



El futuro digital  
es de todos

MinTIC

# INTRODUCCIÓN

Universidad  
Industrial de  
Santander



Mision  
TIC2022

# BIENVENIDOS

# DESARROLLO DE

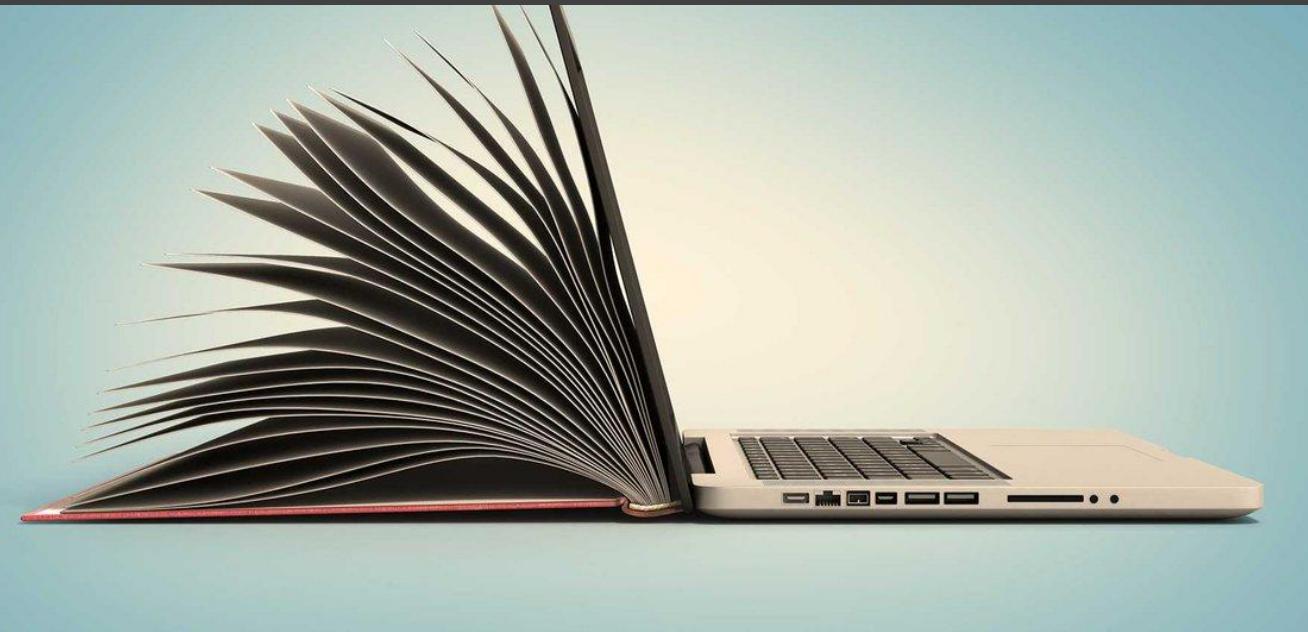
# SOFTWARE

Tatiana Albarracín Serrano

Ingeniera de sistemas – Desarrolladora en *Mercado Libre*

[misiontic.formador54@uis.edu.co](mailto:misiontic.formador54@uis.edu.co)





## INTRODUCCIÓN

- Justificación
- Objetivos
- Que vamos a hacer - tecnologías
- Moodle - grupos

Universidad  
Industrial de  
Santander



Misión  
TIC 2022



# CONTENIDO

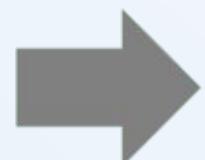
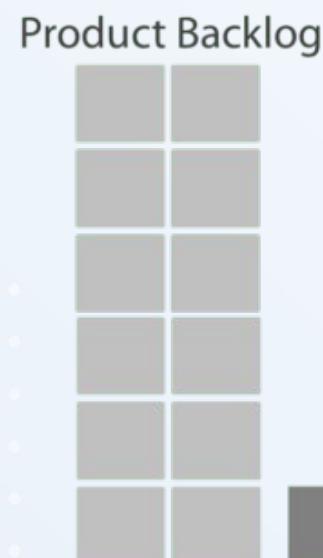
- Introducción a SCRUM
- Introducción Desarrollo Web
- JavaScript
- Manejo de DOM (Document Object Model)
- TypeScript

# Introducción a SCRUM

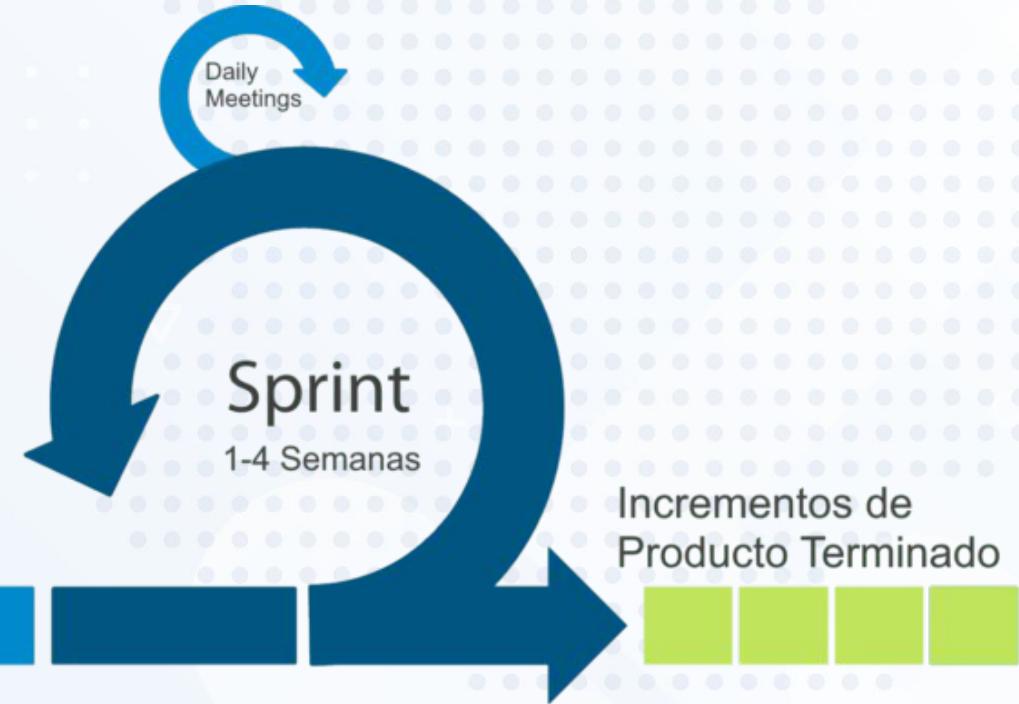
Se recomienda para proyectos complejos, con incertidumbre, entornos cambiantes con múltiples integrantes.

Se enfoca en:

- Comunicación
- Tiempo
- Entregables



Sprint Backlog



# SPRINT

Es el nombre que recibe cada uno de los ciclos o iteraciones que vamos a tener dentro de un proyecto de SCRUM.

