

# JavaScript Avanzado

Clase 01



# Presentación

## 👋 Fernando Luna

- 📝 Technical Writer + 👨💻 Software Engineer
- 👨🏫 Profesor
  - JavaScript + JavaScript Avanzado
  - NodeJS + MongoDB
  - BB.DD. SQL
  - Aplicaciones Progresivas
- 🍽️ Platos ricos {stress: 'off'}
- [fernando.luna@istea.com.ar](mailto:fernando.luna@istea.com.ar)



# RoadMap del curso

- Manejo de DOM moderno y completo
- Contenidos, Estilos, Atributos y clases
- Controlar formularios HTML desde JS
- Objetos con JavaScript (*toda su evolución*)
- Manejo de multimedia con JS
- Integración de librerías JS de terceros
- Módulos en JavaScript (*export - import*)
- El paradigma AJAX
- API Restful (backend + GET POST PUT DELETE)
- Asincronismo (*async - await*)
- Fundamentos Promesas JavaScript
- EcmaScript moderno (ES6 => ES15/16)
- Persistencia y optimización (*WebStorage*)
- Performance Web (*optimización*)
- Funciones de orden superior
- Node JS - NPM - Vite



# Agenda de hoy

- Motores Web
- Herramientas de JS
- Repaso general de las bases de JS
  - Variables
  - Constantes
  - Arrays
    - Métodos de arrays
  - Funciones
    - Simples
    - Parámetros
    - Retorno
- Prácticas



# Motores Web



# Motores Web

Cada navegador web cuenta con lo que se denomina **motor web**.

Éste se ocupa de interpretar el código descargado de cada uno de los sitios web que visitamos.

Poseen diferentes nombres, según el tipo de web browser que utilicemos.

