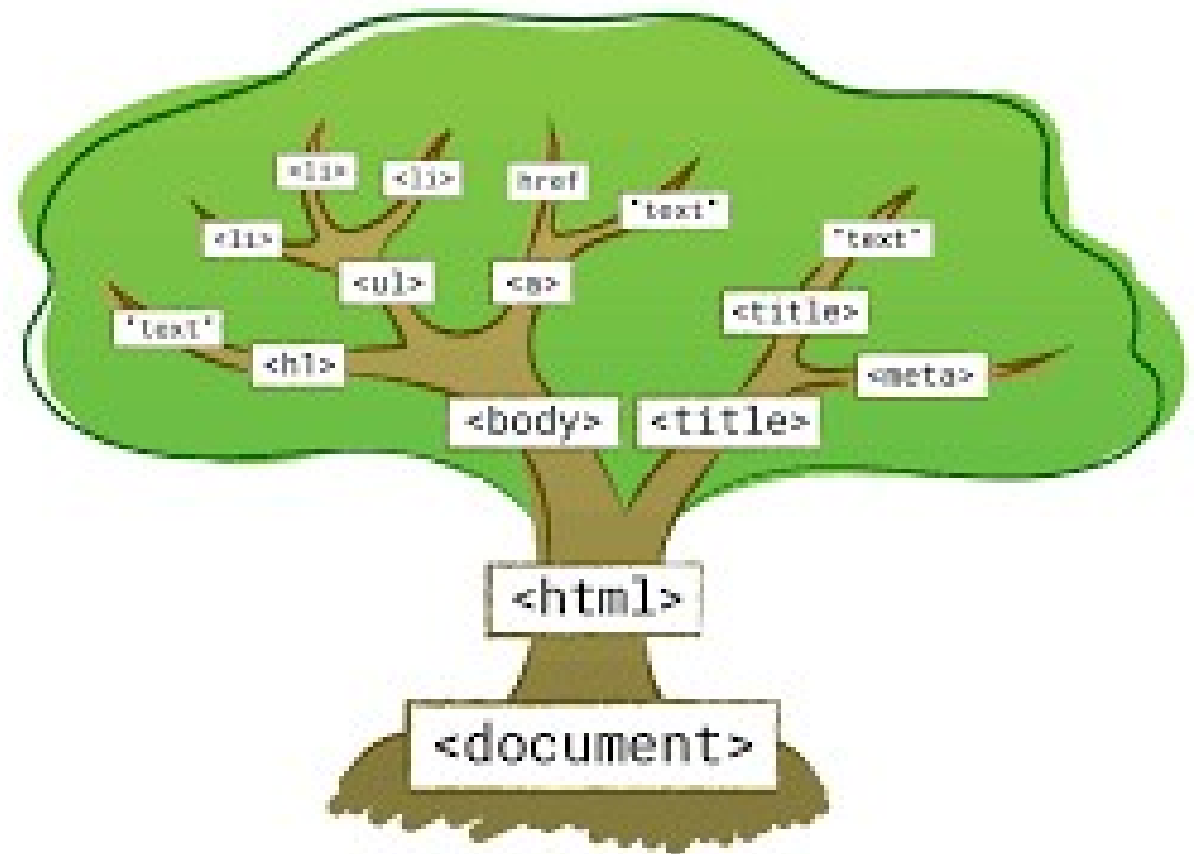
- Arrow Functions
- Promises
- Async/Await
- Destructuring
- Template String

A yellow square containing the letters 'JS' in a bold, black, sans-serif font.

JS

# Client-Side JavaScript

- JavaScript en el browser
  - DOM (Objetos window y document)
  - API HTML 5
  - JQuery
  - React, Vue, Angular
- 





# Server-Side JavaScript

- Node y Standalone JavaScript
- Popular en Cliente y en Servidor
- Rasgos del lenguaje adicionales como un Nuevo entorno de ejecución.
- ES6 (Construido sobre el motor de javascript V8 de Google)
- Soporte para módulos.
- Muchas librerías



