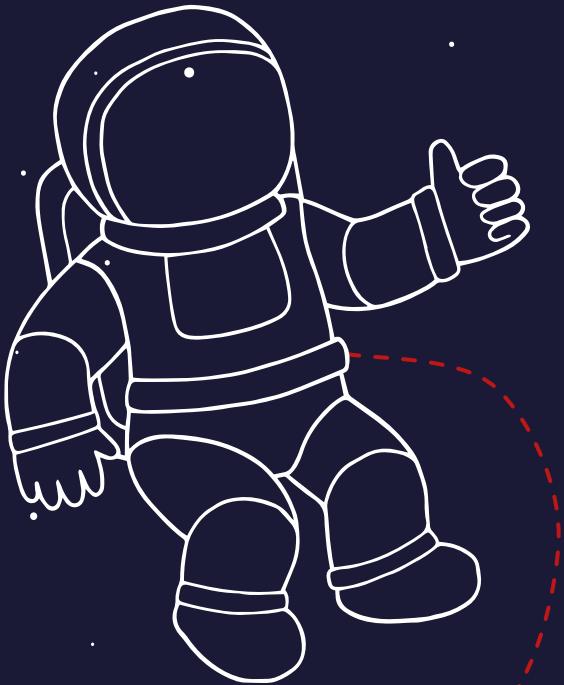


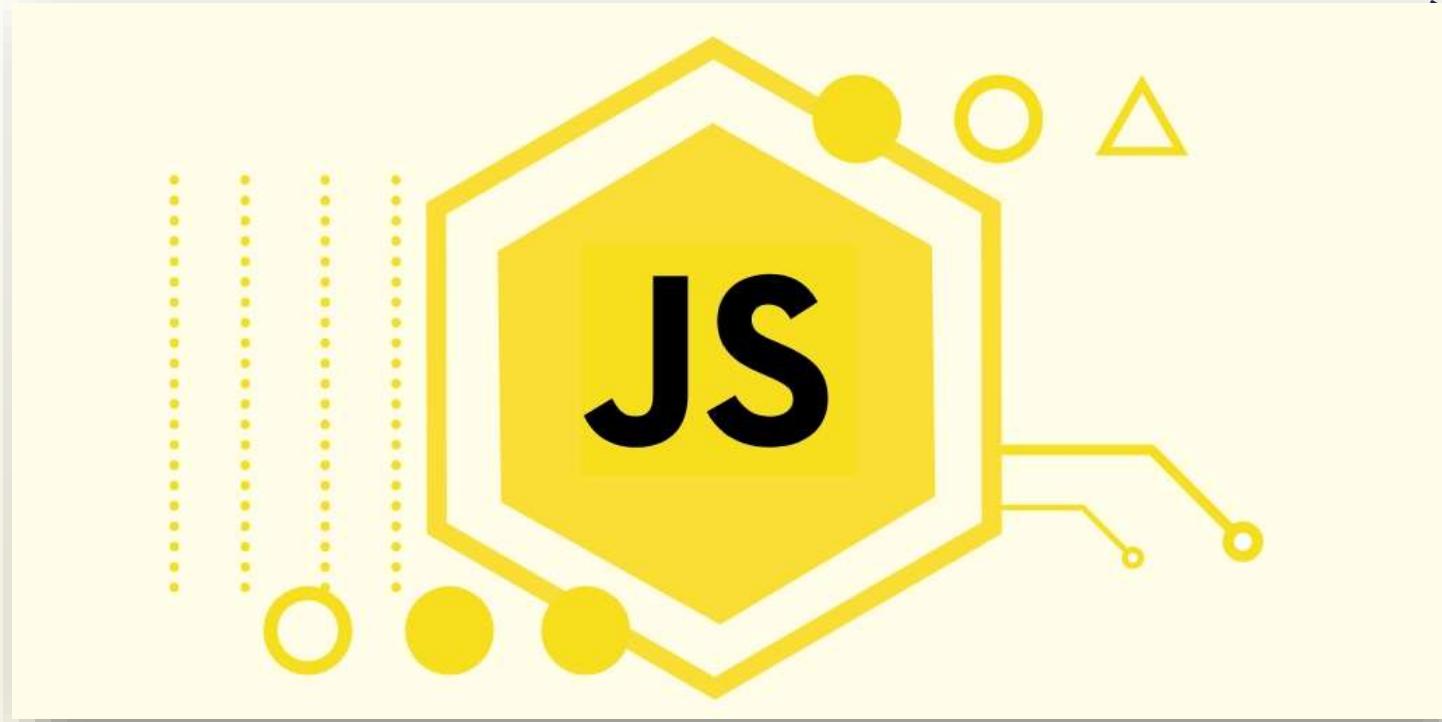


# Programa académico **CAMPUS**

**Ciclo 2:  
JAVASCRIPT  
INTRODUCCIÓN**



# JAVASCRIPT - INTRODUCCIÓN



JS

# JAVASCRIPT - INTRODUCCIÓN



- ¿Qué es JavaScript?
  - JavaScript fue creado para “*dar vida a las páginas web*”.
  - Los programas en este lenguaje se llaman scripts.



# JAVASCRIPT- INTRODUCCIÓN



# JAVASCRIPT - INTRODUCCIÓN



- Se pueden escribir directamente en el HTML de una página web y ejecutarse automáticamente a medida que se carga la página.
- 
- 

A screenshot of a code editor showing a snippet of JavaScript code. The code uses window.scroll() to scroll a page by its scrollHeight property, with a small adjustment of 0.02 added to element.clientHeight. The code is written in a monospaced font with color-coded syntax highlighting.