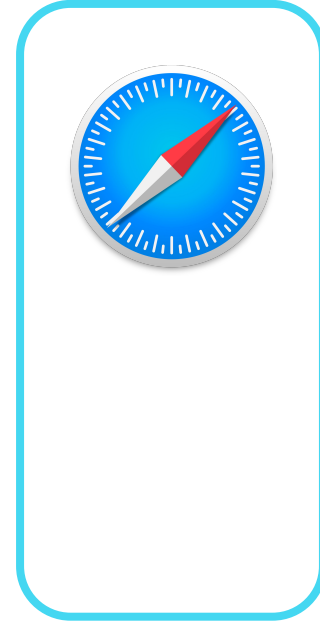
**Blink**



**Gecko**



**Webkit**

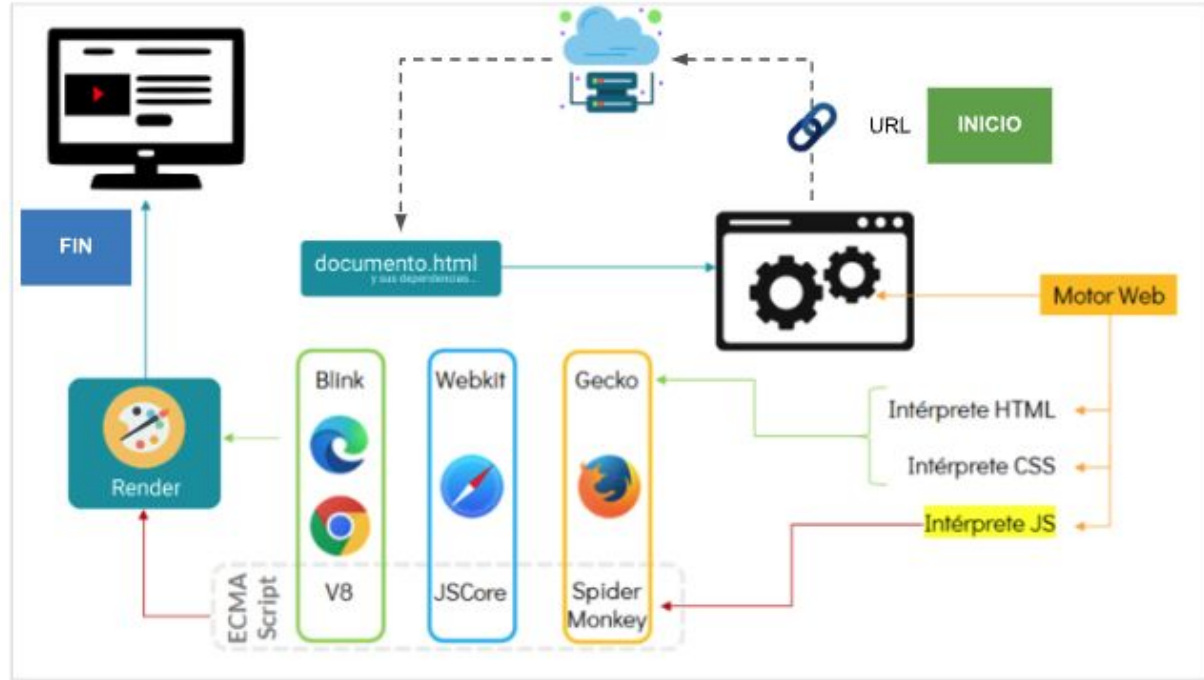


# Motores Web

A su vez, **cada motor web**, se subdivide en **dos intérpretes** diferentes:

- uno interpreta la sintaxis HTML y CSS
- otro interpreta la sintaxis JavaScript

En algún punto, ambos intérpretes dialogan, para intercambiar información.



# Herramientas de JavaScript





# Herramientas de JavaScript

JavaScript cuenta con una serie de herramientas, integradas en el lenguaje en sí.

Algunas de ellas son útiles dentro de la programación de soluciones web, y otras tantas son utilizadas para nosotros “*desarrolladores*”, cuando debemos hacer pruebas, o depurar el código que estamos trabajando.



# Herramientas de JavaScript

## Cuadros de diálogos

JS cuenta con una serie de cuadros de diálogos.

- `alert()`
- `confirm()`
- `prompt()`

Estos permiten generar interacción con el usuario, mostrando un mensaje de texto y/o solicitando que el usuario ingrese un dato o tome una decisión dentro de la lógica de la aplicación.



# Herramientas de JavaScript

## objeto console

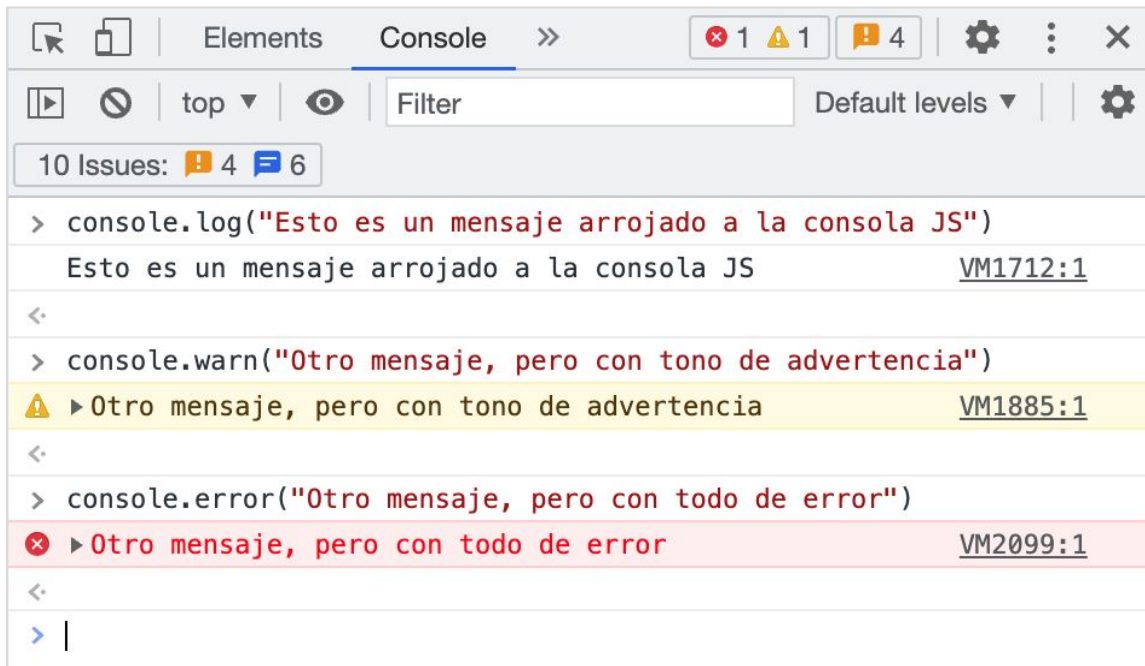
El objeto console **cuenta con una serie de métodos**, útiles para depurar las aplicaciones web que construimos.