# Como mantenerse informado del ecosistema de javascript

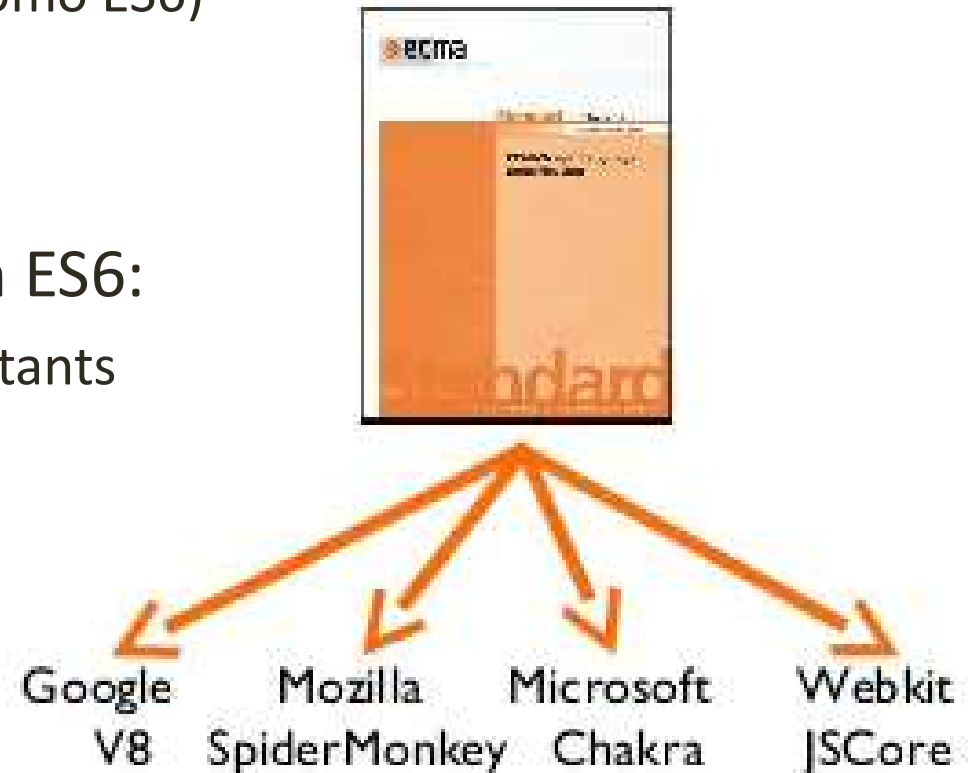
- MDN (Documentación de tecnologías Web: HTML, CSS, JS y otros)
- Listas Awesome (<https://github.com/sorrycc/awesome-javascript>)
- Comenza a seguir a desarrolladores expertos (twitter, github)
- Medium
- Dev.To
- Blogs personales
- Blogs de desarrollo
- Echojs
- Reddit
- Charlas o conferencias por youtube
- Participa en comunidades
- Canales de Youtube

# ECMAScript

- Especificación de lenguaje de script para el navegador
  - ECMA International es la organización responsable del estandard.
  - Versiones: ES3, ES5, ES2015 (conocida como ES6)
  - Última version ES10

- Algunas características agregadas en ES6:
  - Ámbitos de bloque para variables y constants
  - Funciones flechas
  - Rasgos orientados a objetos. Ej. Classes
  - Soporte para módulos

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



JavaScript Implementations

# ES5 vs. ES6

- ECMAScript 5 no agregó ninguna sintaxis nueva
- ¡ECMAScript 6 sí lo hace!

```
[1,2,3,4,5].map(n => n * 2);
```

```
[1,2,3,4,5].map(function(n) {  
    return n * 2;  
});
```

```
let {name, surname} = trainer;
```

```
var name = trainer.name;  
var surname = trainer.surname;
```

```
setInterval(() => age++, 1000);
```

```
setInterval(function () {  
    return age++;  
}, 1000);
```