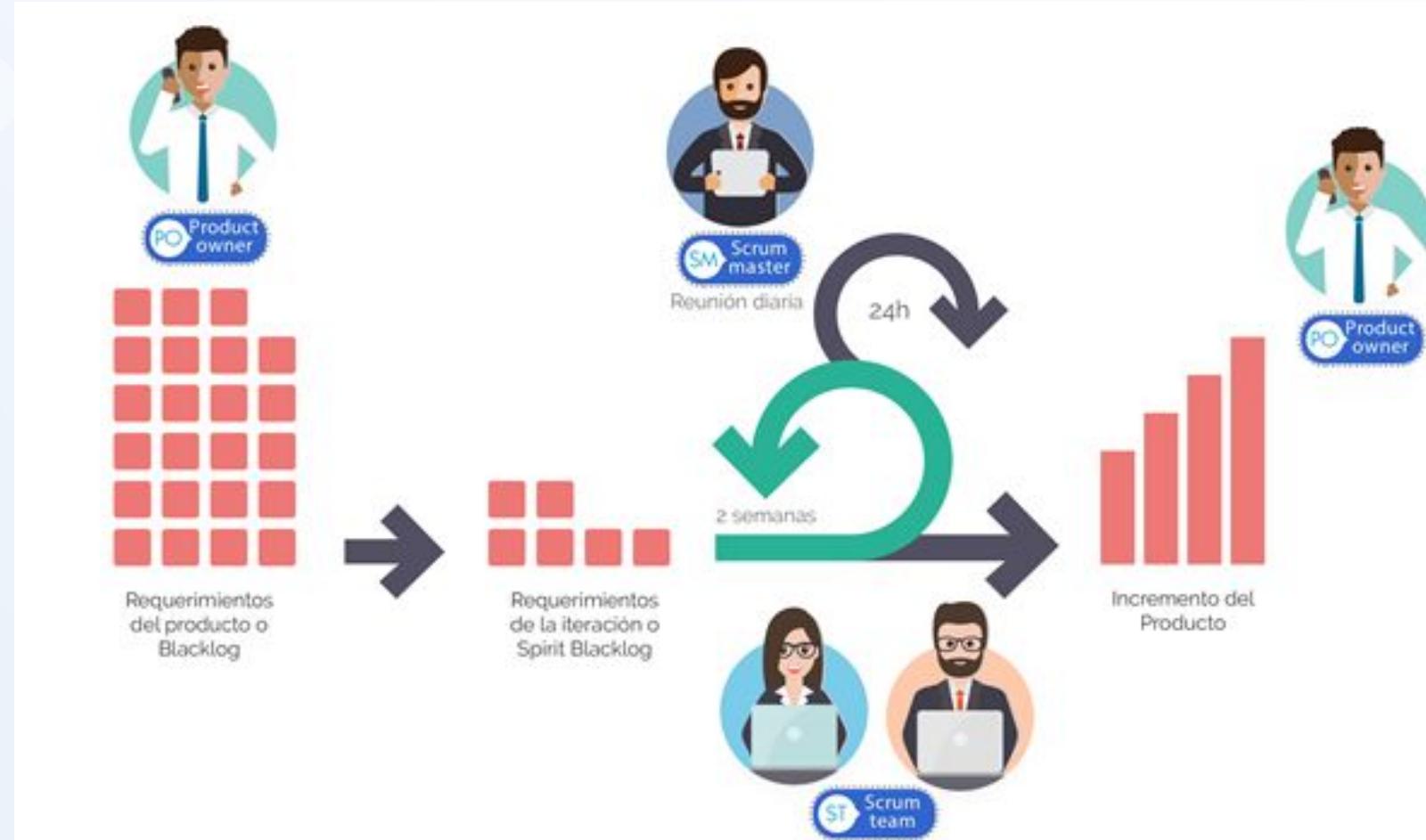
Se caracterizan por:

- Duración de 2-4 semanas en cada iteración.
- Reuniones de seguimiento diarias
- Dentro de cada sprint se gestiona la evolución del proyecto.