- `log()`
- `warn()`
- `error()`
- `table()`

Permiten loguear mensajes, valores de variables o constantes, y contenido de un array, a la consola JS de DevTools.



# Herramientas de JavaScript

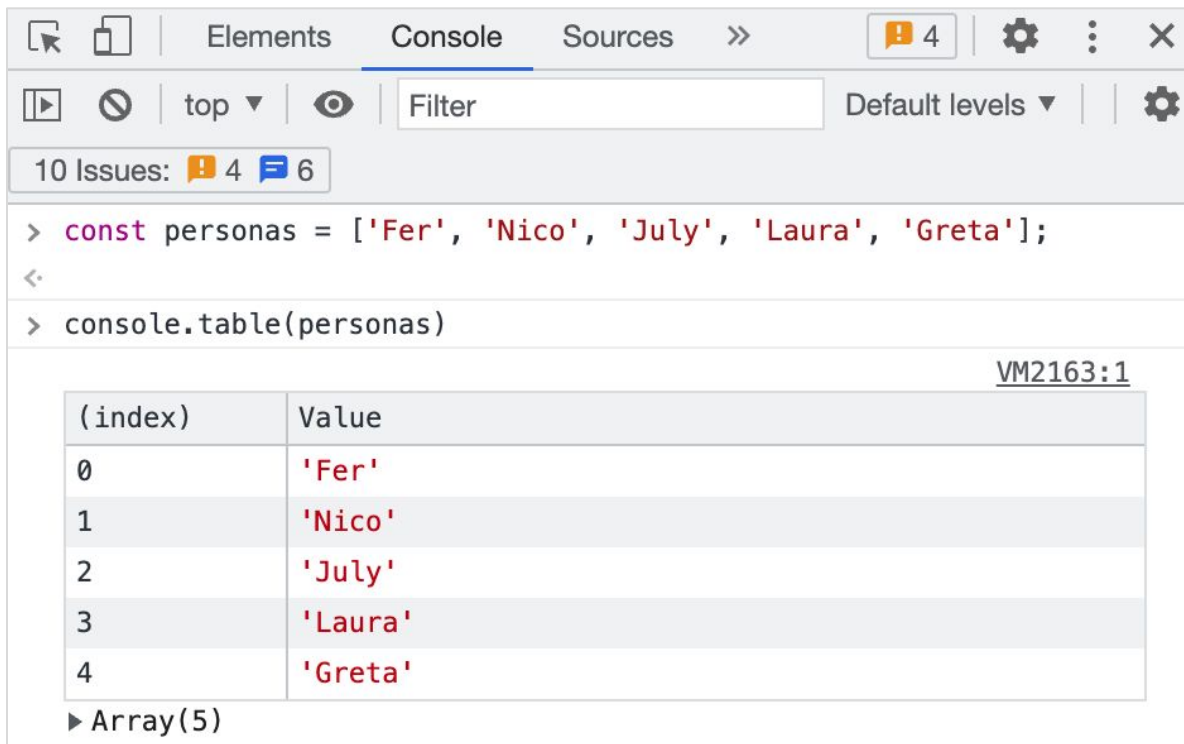


El objetivo de estos métodos es poder loguear mensajes a la consola JS. Pero deben eliminarse cuando una aplicación pasa al ámbito de **Producción**.