# Transpilers



Dart

*BABEL*

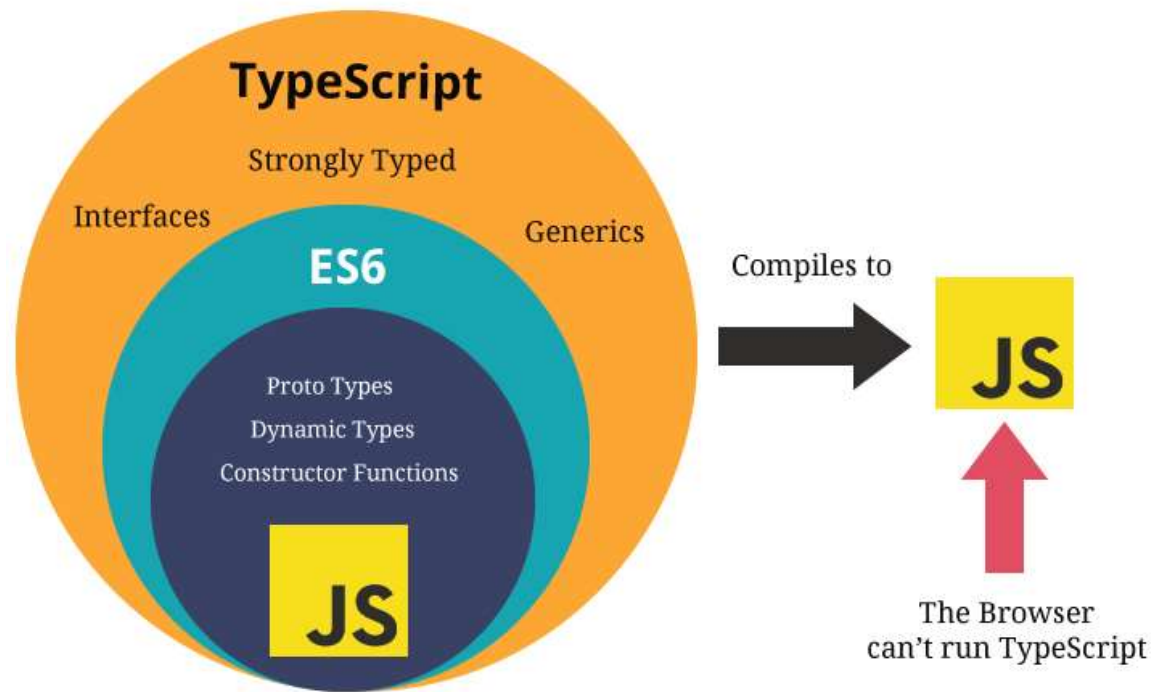
TypeScript

- **Compiladores** traducen código de un lenguaje a otro.
  - Ej. Java to bytecode
- **Transpiladores** traducen código al mismo lenguaje. Hay varios transpilers que traducen código ES6 a ES5



# TypeScript

- JavaScript con tipos
- Hace JavaScript mas deseable para grandes proyectos de software.
  - Chequeo de tipos estático, soporte a herramientas mejorada.
- ES6 y más (compilado a JavaScript plano)
- Usado en aplicaciones [Angular](#) (React y Vue)



# TypeScript: Ejemplo básico

- Para instalarlo es necesario node.js y npm.

`npm install -g typescript`

- Los archivos serán programados en typescript y los fuentes con extensión .ts
- `tsc <nombreArchivo>.ts`

```
let name: string = `Javier Ruiz`;
```

```
let age: number = 28;
```

```
let sentence: string = `Hola, mi nombre es ${ name }.
```

```
Este año voy a cumplir ${ age + 1 } años.`;
```

```
// Esto sería equivalente a
```

```
let sentence: string = "Hola, mi nombre es " + name + ".\n\n" + "Este año voy a  
cumplir " + (age + 1) + " años.";
```

# Linters y formateadores de código

- Permiten mantener la coherencia del código.
- Permiten definir conjuntos de reglas y luego comprueban el código a medida que se escribe de modo que no nos podamos salir del camino marcado.

- ESLint



- JSHint



- Prettier



# Task Runners

- **Gulp**: que se basa en código JavaScript que escribes en un archivo para describir las diferentes tareas y enlazarlas.
- **Grunt**: otro ejecutor de tareas de código abierto y basado en JavaScript, aunque en este caso prima la configuración frente al código a la hora de definir las tareas.
- **npm**: sí, otra vez. El gestor de paquetes es también una excelente herramienta de ejecución de tareas puesto que nos permite crear pequeños scripts que se ejecutan con el comando `npm run`



# Empaquetadores de módulos (Bundlers)

- Permiten incluir en un solo archivos múltiples recursos necesarios para una aplicación Web.