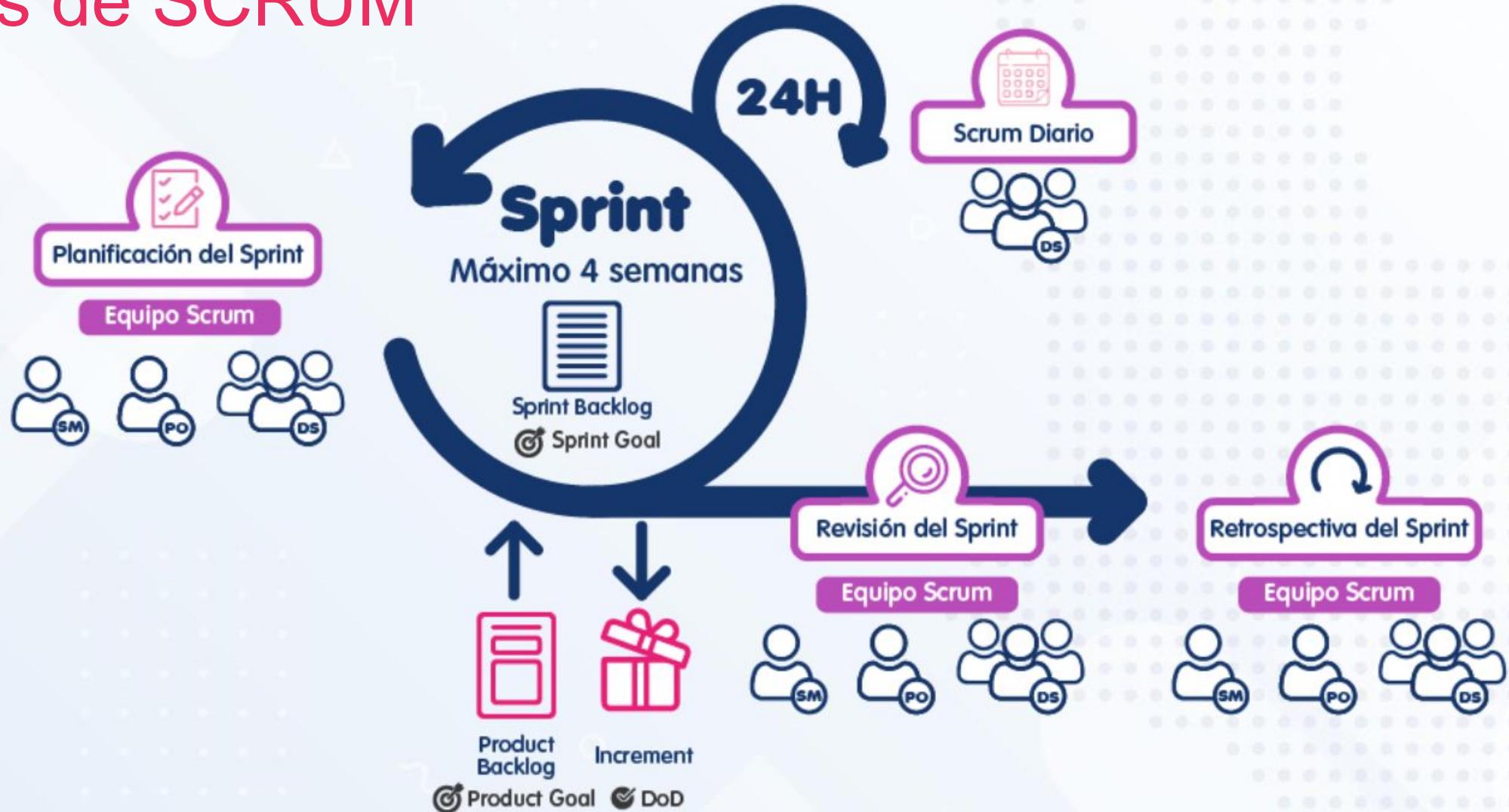


# Roles de la metodología SCRUM

- Product Owner
- Scrum Master
- Scrum Team
- Cliente



# Fases de SCRUM

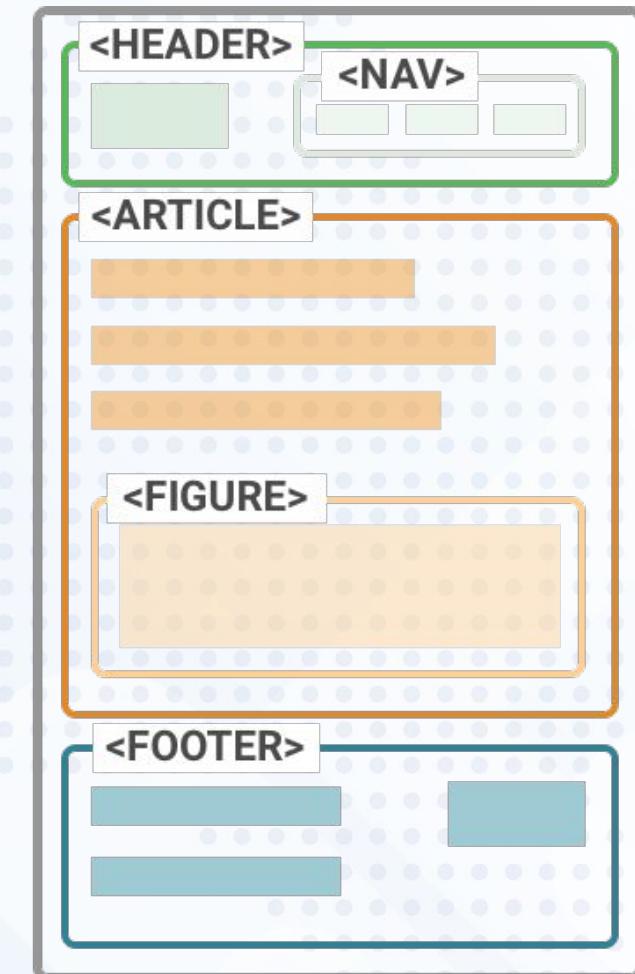
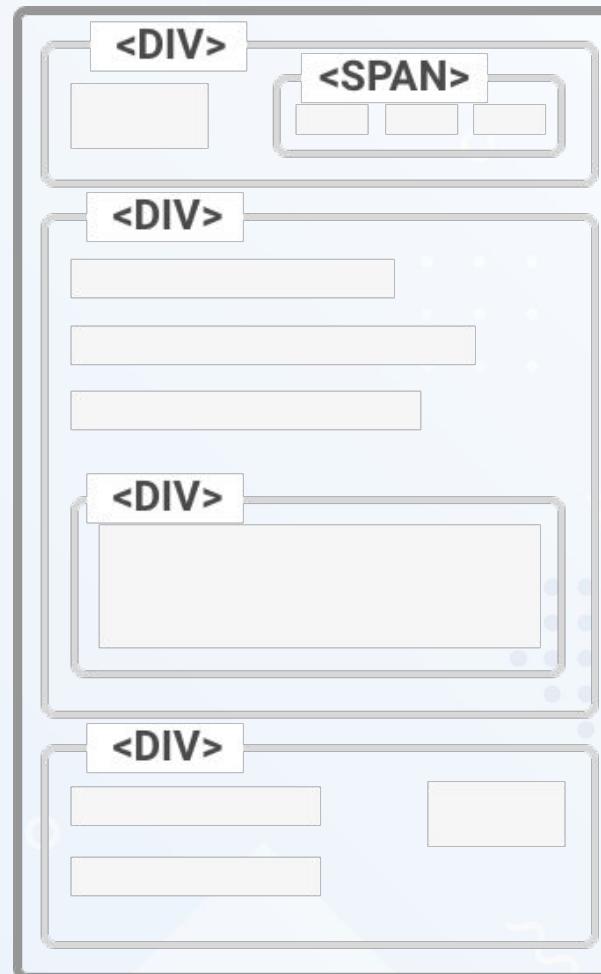


Fuente: [Scrum: el pasado y el futuro | Netmind](#)

# Artefactos

- **Product Backlog (Pila del producto):** es un listado de todas las tareas que se pretenden hacer durante el desarrollo de un proyecto.
- **Sprint Backlog(Pila del Sprint):** Subconjunto de objetivos/requisitos del Product Backlog seleccionado para la iteración actual y su plan de tareas de desarrollo. El equipo lo elabora en la reunión de planificación de la iteración (Sprint planning).
- **Gráfico Burn-Down:** Un gráfico de trabajo pendiente a lo largo del tiempo muestra la velocidad a la que se está completando los objetivos/requisitos. Permite extraer si el equipo podrá completar el trabajo en el tiempo estimado.