# JAVASCRIPT - INTRODUCCIÓN



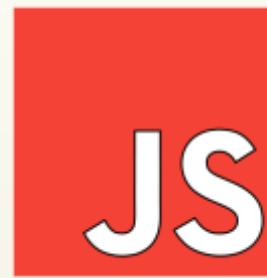
- ¿Por qué se llama JavaScript?



LiveScript



Java



JavaScript



# JAVASCRIPT - INTRODUCCIÓN



- ¿Por qué se llama JavaScript?
  - Pero a medida que evolucionaba, JavaScript se convirtió en un lenguaje completamente independiente con su propia especificación llamada **ECMAScript**, y ahora no tiene ninguna relación con Java.



# JAVASCRIPT - INTRODUCCIÓN



- ## ¿Qué es EmacScript?

- Es una especificación de lenguaje de programación publicada por Ecma International.



# JAVASCRIPT - INTRODUCCIÓN



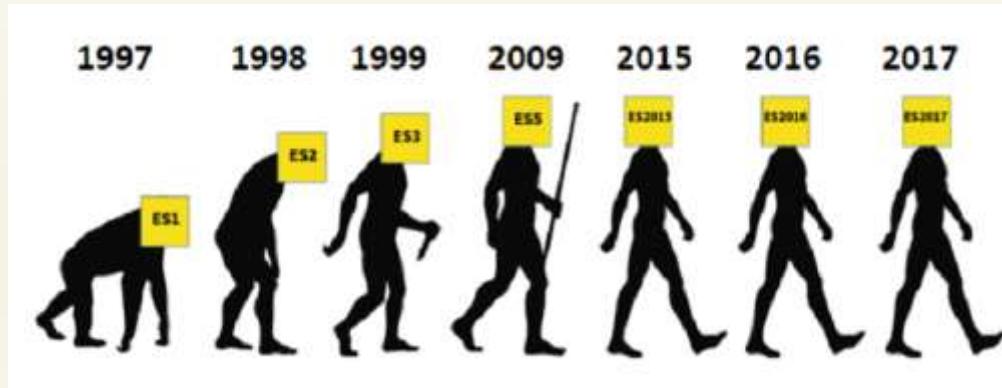
- Define un lenguaje de tipos dinámicos ligeramente inspirado en **Java** y otros lenguajes del estilo de **C**. Soporta algunas características de la programación orientada a objetos.