- Webpack



webpack

- Browserify



PARCEL

Blazing fast, zero configuration web api



- Parcel

- Rollup



rollup.js

# The Strict Mode

- Cambia algunos comportamientos por defecto de javascript para facilitar el descubrimiento de errores, optimizar la performance, y migrar a futuras versiones de ES

```
// antes de cualquier sentencia  
use strict;  
//or 'use strict';
```

**'use strict';**

**JS**

- Elimina algunos errores silenciosos de JavaScript.
- Corrige errores que hacen difícil para los motores de JavaScript realizar optimizaciones.
- Prohíbe cierta sintaxis

# ¿CONOCES LA SINTAXIS DE JAVASCRIPT?

```
1 let language = 'JavaScript'
2 let company = {
3   name: 'EDteam',
4   slogan: 'Nunca te detengas',
5   founded: 2015
6 }
7 console.log(company.name)
8 // 'EDteam'
9 const getMajorNumber = (a,b) => {
10   if (a > b) { return a }
11   else { return b }
12 }
13 getMajorNumber(4,6)
14 // 6
```

JS

Las variables se declaran con **let** (no hay que indicar el tipo de dato)

Los objetos encierran entre llaves parejas con el formato **propiedad: valor** separadas por comas.

**console.log( )** imprime en consola la expresión entre los paréntesis.

Para obtener el valor de una propiedad de un objeto se usa **objeto.propiedad**

Condicional (**if** / **else**)

Comentarios (**líneas 8 y 14**)

Ejecución de la **función**

Definición de función (se recomienda usar constantes con **const**)



# Javascript Data Types

## Primitive

**Boolean**  
**Null**  
**Undefined**  
**Number**  
**String**  
**Symbol**

**IMMUTABLES**

## Object

**Array**  
**Object**  
**Function**  
**Date**  
**Regex**  
**...**

**MUTABLES**



# Inmutabilidad

// El uso de un método de cadena no modifica la cadena

```
var bar = "baz";  
console.log(bar);      // baz  
bar.toUpperCase();  
console.log(bar);      // baz
```

// El uso de un método de arreglo muta el arreglo

```
var foo = [];  
console.log(foo);      // []  
foo.push("gato");  
console.log(foo);      // ["gato"]
```

// La asignación le da al primitivo un nuevo valor (no lo muta)

```
bar = bar.toUpperCase(); // BAZ
```

# Literales

- Boolean: `true, false`
- Number: `123, 4.56`
- String: `"hello", 'world'`
- Null and Undefined: `null, undefined`
- Template literal
- Object literal: `{nombre: "Juan", madre: "Maria"}`

# Valores (Truthy, Falsy)

Usar cualquiera de los siguientes valores con un operador booleano, o en un bloque condicional coercionara a falso:

- 0 (cero)
- "" (string vacio)
- null
- undefined
- NaN
- false



**CAN'T BE COERCED**

**IF YOU USE TRIPLE EQUALS**

# Variables y Constantes



```
var x;
```



```
let x;  
x = 'abc';
```

```
const y = 20;
```

# 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?			

# Scope

## Global vs function vs block

```
var a = 5;  
var b = 10;
```

```
if (a === 5) {  
  let a = 4; // El alcance es dentro del bloque if  
  var b = 1; // El alcance es global
```

```
  console.log(a); // 4  
  console.log(b); // 1  
}
```

```
console.log(a); // 5  
console.log(b); // 1
```

```
a = 10; // global scope  
var b = 20; // function scope  
let c = 30; // block scope  
const d = 40; // block scope
```

- Tip: Evite usar `var` o variables globales



```
> function showName() {  
    var name = "GeeksforGeeks";  
}  
showname()  
console.log(name);
```

✖ ▶ Uncaught ReferenceError: showname is not defined  
at <anonymous>:4:1

> |

```
const message = 'Hi from Linuxhint';  
message = 'Hello from Linuxhint'; // TypeError
```

✖ ▶ Uncaught TypeError: ..html:6  
Assignment to constant variable.  
at ..html:6