# ¿Qué es desarrollo Web?



# ¿Qué es desarrollo Web?



# ¿Qué es desarrollo Web?



**Junior**



**Semi Senior**

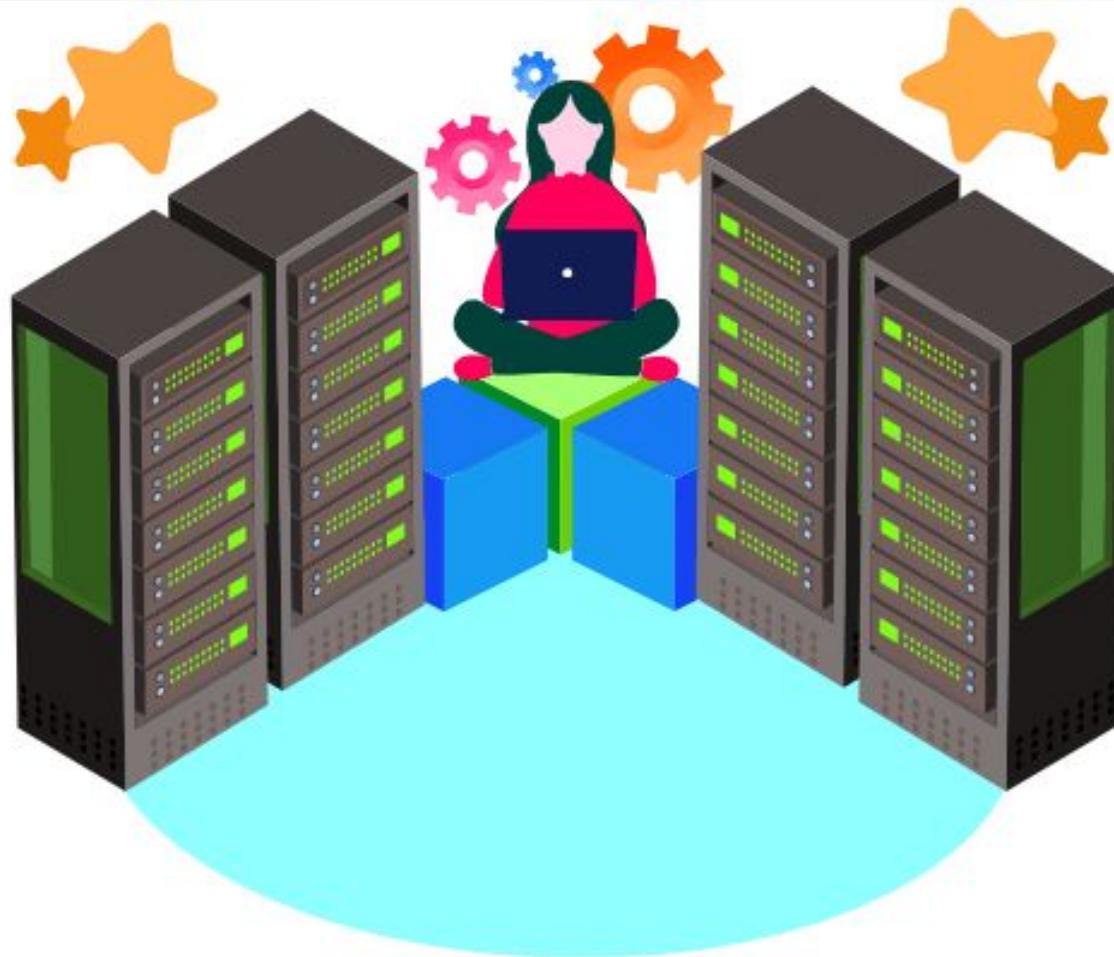


**Senior**

# Frontend



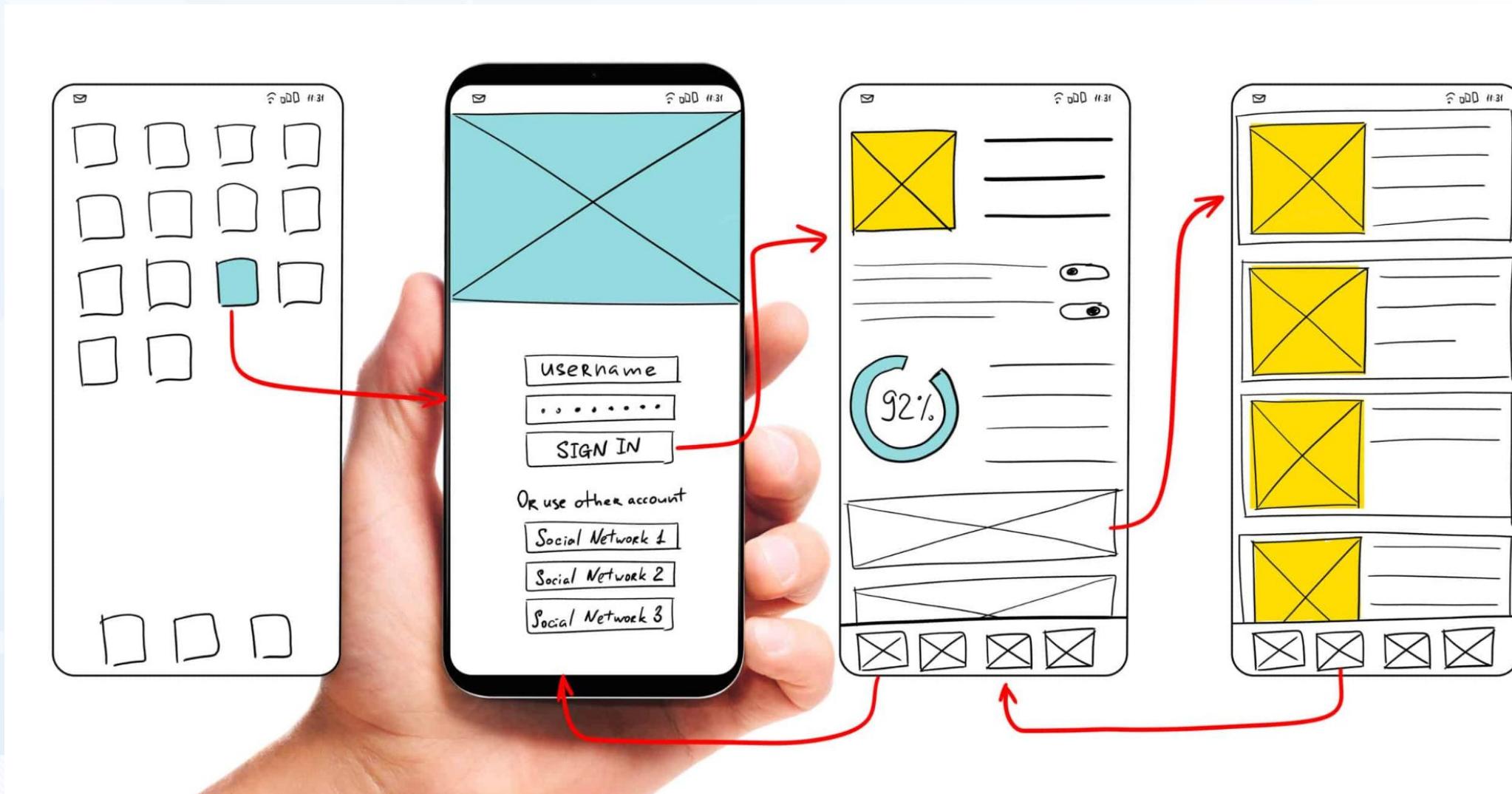