Solo en muy pocos casos, se pueden dejar implementados algunos de ellos.



# Herramientas de JavaScript



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following code and output:

```
> const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];  
<  
> console.table(personas)
```

The output is a table with 5 rows, each representing an element in the array. The table has two columns: '(index)' and 'Value'. The values are 'Fer', 'Nico', 'July', 'Laura', and 'Greta'.

(index)	Value
0	'Fer'
1	'Nico'
2	'July'
3	'Laura'
4	'Greta'

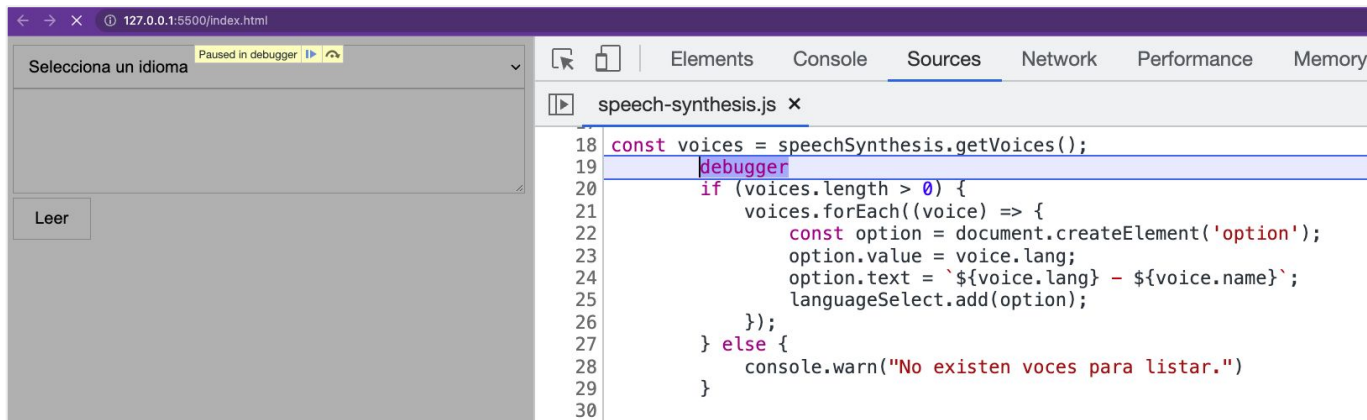
Below the table, the console shows '► Array(5)', indicating the array has 5 elements.

El método **.table()** loguea el contenido de un array en la consola JS, obteniendo una referencia del índice que le corresponde a cada elemento.

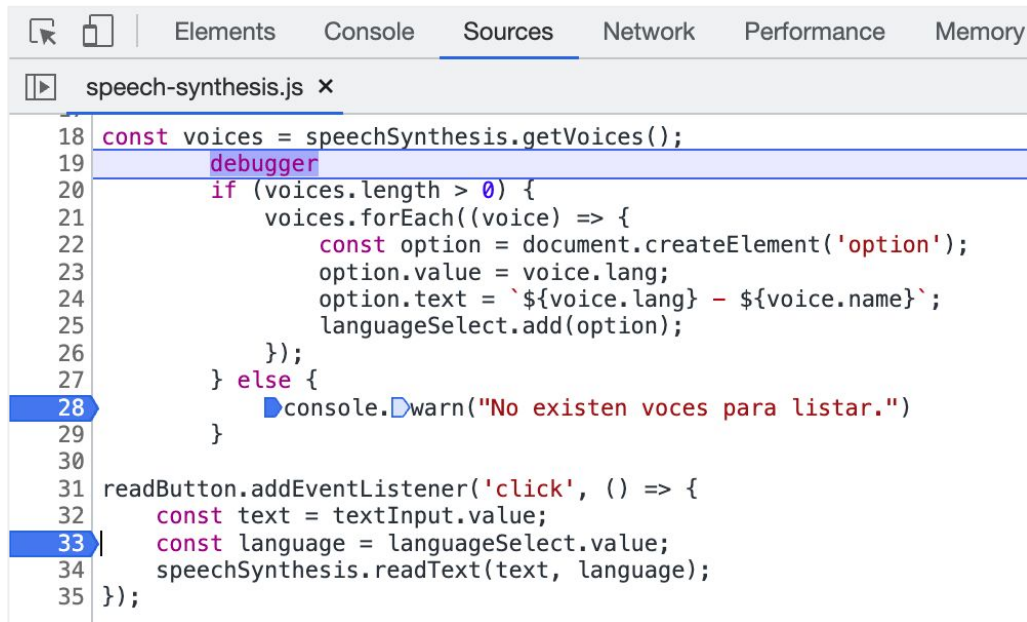
Además, nos devuelve información del total de elementos.