> |

```
const message = 'Hi from Linuxhint';  
const message = 'Hello from Linuxhint'; // SyntaxError
```

✖ Uncaught SyntaxError: ..html:6  
Identifier 'message' has already been  
declared

>



# Hoisting

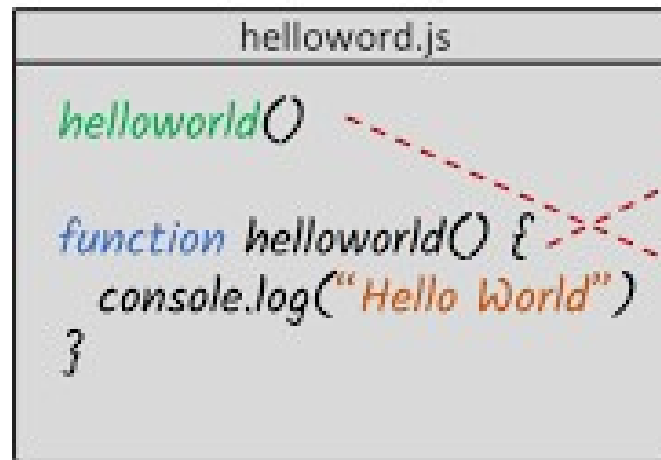
```
add();
```

```
function add() {
```

```
  var myNumber = 4;
```

```
  console.log(myNumber + myNumber);
```

```
}
```



```
function foo() {
```

```
  console.log(x); //retorna undefined
```

```
  var x=10;
```

```
}
```

Equivalente

```
function foo() {
```

```
  var x;
```

```
  console.log(x);
```

```
  x=10;
```

```
}
```

### String declaration

```
const name = "Proful";  
const city = 'Jeypore';  
const msg = `${name} is  
            from ${city}`;
```

### String functions

```
msg.indexOf(city); //15  
msg.lastIndexOf(name); //0  
city.charAt(2); //'y'  
city[3]; //'y'
```

# STRING CHEATSHEET

### String functions

```
city.replace("J", "P");  
city.toUpperCase();  
name.toLowerCase();  
name.concat(" is good")
```

### String functions

```
name.slice(3, 6); //'ful'  
city.split("y");  
           //[ 'Je', 'pore' ]  
name.length; //6
```

# Template Literal

```
let a = 10;
```

```
let b = 20;
```

```
console.log(` ${a}+${b} is ${a+b} `);
```

marcadores

Template literal

Cualquier Expression de javascript

```
console.log(`string text line 1  
string text line 2`);
```

```
// "string text line 1
```

```
// string text line 2"
```

# Objeto Literal

```
Const automovil =  
{  
  make: 'Honda',  
  model: "Civic",  
  "Year": 2001,  
  'owner': {  
    name: "Jorge"  
  }  
}
```

- Un objeto literal consiste de cero o más pares `key:value` llamados *propiedades*
- *Los valores pueden incluir arrays, objetos o funciones*

## Object Declaration

```
const twit = {  
  name: "Proful",  
  follower: 4817,  
  1 : "hi"  
}
```

any type

converted to string

- storing key-value pairs.
- data unordered
- keys are unique

# Propiedades del Objeto

Identificador literal válido


`console.log( obj.make );`



`console.log( obj["make"] );`

Literal String o Number

`Mivar = 'model';`



`console.log( obj[mivar] );`

`obj[0] = 10;` —————> Propiedades pueden ser agregadas dinámicamente.

# Literales Object

```
var persona = {  
  nombreCompleto: ['Bob', 'Smith'],  
  edad: 32,  
  genero: 'masculino',  
  intereses: ['música', 'esquí'],  
  bio: function () {  
    alert(this.nombreCompleto[0] + ' ' + this.nombreCompleto[1]  
+ ' tiene ' + this.edad + ' años. Le gusta ' + this.intereses[0] + ' y  
' + this.intereses[1] + '.');  
  },  
  saludo: function() {  
    alert('Hola, Soy ' + this.nombre[0] + '. ');  
  }  
};
```

```
nombreCompleto : {  
  nombre: 'Bob',  
  apellido: 'Smith'  
},
```

```
var nombre = persona.nombreCompleto[0];  
var edad = persona['edad'];  
persona.bio();
```