# Backend



# FullStack



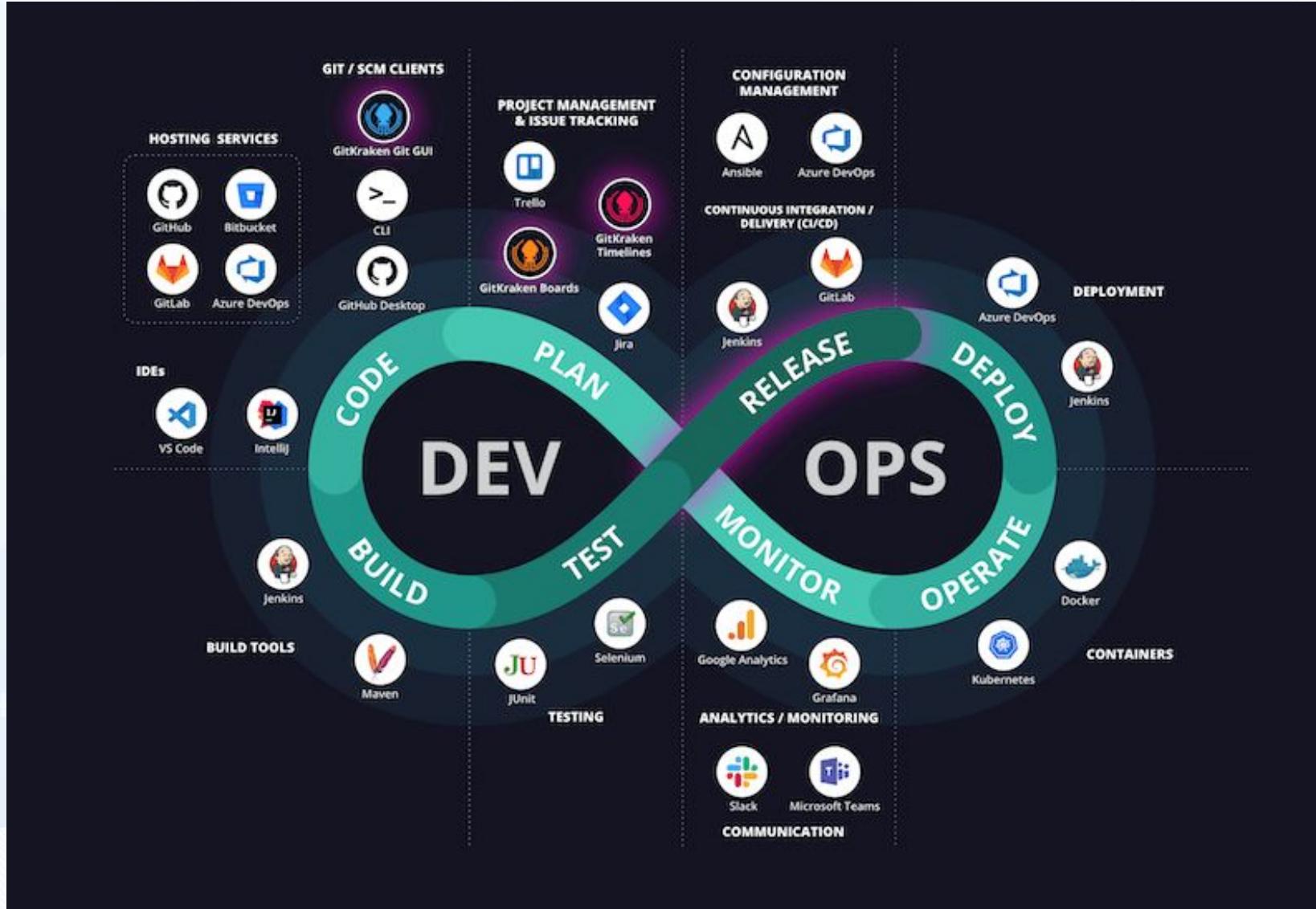
# UX



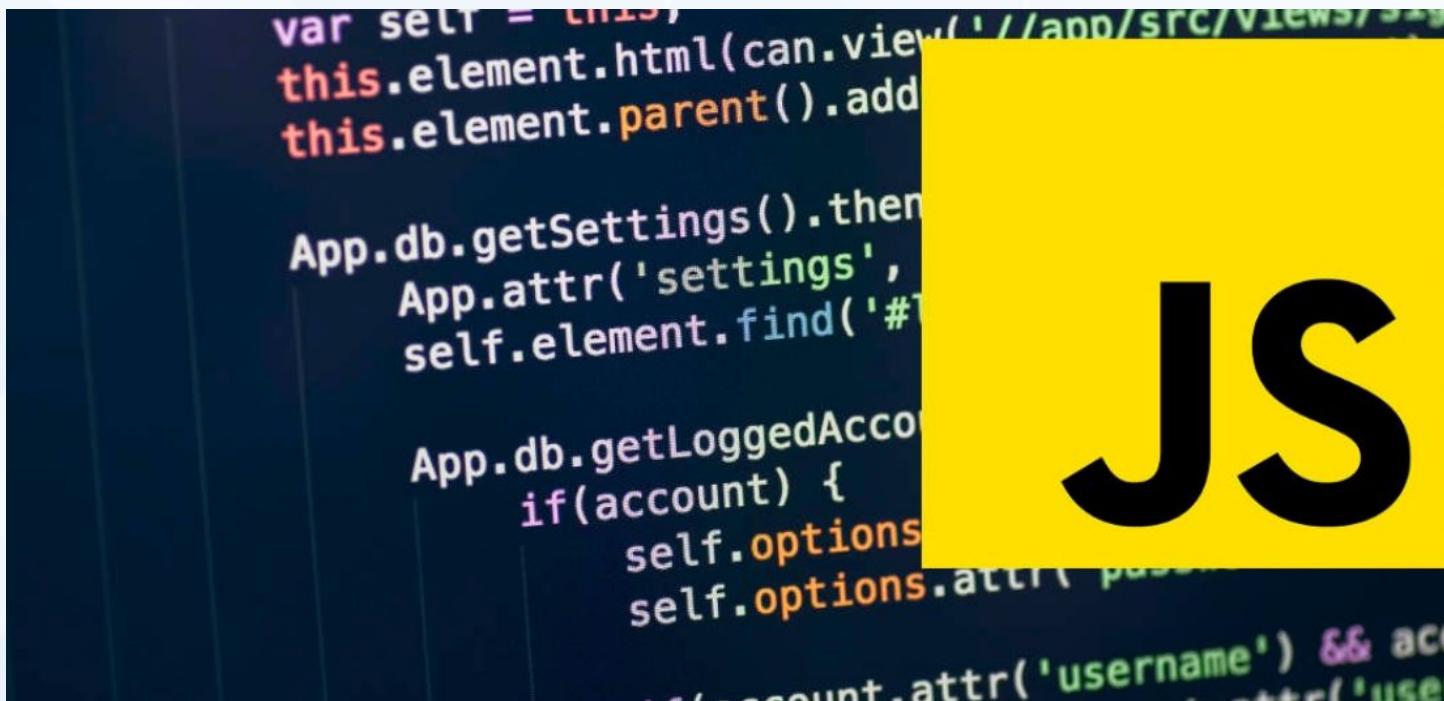
# Tester



# DevOps



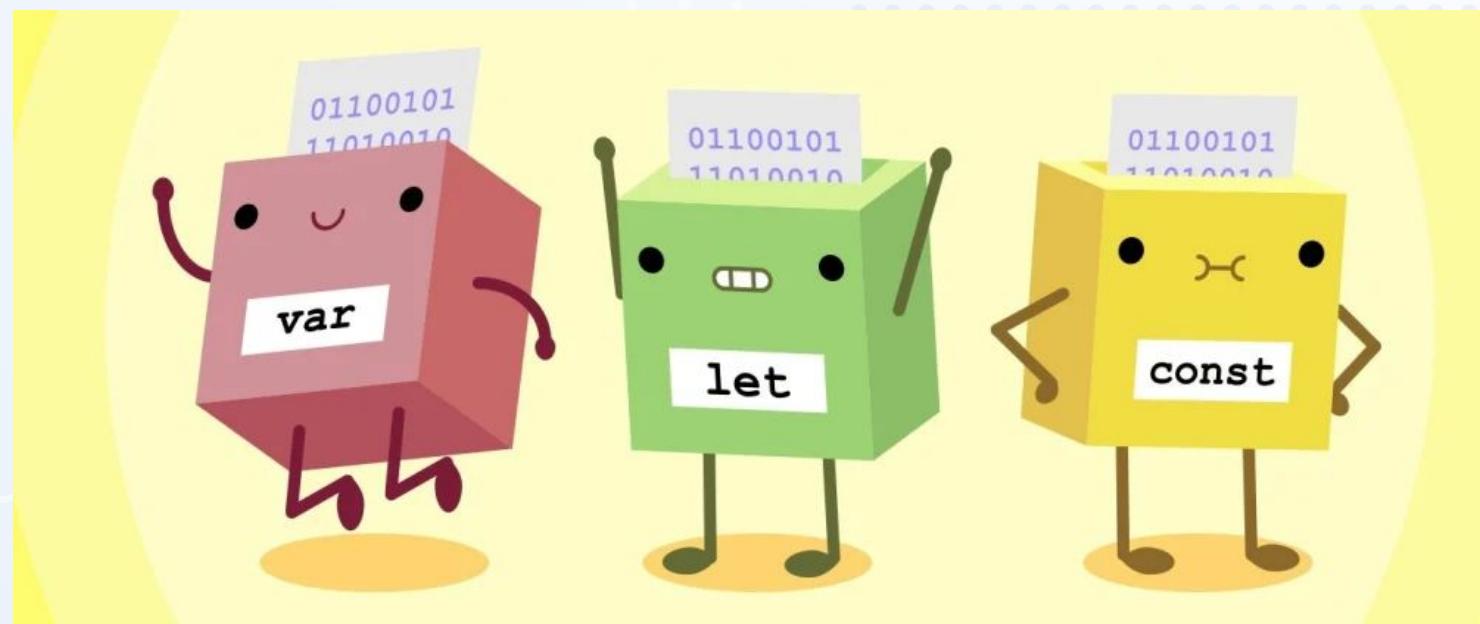
# ¿Qué es JavaScript?