# Herramientas de JavaScript

**debugger** es ideal para depuración de código. Genera un punto de interrupción en nuestro código, el cual nos permite ir ejecutando línea a línea, viendo qué sucede en la aplicación web, qué valores toman las variables, constantes, etcétera.



# Herramientas de JavaScript



The screenshot shows the Chrome DevTools Sources panel with the file 'speech-synthesis.js' open. The code is as follows:

```
18 const voices = speechSynthesis.getVoices();
19 debugger
20 if (voices.length > 0) {
21   voices.forEach((voice) => {
22     const option = document.createElement('option');
23     option.value = voice.lang;
24     option.text = `${voice.lang} - ${voice.name}`;
25     languageSelect.add(option);
26   });
27 } else {
28   console.warn("No existen voces para listar.");
29 }
30
31 readButton.addEventListener('click', () => {
32   const text = textInput.value;
33   const language = languageSelect.value;
34   speechSynthesis.readText(text, language);
35 });
```

Line 19 is highlighted with a blue background and a red 'debugger' label. Line 28 is also highlighted with a blue background. The 'Sources' tab is selected in the top bar.

**debugger** solo funciona con DevTools activado.

A su vez, **DevTools > Sources**, nos permite definir manualmente otros tantos puntos de interrupción en el código JS.

# Programación con JS





# Programación con JS

## Variables

Las variables son espacios en memorias reservados habitualmente por una aplicación, para contener valores, ya sea generados previamente por la aplicación en sí, o que vayan a ser generados por el usuario si es que carga algún dato que deba ser guardado por la aplicación.

Pueden declararse, inicializarse, y cambiar su valor, de acuerdo al comportamiento del programa, el cual es definido por nosotros “*programadores*”.



# Programación con JS

Los valores que puede almacenar una variable dentro del lenguaje JS, pueden ser diferentes. Entre los más comunes, encontramos a:

- numéricos
- cadenas alfanuméricas (textos)
- boolean
- objetos
- arrays



# Programación con JS

Variables declaradas  
e inicializadas, con  
valores asignados de  
forma estática.

```
Variables

let nombre = 'Educación IT';

let numero = 2103;

let usuarioLogueado = true;

let cursos = ['JavaScript Avanzado', 'VueJS', 'React', 'PWA'];

let persona = {nombre: 'Fer', ocupacion: 'Profesor'};
```



# Programación con JS



Declaración de variables en JavaScript.

Siempre utilizar la palabra reservada **let**, en lugar de **var**.

Si bien ambas están disponibles, y lo seguirán estando por muchos años más, desde 2015 se recomienda desestimar el uso de **var**.



# Programación con JS



```
let companyName = "Educación IT"  
  
let totalCarrito = 15,550.91
```

Declaración de variables de propósito general. Toda variable debe hacer honor a su nombre cuando la declaramos.

# Programación con JS



```
let div = document.getElementById("divCarrito")  
  
let container = document.getElementById("container")
```

Todo aquel elemento que no deba variar el valor asignado dentro de una aplicación JS, nunca debe declararse con **let** ni tampoco con **var**.