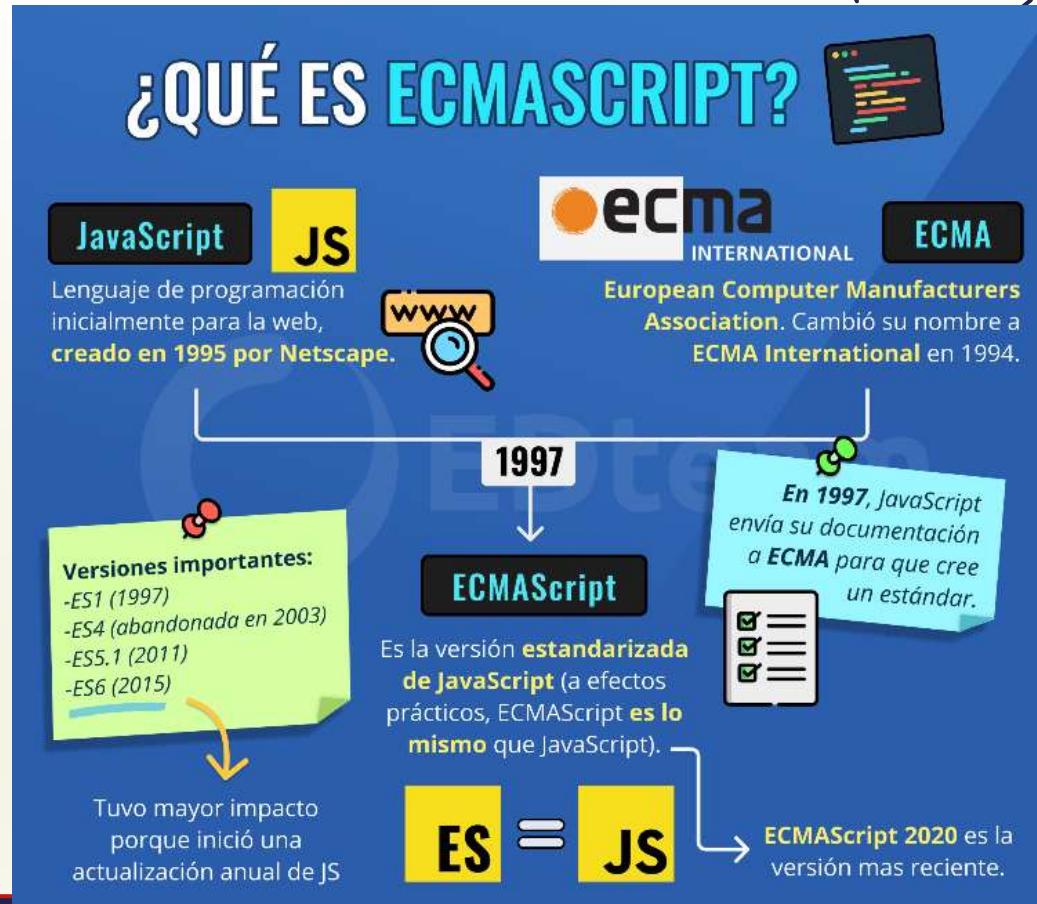
# JAVASCRIPT - INTRODUCCIÓN



- Define un lenguaje de tipos dinámicos ligeramente inspirado en **Java** y otros lenguajes del estilo de **C**. Soporta algunas características de la programación orientada a objetos.