//Notación de puntos  
//Notación de corchetes

# 2 formas de crear objetos

```
function maker(nombre, dni, grado, nivel) {  
  var it = {};  
  it.nombre = nombre;  
  it['dni'] = dni;  
  it.grado = grado;  
  it.nivel = nivel;  
  return it;  
}  
myObject = maker("Jose Perez", '32887533', 'A', 3);
```

```
function Person(name, age, sex) {  
  this.name = name;  
  this.age = age;  
  this.sex = sex;  
}  
var rand = new Person('Rand McKinnon', 33, 'M');  
var ken = new Person('Ken Jones', 39, 'M');
```

# Recorrer Objetos

```
let user = {  
  name: "Jose",  
  age: 30,  
  isAdmin: true  
};
```

```
for (let key in user) {  
  // keys  
  alert( key ); // name, age, isAdmin  
  // values for the keys  
  alert( user[key] ); // Jose, 30, true  
}
```



## Object Declaration

```
const twit = {  
  name: "Proful",  
  follower: 4817,  
  1: "hi"  
}
```

*any type* (arrow pointing to "Proful")  
*converted to string* (arrow pointing to "hi")

- storing key-value pairs.
- data unordered
- keys are unique

## Dot Notation

```
twit.name // "Proful"  
twit.follower // 4817  
twit.follower.count  
Accessing nested props
```

## Square Notation

```
twit['name'] // "Proful"
```

Can be dynamic/variable

```
const {name, followers} = twit  
name // Proful  
followers // 4800
```

```
const linkedin = { name }  
{ name: 'Proful' }
```

## Empty Object creations

```
const person = {}  
const person = new Object()
```

```
const twit = {  
  name: "Proful"  
}
```

```
function change(inst){  
  insta.name = "Steve"  
}
```

```
change(twit)  
twit.name // "Steve"
```

- Pass by reference

# JAVASCRIPT OBJECT CHEATSHEET

```
delete twit.name // both key & value
```

```
twit.randomKey // undefined
```

```
twit.follower = 5000
```

*declared as const but mutable* (arrow pointing to the assignment)

```
for(const key in twit) {  
  console.log(key) // name  
                  // followers  
}
```

```
const twit = {  
  name: "Proful",  
  get profile() {  
    return `Hi ${this.name.toLowerCase()}`  
  },  
  set profile(prof) {  
    this.name = "Mr " + prof  
  }  
}
```

*getter* (arrow pointing to get profile)  
*setter* (arrow pointing to set profile)

```
twit.profile // 'Hi proful'  
twit.profile = 'Steve'  
twit.name // 'Hi Steve'
```

```
const twit = {  
  name: "Proful",  
  hi() {  
    console.log(`Hi ${this.name}`)  
  },  
  hello: () => {  
    console.log(`Hello ${twit.name}`)  
  },  
}
```

*this refer to twit object* (arrow pointing to this in hi)  
*You cannot use this here* (arrow pointing to twit in hello)

# JSON (JavaScript Object Notation)

- Usado como un format para intercambiar datos.

```
"title": "Agenda",  
"type": "object",  
"amigos": {  
  "nombre": {  
    "description": "Nombre",  
    "type": "string"  
  },  
  "apellidos": {  
    "description": "Apellidos",  
    "type": "string"  
  },  
  "telefono": {  
    "description": "Teléfono",  
    "type": "number"  
  }  
}
```



```
const user = {  
  firstName: "John",  
  lastName: "Smith",  
  age: 25  
}  
  
// storing data in local storage  
localStorage.setItem("userData", JSON.stringify(user));  
  
// retrieve data from local storage  
const userData = JSON.parse(localStorage.getItem('userData'))
```

- Requiere usar comillas dobles para las cadenas y los nombres de propiedades.
- Las comillas simples no son válidas.