# Programación con JS



```
const div = document.getElementById("divCarrito")  
  
const container = document.getElementById("container")
```

**Ej:** *objetos literales, funciones constructoras, flecha, anónimas, acceso a elementos HTML, arrays...* todo debe ser declarado con la palabra reservada **const**.

# Propiedades y métodos en variables





# Propiedades y métodos en variables

Toda variable hereda una serie de propiedades y métodos, dado que por cada variable que creamos, esta es instanciada a partir de un objeto.



Variables

```
let nombre = 'Educación IT';  
  
//instancia del objeto String
```



# Propiedades y métodos en variables

Los métodos disponibles para una variable del tipo **String**, son propios del tipo de datos en cuestión. Una variable numérica no contendrá métodos como **toUpperCase** o **toLowerCase**.

```
Variables

let nombre = 'Educación IT';

nombre.length //devuelve el total de caracteres
nombre.at(2) //devuelve el caracter en dicha posición
nombre.trim() //elimina espacios agregados al inicio o final del
nombre.toUpperCase() //convierte el texto a mayúsculas
nombre.toLowerCase() //convierte el texto a minúsculas
```



# Propiedades y métodos en variables

Los métodos **numéricos** nos ayudan a trabajar mejor el formato de cada número, de acuerdo a la necesidad al momento de almacenarlos, o de representarlos en pantalla.

Variables

```
let numero = 2103;

numero.toPrecision(6); //convierte el número a 6 dígitos, acortando o agregando
numero.toFixed(2);     //fuerza un redondeo decimal a la cantidad numérica
numero.toLocaleString(); //formatea el número a la configuración regional del S.O.
numero.toString();     //convierte el número en texto
```

# Propiedades y métodos en variables

Toda variable en JS es de tipado débil. Esto implica que cuando se declara una variable, no se define el tipo de datos que contendrá. Se le asigna el mismo directamente.

A su vez, podemos cambiar el valor de una variable, asignándole otro tipo de datos, sin ningún problema. 🧘

```
Variables

let numero = 2103;
console.log(numero.toFixed(2)) //muestra en consola 2103.00

numero = 'Otro valor';
console.log(numero)           //muestra en consola 'Otro
valor'
```



# Constantes

# Constantes

## Constante

Una constante se diferencia de una variable porque, una vez creada la misma, esta no podrá cambiar durante el ciclo de vida de la aplicación. De allí su nombre “*constante*”.

```
Constantes

const nombre = 'Educación IT';
console.log(nombre);           //muestra 'Educación IT' en la consola JS

nombre = 'ISTEA';              //muestra un error en la consola JS
```

✖ ▶ Uncaught TypeError: Assignment to constant variable.  
at <anonymous>:1:5

VM421:1

> |



# Constantes



**Las constantes deben ser utilizadas siempre que declaremos objetos, arrays, o cualquier elemento que enlace JS con el DOM HTML.**

De esta forma evitaremos utilizar esa palabra al declarar otra variable, o sobrescribir a esta, por accidente, con cualquier otro valor.

Constantes

```
const persona = {nombre: 'Fer Moon', ocupacion: 'Profesor'};  
  
const container = document.getElementById('container');
```



**var** versus **let**







## **var** versus **let**

Ambas son palabras reservadas para declarar variables en JS. Tienen algunas diferencias mínimas en su comportamiento.

**var**: toda variable declarada con var, puede volver a declararse en cualquier otra parte de nuestro programa.

**let**: las variables declaradas con let, no pueden volver a declararse, pero cumplen un papel importante dentro del **Scope**, lo cual veremos más adelante.



# var versus let

Si bien ambas opciones se pueden seguir utilizando en JS, desde 2015 se viene promoviendo el uso de **let** dentro de las aplicaciones web, por sobre el uso de **var**.

💡 Como recomendación, desestima el uso de **var**, y utiliza **let** siempre que declares variables en JS. Ten esto presente siempre, porque ya lleva 9 años desde que comenzó a promoverse el uso de **let**.



# Funciones

# Funciones

Las funciones son pequeños contenedores de bloques de código, las cuales nos permite definir una tarea o rutina que puede ser ejecutada en diferentes partes de nuestra aplicación.