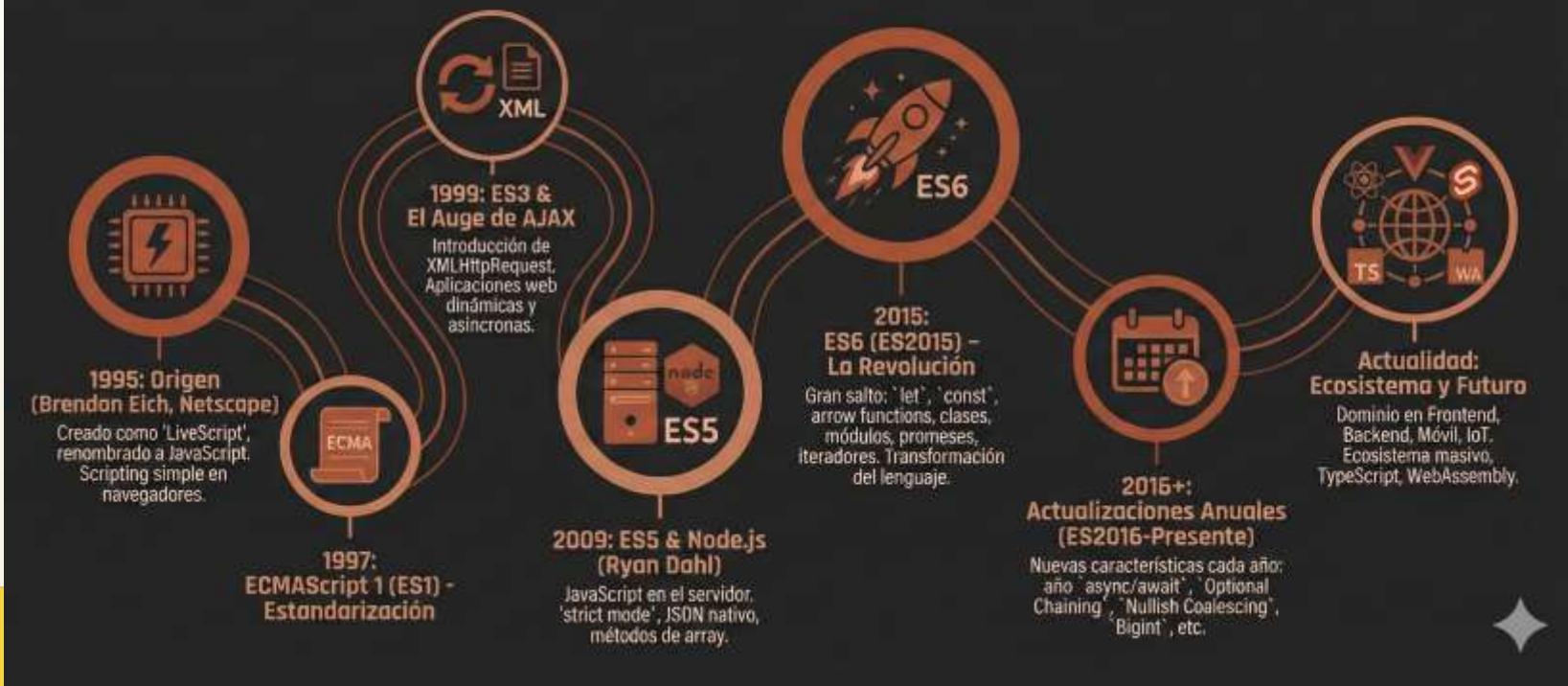
# JAVASCRIPT - INTRODUCCIÓN



# JAVASCRIPT - INTRODUCCIÓN



## LA EVOLUCIÓN DE JAVASCRIPT: De 'LiveScript' a la Web Moderna



JS



# JAVASCRIPT - INTRODUCCIÓN



## ECMA Script versions

ES1	Jun 1997
ES2	Jun 1998
ES3	Dec 1999
ES5	Dec 2009
ES5.1	Jun 2011



ES6	ES2015
ES7	ES2016
ES8	ES2017
ES9	ES2018
ES10	ES2019
ES11	ES2020
ES.Next	



# JAVASCRIPT - INTRODUCCIÓN



JS

# JAVASCRIPT - INTRODUCCIÓN



# JAVASCRIPT - INTRODUCCIÓN



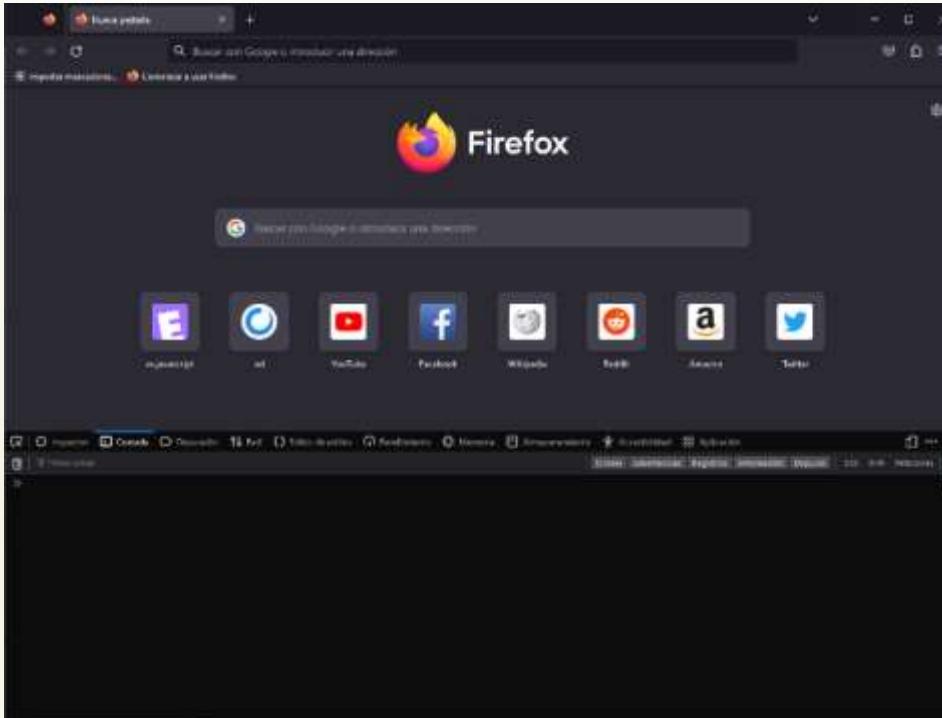
## Herramienta Navegador



# JAVASCRIPT - INTRODUCCIÓN



Presione las teclas F12



# JAVASCRIPT - INTRODUCCIÓN



Escriba en la consola:  $2 + 2$

A screenshot of a browser developer tools console. The top bar has tabs: Inspector, Consola (selected), Depurador, Red, and Editor de estilo. Below the tabs are icons for clear and filter. The console output shows:

```
» 2+2
← 4
»
```



# JAVASCRIPT - INTRODUCCIÓN



```
<!DOCTYPE HTML>
<html>

<head>
  <meta charset="utf-8">
</head>

<body>

  Hay un error en el script de esta
  página.
  <script>
    lalala
  </script>

</body>

</html>
```



# JAVASCRIPT - INTRODUCCIÓN



A screenshot of a browser's developer tools console. The tabs at the top are Inspector, Consola, Depurador, Red, Editor de estilos, and Rendimiento. Below the tabs, there is a toolbar with a trash can icon and a filter icon labeled "Filtrar salida". The main area shows an error message:

```
! Uncaught ReferenceError: lalala is not defined
<anonymous> file:///D:/GIT/GitLab/Campus LAB/Ciclo 2/JavaScript/ejemplos/01-error.html:12
[saber más]
```

At the bottom left of the console window, there is a double arrow icon followed by a vertical line.