# Array

```
a = ["x", 99, true];  
a.b = "hello";  
  
a[100] = 10;
```

→

```
{  
  0: "x",  
  1: 99,  
  2: true  
}
```

- Un array es un objeto especial donde los elementos son almacenados como propiedades de un objeto
- Array tiene propiedades y métodos predefinidos **length**, **keys()**

# Array - Funciones

```
[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
```



```
// Object.values() and Object.keys() example
var languageInfo = {
  name: "JavaScript",
  founder: "Brendan Eich",
  foundedYear: 1995
}

// To get all the values of an Object in an array
Object.values(languageInfo)
// ['JavaScript', 'Brendan Eich', 1995]

// And to get only keys from an object
Object.keys(languageInfo)
// ['name', 'founder', 'foundedYear']
```



# Operadores

- Todos los operadores, ej. +, -, =, & & ...
- Igualdad estricta : **==**, **!=**
- Operadores de tipo: **typeof**, **instanceof**  
    typeof('Jon') // string  
    typeof 23     // number  
    color1=new String("verde")  
    color1 instanceof String // devuelve verdadero (true)  
    color2="coral"  
    color2 instanceof String // devuelve falso (color2 no es un objeto)
- Operadores de propiedad: **in**, **delete**

# Conversión automática de tipos

2 + 4/2	
2 + 3/2	
"2" + 3/2	
3/2 + "2"	
3/2 * "2"	
3/2 + "two"	
3/2 * "two"	

0 == false	
"" == false	
0 == ""	
null == false	
undefined == false	
! null == true	
! undefined == true	

[https://www.w3schools.com/js/js\\_type\\_conversion.asp](https://www.w3schools.com/js/js_type_conversion.asp)

# Funciones como First-class Citizens

- En JavaScript, las funciones son **objetos**
  - Pueden asignarse a variables
    - Asignadas como una propiedad de un objeto
    - *Function literals (function expressions, anonymous functions)*
  - Pueden ser pasadas como argumento a otra función.
  - Pueden ser retornadas como resultados de una función



# Function Examples

1

```
function foo() {  
    alert("foo");  
}
```

Declaración de una  
función regular

2

```
bar = function() {  
    alert("bar");  
};
```

- Function literal
- Asignación

```
setTimeout( bar, 5000 );
```

Function como  
parámetro

```
setTimeout( function() {  
    return bar;},  
5000 )
```

Function literal  
Como parámetro

3

```
a => a + 100;
```

Function flecha

# Ejercicio

- Cree una función anónima
- Retorne cualquier valor string
- Asigne la función a una variable
- Use la variable para imprimir el valor

# Ejercicio 2

```
function add(a, b) {  
    return a + b;  
}
```

1

```
let sum = add;
```

2

```
function average(a, b, fn) {  
    return fn(a, b) / 2;  
}
```

3

```
let result = sum(10,20);
```

```
console.log(result) //que imprime aquí
```

4

```
let result = average(10, 20, sum);
```

```
console.log(result) //que imprime aquí
```

5

# Ejercicio 3

```
function compararPor(nombreProp) {
```

```
    return function (a, b) {
```

```
        let x = a[nombreProp],
```

```
        y = b[nombreProp];
```

```
        if (x > y) { return 1; }
```

```
        else if (x < y) { return -1 }
```

```
        else { return 0; }
```

```
    }
```

```
}
```

```
let productos = [
```

```
    {nombre: 'iPhone', precio: 900},
```

```
    {nombre: 'Galaxy S10', precio: 850},
```

```
    {nombre: 'Sony Xperia', precio: 700}
```

```
];
```

# Arrow Functions

- *lambda expressions, lambdas*
- Una forma mas concisa de escribir una function literal

`() => 'Aprobado'`

`(a, b) => a*b`

```
function(a) {  
  return a*2  
}
```



```
(a) => {return a*2}
```



```
a => a*2
```

# Argumentos de funciones

```
function add(x,y) {  
    return x+y;  
}
```

```
add(10,20);  
add("10","20");  
add(10);  
add(10,20,30);
```

- Una variable especial `arguments` mantiene todos los argumentos pasados a la función
- `arguments` no es un array pero es similar.

```
arguments.length,  
arguments[0],  
arguments[1],...
```

# Parámetros por defecto

// ES6

```
function saludar(nombre, genero = 'Sr.', saludo = 'Hola ' +  
genero){
```

```
    console.log(saludo + ' ' + nombre);
```

```
};
```

```
saludar('Peter'); // Hola Sr. Peter
```

```
saludar('Alex', undefined, 'Que tal'); //Que tal Alex
```