Existen tres tipos de funciones:

- convencionales
- con parámetros
- con retorno

Ambas se pueden combinar, de acuerdo a la necesidad, del desarrollo de la aplicación.



# Funciones

Las funciones, tanto en JavaScript, como en cualquier otro lenguaje de programación, respetan una serie de principios.

- [DRY](#)
- [KISS](#)
- [YAGNI](#)

Su finalidad principal, es poder realizar una tarea determinada, de la forma más agnóstica posible, y así ser llamada desde cualquier parte de nuestra aplicación. Esto evitará que repitamos código de manera innecesaria.



# Funciones

```
Funciones

let numero1 = 21
let numero2 = 3

function calcular() {
  let resultado = numero1 * numero2;
  console.log(resultado);
}
```

**Función convencional:** se declaran con la palabra reservada function, y se define dentro del bloque, el código a ejecutar.

# Funciones

**Función con parámetros:** reciben uno o más parámetros a través de los paréntesis.

Estos pueden ser a partir de variables o constantes que tengan valores, o de datos estáticos que los pasamos directamente cuando invocamos a la función.

```
Funciones con parámetros

let numero1 = 21
let numero2 = 3

function calcular(num1, num2) {
  let resultado = num1 * num2;
  console.log(resultado);
}

//llamado a la función
calcular(numero1, numero2)

//Otro llamado a la función
calcular(1975, 2103)
```

# Funciones

```
Funciones con retorno

let numero1 = 21
let numero2 = 3

function calcular(num1, num2) {
  let resultado = num1 * num2;
  return resultado;
}

//llamado a la función
calcular(numero1, numero2)
//escribe en la consola el resultado de la
multiplicación
```

**Función con retorno:** utilizan la palabra reservada **return**, la cual devuelve el resultado de una operación.

Pueden, o no, recibir parámetros. Usualmente este tipo de función se utiliza con una constante o variable donde se atrapa el valor devuelto.



# Arrays

# Arrays

Los **arrays**, usualmente llamados **colecciones**, nos permiten definir bajo una misma estructura, una serie de valores útiles dentro de nuestra aplicación.

Estos valores pueden ser de un mismo tipo de dato, (*o de diferentes tipos*), aunque esto último no es muy común por una cuestión de buenas prácticas.



Arrays

```
const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];  
  
const pouporri = ['Moon', 21, 75, true, personas];
```



# Arrays

**Cada valor almacenado** en un array, **obtiene una posición numérica, denominada índice**. Esta posición numérica inicia siempre a partir del valor 0 (cero), y se incrementa en un dígito por cada nuevo valor contenido en el array.

```
Arrays

const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];
// posición      0      1      2      3      4
```

# Arrays

Para obtener el valor almacenado dentro de un array, sólo debemos invocar el nombre del array e indicar entre corchetes, la posición del valor en cuestión.

```
Arrays

const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];
// posición      0      1      2      3      4

personas[3];    //devuelve 'Laura'
```

# Arrays

La propiedad **.length**, nos permite conocer cuántos elementos están almacenados en el array. Devuelve como resultado un valor numérico.



Arrays

```
const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];  
// posición      0      1      2      3      4  
  
personas.length; //devuelve el número 5 como resultado
```

# Arrays

Dentro de los métodos convencionales de un array, contamos con cuatro opciones para agregar o quitar elementos: **.push()** - **.pop()** - **.unshift()** - **.shift()**

```
Arrays

const personas = ['Fer', 'Nico', 'July', 'Laura', 'Greta'];
// posición      0      1      2      3      4

personas.push('Constantito'); //agrega un nuevo elemento al final del array.
personas.unshift();           //agrega un nuevo elemento al inicio del array.
personas.pop();                //quita el último elemento del array.
personas.shift();              //quita el primer elemento del array.
```



# Arrays

Existen otros tantos métodos de arrays, los cuales nos permiten realizar todo tipo de operaciones sobre los datos almacenados en este. Veamos cuáles son:

Método	Descripción
<b>forEach()</b>	Permite iterar el contenido de un array. Retorna un callback como parámetro del elemento actual de la iteración, y su índice.
<b>find()</b>	Recorre el array y retorna la primera coincidencia del elemento que busca.
<b>sort()</b>	Ordena los elementos del array y retorna un arreglo ordenado.
<b>reverse()</b>	Invierte el orden actual de los elementos del array.
<b>concat()</b>	Concatena dos o más arrays, sin cambiar los existentes.
<b>includes()</b>	Determina si un array incluye un determinado elemento, retornando un valor booleano.
<b>indexOf()</b>	Retorna el primer índice en el que se encuentra un elemento dato. Si no existe el elemento, devuelve -1.



# Arrays

Método	Descripción
<b>findIndex()</b>	Retorna el índice del primer elemento de un array que cumpla con la función de prueba proporcionada. Caso contrario retorna -1.
<b>fill()</b>	Cambia todos los elementos de un array por un valor estático, y retorna un array modificado.
<b>slice()</b>	Devuelve una copia de una parte del array. Se puede indicar un inicio y un fin.
<b>splice()</b>	Quita uno o más elementos del array, empalmando los valores de los extremos resultantes.
<b>lastIndexOf()</b>	Busca un elemento en un array y devuelve su posición. La búsqueda la inicia a partir del último elemento del array.
<b>flat()</b>	Crea un nuevo array con todos los elementos de éste, y de cualquier sub-array interno.
<b>join()</b>	Uno los elementos de un array en una cadena, devolviendo un string como resultado.



# Arrays

Método	Descripción
<b>some()</b>	Itera el array y retorna un valor booleano si, como mínimo, uno de los elementos presentes pasa una condición determinada.
<b>every()</b>	Es similar a some(), pero con la diferencia de que dicho booleano será true, si todos los elementos del array pasan la condición determinada.
<b>filter()</b>	Recorre un array y retorna un nuevo array con los elementos que cumplan una determinada condición.
<b>map()</b>	Permite recorrer un array y modificar (mapear) los elementos presentes, retornando un nuevo array con la misma longitud del original.

Son varios los métodos. Algunos de ellos se nombran como funciones de orden superior, y cumplen un papel importante cuando se manipulan arrays de objetos.





# Espacio de prácticas





# Espacio de prácticas



25 minutos 

## Crea una función JS llamada `obtenerNombre()`

- define una constante **nombre**, la cual obtendrá mediante **prompt()**, el nombre del usuario
- valida qué **nombre** posea un valor, utilizando el condicional **if**. Aplica todos los métodos de String que consideres necesarios
- escribe el valor de esta constante en la consola JS mediante el método **.log()**
- si la constante no posee un valor, entonces muestra una advertencia en la consola JS utilizando el método **warn()**

## Crea una función JS llamada `recorrerArray()`

- define el siguiente array `const cursos = ['Vanilla JS', 'PWA', 'React JS', 'VueJS', 'Svelte', 'Vanilla CSS'];`
- utiliza el ciclo **for** convencional para recorrer el array. Por cada iteración, loguea en la consola JS, el valor del elemento actual del array

## Crea una función JS llamada `ubicarElemento(param)`

- define el array anterior de forma global, luego crea la función indicada la cual recibirá un parámetro
- utiliza sobre el array, el método **indexOf()** para identificar si existe el elemento recibido como parámetro
- **si existe**, loguea en la consola JS un mensaje acorde, **sino**, loguea una advertencia

## Crea una función JS llamada `elementosCoincidentes(param)`

- recorre el array completo y loguea en la consola JS, los elementos que coincidan con el nombre, o parte de este
- permite que la búsqueda del elemento sea por parte del nombre, y no por el nombre exacto;  
ten en cuenta que el usuario puede escribirlo en minúsculas, mayúsculas, y hasta agregar espacio al inicio o final del texto



# Espacio de prácticas



Puesta en común del ejercicio

**¡Gracias!**

**educación IT**