# JAVASCRIPT - INTRODUCCIÓN



A screenshot of a browser developer tools interface showing the "Editor de estilos" tab selected. The left sidebar shows the "Fuentes" and "Contorno" sections, with the "Hilo principal" section expanded, showing the file structure: "file://", "D:/GIT/GitLab/Campus%20LAB/Ciclo%202/Jav", and "01-error.html". The "01-error.html" file is selected and displayed in the main editor area. The code is as follows:

```
1 <!DOCTYPE HTML>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6 </head>
7
8 <body>
9
10 Ha <s> ReferenceError: lalala is not defined </s> ina.
11 <s>
12 lalala
13 </script>
14
15 </body>
16
17 </html>
18
```

A red box highlights the error message "ReferenceError: lalala is not defined" in line 10, and a red underline is under the word "lalala" in line 12.

# JAVASCRIPT - INTRODUCCIÓN



## Herramientas online



# JAVASCRIPT - INTRODUCCIÓN



http://jsbin.com/



The screenshot shows the JSBin web-based code editor and execution environment. The interface includes tabs for HTML, CSS, JavaScript, Console, and Output, with the JavaScript tab currently active. The HTML pane contains a basic HTML document structure with a script tag. The JavaScript pane contains the code `console.log("Hola mundo")` and `console.log(2+3)`. The Console pane displays the results of these logs: 2+2, 4, 5, 5, and "Hola mundo". The Output pane is currently empty. A yellow bar at the bottom features the letters "JS".

File Add library

HTML CSS JavaScript Console Output

Login or Register Blog 1 Help

HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JS Bin</title>
</head>
<body>
  <script>
    console.log(2+3)
  </script>
</body>
</html>
```

JavaScript

```
console.log("Hola mundo")
```

Console

Clear

2+2  
4  
5  
5  
"Hola mundo"

Output

Run with JS Auto-run JS

JS

# JAVASCRIPT - INTRODUCCIÓN



<https://jsfiddle.net>

A screenshot of the jsFiddle.net interface. On the left, there's a sidebar with "Fiddle meta" (name: Holamundo, private fiddle selected), "Groups" (empty), "Resources" (URL: [click](#)), "Async requests" (empty), and "Other (links, license)". Below the sidebar is an advertisement for "VPS SSD" with a price of \$4.20. The main area has tabs for "HTML", "JavaScript" (containing `console.log("Holamundo")`), and "CSS" (containing `body { background-color: #ff0000; }`). The preview window is empty. At the bottom, there's a developer toolbar with tabs like "Elementos", "Consola", "Fuentes", "Red", "Rendimiento", "Memoria", and "Niveles predeterminados" set to 70. The "Consola" tab shows the output: "Se ha cerrado la conexión" and "Hola mundo".

HTML

```
<body></body>
```

JavaScript - No Library (pure JS)

```
console.log("Holamundo")
```

CSS

```
body { background-color: #ff0000; }
```

Fiddle listing shows latest 10 fiddles

Holamundo

run description

private fiddle

Groups

Resources URL: [click](#)

Async requests

Other (links, license)

VPS SSD

Definito de un VPS por solo \$4.20 con un ancho de banda de 25Gb/s

Elementos Consola Fuentes Red Rendimiento Memoria Niveles predeterminados 70

Se ha cerrado la conexión

Hola mundo



# JAVASCRIPT - INTRODUCCIÓN



<https://codepen.io>

A screenshot of the CodePen interface. The top navigation bar shows the file name "holamundo-js.html". The interface is divided into three panels: HTML, CSS, and JS. The HTML panel contains the code: 

```
<div> Hola mundo!</div>
```

. The CSS panel contains the code: 

```
body { font-family: system-ui; background: #F0F0E6; color: white; text-align: center; font-size: 2em; } 
```

. The JS panel contains the code: 