- Lenguaje del lado del cliente
- Lenguaje orientado a objetos
- No es fuertemente tipado
- De alto nivel
- Lenguaje interpretado

# JavaScript

- Lenguaje case-sensitive
- Comentarios en js
- Scope
- Let, const, var



# Funciones

```
function Saludar(){  
    alert("Hola a todos");  
}
```

```
var Cantar = function(){  
    alert("Yo canto muy bien");  
}
```

Ejemplo:

```
function Valores(a,b,c){  
    return a+b+c;  
}  
  
var suma = Valores(3,4,5); // esta función nos retorna el valor de 12
```

# Parámetros de funciones

```
function multiply(a, b = 1) {  
    return a * b;  
}  
  
multiply(5); // 5
```



# Funciones flecha

```
// Función tradicional
function (a){
  return a + 100;
}

// Desglose de la función flecha

// 1. Elimina la palabra "function" y coloca la flecha entre el argumento y el corchete de apertura.
(a) => {
  return a + 100;
}

// 2. Quita los corchetes del cuerpo y la palabra "return" – el return está implícito.
(a) => a + 100;

// 3. Suprime los paréntesis de los argumentos
a => a + 100;
```

# Hoisting

```
function nombreDelGato(nombre) {  
    console.log("El nombre de mi gato es " + nombre);  
}  
  
nombreDelGato("Maurizzio");  
/*  
El resultado del código es: "El nombre de mi gato es Maurizzio"  
*/
```

```
nombreDelGato("Dumas");  
  
function nombreDelGato(nombre) {  
    console.log("El nombre de mi gato es " + nombre);  
}  
/*  
El resultado del código es: "El nombre de mi gato es Dumas"  
*/
```



# Hoisting

```
var x = 5;

(function () {
    console.log("x:", x); // no obtenemos '5' sino 'undefined'
    var x = 10;
    console.log("x:", x); // 10
}());
```

```
var x = 5;

(function () {
    console.log("x:", x); // no obtenemos '5' sino 'undefined'
    var x = 10;
    console.log("x:", x); // 10
}());
```

# Tipos de datos

JavaScript es un lenguaje débilmente tipado y dinámico

```
let foo = 42;      // foo ahora es un número
foo      = 'bar'; // foo ahora es un string
foo      = true;  // foo ahora es un booleano
```



# Tipos de datos primitivos y No primitivos

Los datos primitivos son aquellos que no son un objeto y no tienen métodos

```
// El uso de un método de cadena no modifica la cadena
var bar = "baz";
console.log(bar);          // baz
bar.toUpperCase();
console.log(bar);          // baz

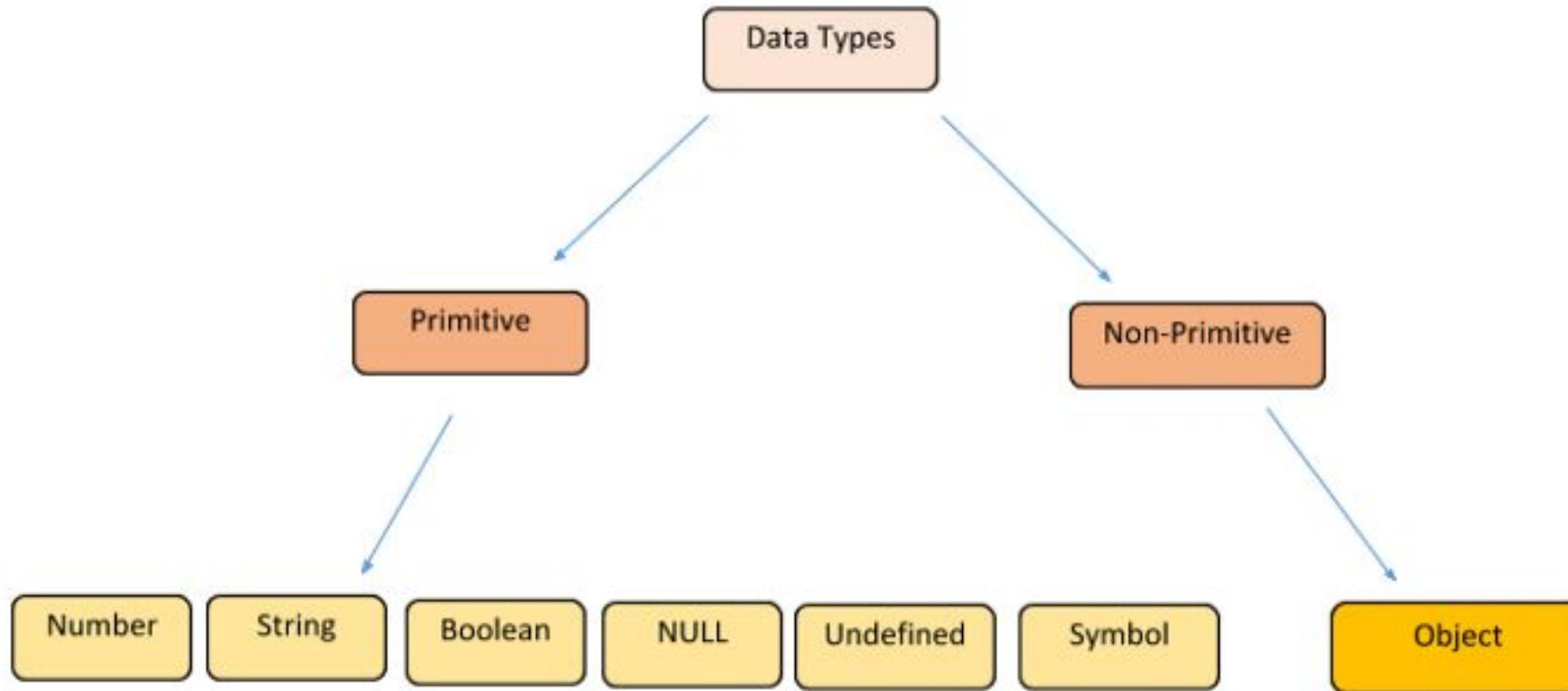
// El uso de un método de arreglo muta el arreglo
var foo = [];
console.log(foo);          // []
foo.push("plugh");
console.log(foo);          // ["plugh"]

// La asignación le da al primitivo un nuevo valor (no lo muta)
bar = bar.toUpperCase();    // BAZ
```



# Tipos de datos

JavaScript es un lenguaje débilmente tipado y dinámico



# Tipos de datos

Concatenación con el operador (+)

```
x = 'La respuesta es ' + 42 // "La respuesta es 42"  
y = 42 + ' es la respuesta' // "42 es la respuesta"
```



```
'37' - 7 // 30  
'37' + 7 // "377"
```



# Strings

Método	Descripción
<a href="#">charAt</a> , <a href="#">charCodeAt</a> , <a href="#">codePointAt</a>	Devuelve el carácter o código de carácter en la posición especificada en la cadena.
<a href="#">indexOf</a> , <a href="#">lastIndexOf</a>	Devuelve la posición de la subcadena especificada en la cadena o la última posición de la subcadena especificada, respectivamente.
<a href="#">startsWith</a> , <a href="#">endsWith</a> , <a href="#">includes</a>	Devuelve si o no la cadena comienza, termina o contiene una subcadena especificada.
<a href="#">concat</a>	Combina el texto de dos cadenas y devuelve una nueva cadena.
<a href="#">fromCharCode</a> , <a href="#">fromCodePoint</a>	Construye una cadena a partir de la secuencia especificada de valores Unicode. Este es un método de la clase <code>String</code> , no una instancia de <code>String</code> .
<a href="#">split</a>	Divide un objeto <code>String</code> en un arreglo de cadenas separando la cadena en subcadenas.
<a href="#">slice</a>	Extrae una sección de una cadena y devuelve una nueva cadena.
<a href="#">substring</a> , <a href="#">substr</a>	Devuelve el subconjunto especificado de la cadena, ya sea especificando los índices inicial y final o el índice inicial y una longitud.
<a href="#">match</a> , <a href="#">matchAll</a> , <a href="#">replace</a> , <a href="#">replaceAll (en-US)</a> , <a href="#">search</a>	Trabaja con expresiones regulares.
<a href="#">toLowerCase</a> , <a href="#">toUpperCase</a>	Devuelve la cadena en minúsculas o mayúsculas, respectivamente.
<a href="#">normalize</a>	Devuelve la forma de normalización Unicode del valor de la cadena llamada.
<a href="#">repeat</a>	Devuelve una cadena que consta de los elementos del objeto repetidos las veces indicadas.
<a href="#">trim</a>	Recorta los espacios en blanco desde el principio y el final de la cadena.