# Desestructuración en iterables

1

```
const array = [1, 2, 3]
```

```
const [a, b, c] = array //desestructura el array en variables
```

```
console.log(a, b, c) // 1, 2, 3
```

2

```
const obj = {
```

```
    primerNombre: 'Jose',
```

```
    Color: 'Azul'
```

```
}
```

```
const {primerNombre, Color} = obj
```

```
console.log(primerNombre,Color) // 'Jose', 'Azul'
```

# Beneficio de desestructuración




- 1 

```
const note = {  
  id: 1,  
  title: 'My first note',  
  date: '01/01/1970',  
}
```
- 2 

```
// Crea variables desde las propiedades del objeto  
const id = note.id  
const title = note.title  
const date = note.date
```
- 3 

```
// desestructura propiedades en variables  
const { id, title, date } = note
```

# Operador spread (propagador)

- Permite que una expresión sea expandida en situaciones donde se esperan múltiples argumentos (llamadas a funciones) o múltiples elementos (arrays literales).
- Arrays literales:            [...iterableObj, 4, 5, 6]
- Llamadas a funciones            myfunction(...iterableObj);
- Desestructuración:            [a, b, ...iterableObj] = [1, 2, 3, 4, 5];

# Operador spread (propagador)

//un array literal mas poderoso

```
var partes = ['hombros', 'rodillas'];
```

```
var todo = ['cabeza', ...partes, 'cadera', 'pies'];
```

//combinar 2 arreglos en una nueva estructura

```
var arr1 = [0, 1, 2];
```

```
var arr2 = [3, 4, 5];
```

```
var newarray = [...arr1, ...arr2]; // [0,1,2,3,4,5]
```

# Propagación – Copias de objetos

1 // Array de usuarios

```
const usuarios = [  
  { id: 1, nombre: 'Ben' },  
  { id: 2, nombre: 'Ana' },  
]
```

2 // agregamos un nuevo usuario

```
const newUser = { id: 3, nombre: 'Ron' }  
users.push(newUser)
```

3 const updatedUsers = [...users, newUser]

# Propagación – conversión a array

1

// Create a set

```
const set = new Set()  
set.add('octopus')  
set.add('starfish')  
set.add('fish')
```

2

// Convertir Set a Array

```
const seaCreatures = [...set]  
console.log(seaCreatures) // ["octopus", "starfish", "fish"]
```

3

const string = 'hello'

```
const stringArray = [...string] // ["h", "e", "l", "l", "o"]
```

# Propagación con objetos

1 // Create an object and a copied object with spread

```
const originalObject = { enabled: true, darkMode: false }
```

```
const secondObject = { ...originalObject }
```

```
console.log(secondObject) // {enabled: true, darkMode: false}
```

2 const usuario = {

```
  id: 3,
```

```
  nombre: 'Ron',
```

```
}
```

```
const usuarioUpdate = { ...user, isLoggedIn: true }
```

```
console.log(usuarioUpdate) // {id: 3, nombre: "Ron", isLoggedIn: true}
```



# Parámetro REST

Los parámetros Rest nos proporcionan una manera de pasar un conjunto indeterminado de argumentos que el operador agrupa en forma de Array.

// ES6

```
function printName(name, ...fancyNames){  
    var fullName = name;  
    fancyNames.forEach(fancyN => fullName += ' ' + fancyN);  
    console.log(fullName);  
};  
printName('Felipe'); // Felipe  
printName('Felipe', 'Juan', 'Froilan'); //Felipe Juan Froilan
```

## Rest parameters

*When using rest arguments, you are collapsing all remaining arguments of a function into one array*

```
function sum( first, ...others ) {  
  for ( var i = 0; i < others.length; i++ )  
    first += others[i];  
  return first;  
}  
console.log(sum(1,2,3,4)) // output => 10;
```

*Rest parameters have to be at the last argument. This is because it collects all remaining/ excess arguments into an array*

```
let [c, ...rest] = [1,2,3,4,5]; // rest -> [2,3,4,5]
```



*Here ...rest is a collector, it collects the rest of the parameters*

**ALGUNA PREGUNTA**

**CONSULTEN GOOGLE**