# Array

```
let frutas = ["Manzana", "Banana"]

console.log(frutas.length)
// 2
```



```
let primero = frutas[0]
// Manzana

let ultimo = frutas[frutas.length - 1]
// Banana
```



```
frutas.forEach(function(elemento, indice, array) {
    console.log(elemento, indice);
})
// Manzana 0
// Banana 1
```



# Array

## Añadir un elemento al final de un Array

```
let nuevaLongitud = frutas.push('Naranja') // Añade "Naranja" al final  
// ["Manzana", "Banana", "Naranja"]
```



## Eliminar el último elemento de un Array

```
let ultimo = frutas.pop() // Elimina "Naranja" del final  
// ["Manzana", "Banana"]
```



## Añadir un elemento al principio de un Array

```
let nuevaLongitud = frutas.unshift('Fresa') // Añade "Fresa" al inicio  
// ["Fresa", "Manzana", "Banana"]
```



## Eliminar el primer elemento de un Array

```
let primero = frutas.shift() // Elimina "Fresa" del inicio  
// ["Manzana", "Banana"]
```



# Operadores

## Operadores aritméticos

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	Dividir
%	Resto de la división
++	Incremento
--	Decremento

## Operadores Comparación

Operador	Significado
==	Igual
!=	Distinto
>=	Mayor o igual
<=	Menor o igual
>	Mayor
<	Menor

## Operadores lógicos

Operador	Significado
&&	AND (Y lógico)
	OR (O lógico)
!	NOT (NO lógico)

# Operadores

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
<code>==</code>	Es igual	<code>a == b</code>
<code>====</code>	Es estrictamente igual	<code>a === b</code>

# Estructuras de control (while - if)

```
while (x < 10) {  
    x++;  
}
```



```
if (condition_1) {  
    statement_1;  
} else if (condition_2) {  
    statement_2;  
} else if (condition_n) {  
    statement_n;  
} else {  
    statement_last;  
}
```

# Estructuras de control (switch)

```
switch (fruitytype) {  
    case 'Oranges':  
        console.log('Las naranjas cuestan $0.59 la libra.');//  
        break;  
    case 'Apples':  
        console.log('Las manzanas cuestan $0.32 la libra.');//  
        break;  
    case 'Bananas':  
        console.log('Los plátanos cuestan $0.48 la libra.');//  
        break;  
    case 'Cherries':  
        console.log('Las cerezas cuestan $3.00 la libra.');//  
        break;  
    case 'Mangoes':  
        console.log('Los mangos cuestan $0.56 la libra.');//  
        break;  
    case 'Papayas':  
        console.log('Los mangos y las papayas cuestan $2.79 la libra.');//  
        break;  
    default:  
        console.log(`Lo sentimos, no tenemos ${fruitytype}.`);  
}  
console.log("¿Hay algo más que quieras?");
```



# Estructuras de control (for)

```
for (var i = 0; i < 9; i++) {  
    n += i;  
    mifuncion(n);  
}
```

# Estructuras de control (break -continue)

```
function comprobarBreak(x) {  
    var i = 0;  
    while (i < 6) {  
        if (i == 3)  
            break;  
        i++;  
    }  
    return i * x;  
}
```

```
i = 0;  
n = 0;  
while (i < 5) {  
    i++;  
    if (i == 3)  
        continue;  
    n += i;  
}
```

# Ternarios

```
var stop = false, age = 23;  
  
age > 18 ? (  
    alert("OK, puedes continuar."),  
    location.assign("continue.html")  
) : (  
    stop = true,  
    alert("Disculpa, eres menor de edad!")  
);
```



# Fechas

Permite trabajar con fechas y horas.

Method	Description
getFullYear()	Get the <b>year</b> as a four digit number (yyyy)
getMonth()	Get the <b>month</b> as a number (0-11)
getDate()	Get the <b>day</b> as a number (1-31)
getHours()	Get the <b>hour</b> (0-23)
getMinutes()	Get the <b>minute</b> (0-59)
getSeconds()	Get the <b>second</b> (0-59)
getMilliseconds()	Get the <b>millisecond</b> (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

# Template literals - Template Strings

Los template literals utilizan comillas (` `) en lugar de comillas (") para definir una cadena.

- Se puede usar comillas simples y dobles dentro de un template literal
- Permite multilínea
- Permite interpolar variables y expresiones dentro strings

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;
```

# JSON

JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript.

```
[  
  {  
    "name": "Molecule Man",  
    "age": 29,  
    "secretIdentity": "Dan Jukes",  
    "powers": [  
      "Radiation resistance",  
      "Turning tiny",  
      "Radiation blast"  
    ]  
  },  
  {  
    "name": "Madame Uppercut",  
    "age": 39,  
    "secretIdentity": "Jane Wilson",  
    "powers": [  
      "Million tonne punch",  
      "Damage resistance",  
      "Superhuman reflexes"  
    ]  
  }  
]
```



# Object literals (Objetos literales)

Objetos en javascript

```
var myCar = {  
    make: 'Ford',  
    model: 'Mustang',  
    year: 1969  
};
```



# JSON vs Object literals (Objetos literales)

- JSON se usa cuando desea almacenar datos y transferirlos a diferentes plataformas y lenguajes de programación.
- Una forma fácil de detectar JSON es que verá que los nombres de sus propiedades siempre están entre comillas
- A menudo verá datos JSON almacenados como un archivo separado de JavaScript (.json)
- Los object literals aparecen directamente en el código JS
- En los object literals también puedo almacenar funciones

# TypeOf

Operador que se puede usar para encontrar el tipo de dato de una variable

```
typeof "John"           // Returns "string"
typeof 3.14             // Returns "number"
typeof NaN              // Returns "number"
typeof false            // Returns "boolean"
typeof [1,2,3,4]         // Returns "object"
typeof {name:'John', age:34} // Returns "object"
typeof new Date()        // Returns "object"
typeof function () {}    // Returns "function"
typeof myCar             // Returns "undefined" *
typeof null              // Returns "object"
```

# Try catch

Manejar errores

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}
```

# Callbacks

Una función de callback es una función que se pasa a otra función como un argumento, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

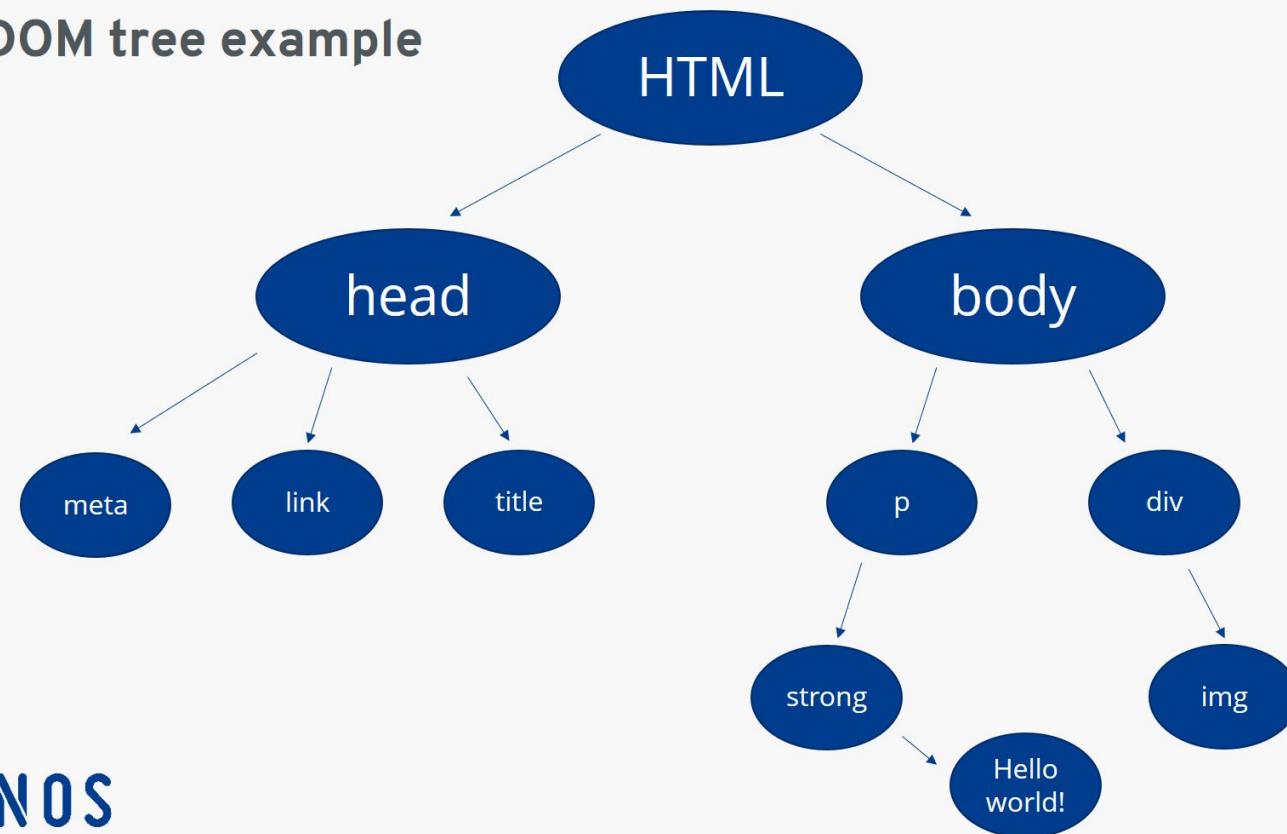
```
function saludar(nombre) {  
    alert('Hola ' + nombre);  
}  
  
function procesarEntradaUsuario(callback) {  
    var nombre = prompt('Por favor ingresa tu nombre.' );  
    callback(nombre);  
}  
  
procesarEntradaUsuario(saludar);
```



# Manejo del DOM(Document Object Model)

Es una interfaz de programación para los documentos HTML. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

DOM tree example



# DOM WEB API

```
document.getElementById("firstName");
```

```
document.getElementsByTagName("span");
```

```
document.getElementsByClassName("container");
```

```
document.querySelector(".container");
```

```
document.querySelectorAll(".container");
```

# DOM WEB API

```
document.createElement("div");
```

```
document.createTextNode("Este es un nodo de texto");
```

```
const list = document.querySelector("#list");
const item = document.createElement("li");
const text = document.createTextNode("Item 1");

// Insertamos el texto dentro del item
item.appendChild(text);

// Insertando el item en la lista
container.appendChild(item);
```

# DOM WEB API

```
// Obtenemos el elemento al que queremos cambiar el contenido
let container = document.querySelector(".container");

// Reemplazamos el contenido
container.innerHTML = "<span>Bienvenidos</span>";
```

```
input.setAttribute("name", "fullName");

// Ahora el input quedará de la siguiente manera

<input class="form-control"
      id="persona-nombre"
      placeholder="Nombre completo"
      name="fullName" />
```

# DOM WEB API

```
input.getAttribute("name"); // Salida: "fullName"
```

```
input.removeAttribute("name");

// input final
<input class="form-control"
       id="persona-nombre"
       placeholder="Nombre completo" />
```

# DOM WEB API

```
// Nuestro HTML
<div class="parent">
  <p class="p-child">Titulo</p>
  <span class="span-child">Subtitulo</span>
</div>

// Obtenemos el nodo padre
const parent = document.querySelector(".parent");

// Obtenemos el nodo a eliminar
const nodoAEliminar = document.querySelector(".p-child");

// Eliminamos el nodo
parent.removeChild(nodoAEliminar);
```

# Eventos

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

# TypeScript

Es un superset de js creado por Microsoft. La primera versión salió en 2012. Es un superconjunto de tipo estricto del lenguaje JavaScript. TypeScript es un lenguaje de programación de código abierto que también recibe muchas mejoras de la comunidad. Es necesario instalar TypeScript para poder usarlo.

JavaScript	TypeScript
Lenguaje de <i>scripting</i> orientado a objetos	Lenguaje de <i>scripting</i> orientado a objetos
Puede ser interpretado y ejecutado	Compila a JavaScript
Lenguaje interpretado, ejecutado directamente en un navegador	Lenguaje compilado, no se puede ejecutar directamente en un navegador
Dinámicamente tipado	Puede ser estáticamente tipado
Más flexible	Mejor estructurado
Bueno para proyectos pequeños y simples	Ideal para proyectos complejos

# TypeScript -variables

```
var full_nombre:string = "jorge cano";
```

```
var age:number = 27;
```

```
var developer:boolean = true;
```

```
var skills:Array <string>= ['JavaScript','TypeScript','Angular'];
```

```
var numberArray:number[] = [123,123,1213,1231];
```

# TypeScript -Enums

Las enumeraciones permiten a un desarrollador definir un conjunto de constantes con nombre. El uso de enumeraciones puede hacer que sea más fácil documentar la intención o crear un conjunto de casos distintos. TypeScript proporciona enumeraciones numéricas y basadas en cadenas.

```
enum Status {  
    ok = 200,  
    created = 201,  
    bad_request = 400,  
    not_found = 404  
}  
  
const errorCode = Status.ok;
```

```
enum Direction {  
    Up = "UP",  
    Down = "DOWN",  
    Left = "LEFT",  
    Right = "RIGHT"  
}
```

# TypeScript - funciones

```
function hello():void {  
}
```

```
function setName(name:string):void{  
}
```

```
function setName(name:string, surName:string ):string {  
    return "string";  
}
```

# TypeScript - clases

```
class Persona {  
    nombre:string;  
    constructor(nuevoNombre:string) {  
        this.nombre=nuevoNombre;  
    }  
  
    decirMiNombre() {  
        console.log(this.nombre);  
    }  
}  
  
let persona = new Persona("Camila");
```

```
class Persona {  
    nombre:string;  
    constructor(nuevoNombre?:string) {  
        this.nombre=nuevoNombre;  
    }  
  
    decirMiNombre() {  
        console.log(this.nombre);  
    }  
}  
  
let persona = new Persona();
```

# TypeScript - interfaces

“Interfaces, y a veces llamadas firmas, es el mecanismo que usa Typescript para definir tipos en las clases”

```
● ● ●

interface Recipe {
  name: string;
  servingSize: number;
  spiceLevel?: number;
}

const cake: Recipe = {
  name: 'cake',
  servingSize: 8,
};

// We create a cake Recipe, and there is no error for not having
// a spiceLevel key
```