```
console.log("Hola mundo"); console.log(2 + 2);
```

. Below the panels, the preview area shows a yellow hand icon and the text "Hola mundo!". At the bottom, the console output shows the result: "Hola mundo". A small yellow sidebar on the left features a person icon and the letters "JS".

# JAVASCRIPT - INTRODUCCIÓN



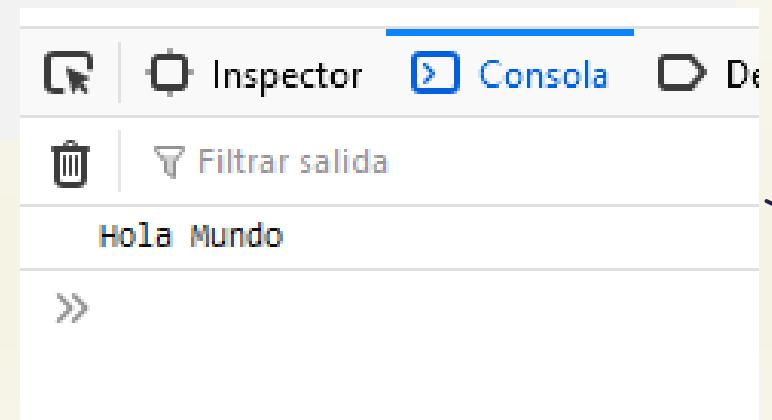
Hola mundo  
Y la etiqueta <script>



# JAVASCRIPT - INTRODUCCIÓN



- <body>
- <script>
- console.log('Hola Mundo');
- </script>
- </body>



# JAVASCRIPT - INTRODUCCIÓN



## Script externos



# JAVASCRIPT - INTRODUCCIÓN



```
<body>
  <script src="js/01-script.js"></script>
</body>
```

```
js > JS 01-script.js
  1 console.log('Hola mundo');
  2
```



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Escribe un programa por consola del navegador en Javascript que realice el siguiente patrón:

```
*  
**  
****  
*****
```



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Escribe un programa por consola del navegador en JavaScript que realice el siguiente patrón:

```
*  
**  
****  
*****
```



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Escribe un programa en JavaScript un script interno de HTML que realice el siguiente patrón:

```
*****
```

```
*      *
```

```
*      *
```

```
*      *
```

```
*****
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Escribe un programa en JavaScript un script externo de HTML que realice el siguiente patrón:

```
*      *****
 *  *      *
 *  *
 *  *      *****
 *  *      *
 *  *  *      *
 ****      *****
```



# JAVASCRIPT - INTRODUCCIÓN



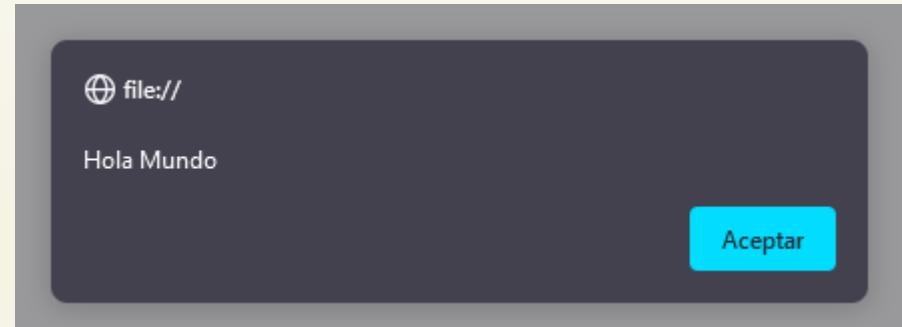
**alert**



# JAVASCRIPT - INTRODUCCIÓN



```
<script>
  alert('Hola Mundo');
</script>
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Comentarios



# JAVASCRIPT - INTRODUCCIÓN



```
/* Un ejemplo con dos mensajes.  
Este es un comentario multilínea.  
*/  
alert('Hola');  
alert('Mundo');
```



# JAVASCRIPT - INTRODUCCIÓN



- ```
<script>
    // Este comentario ocupa una Línea
propia.
```
- ```
    alert('Hola');
```
- ```
    alert('mundo'); // Este comentario sigue
a La sentencia.
```
- ```
</script>
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Variables y tipos de datos



# JAVASCRIPT - INTRODUCCIÓN



## Tipado Débil y Dinámico

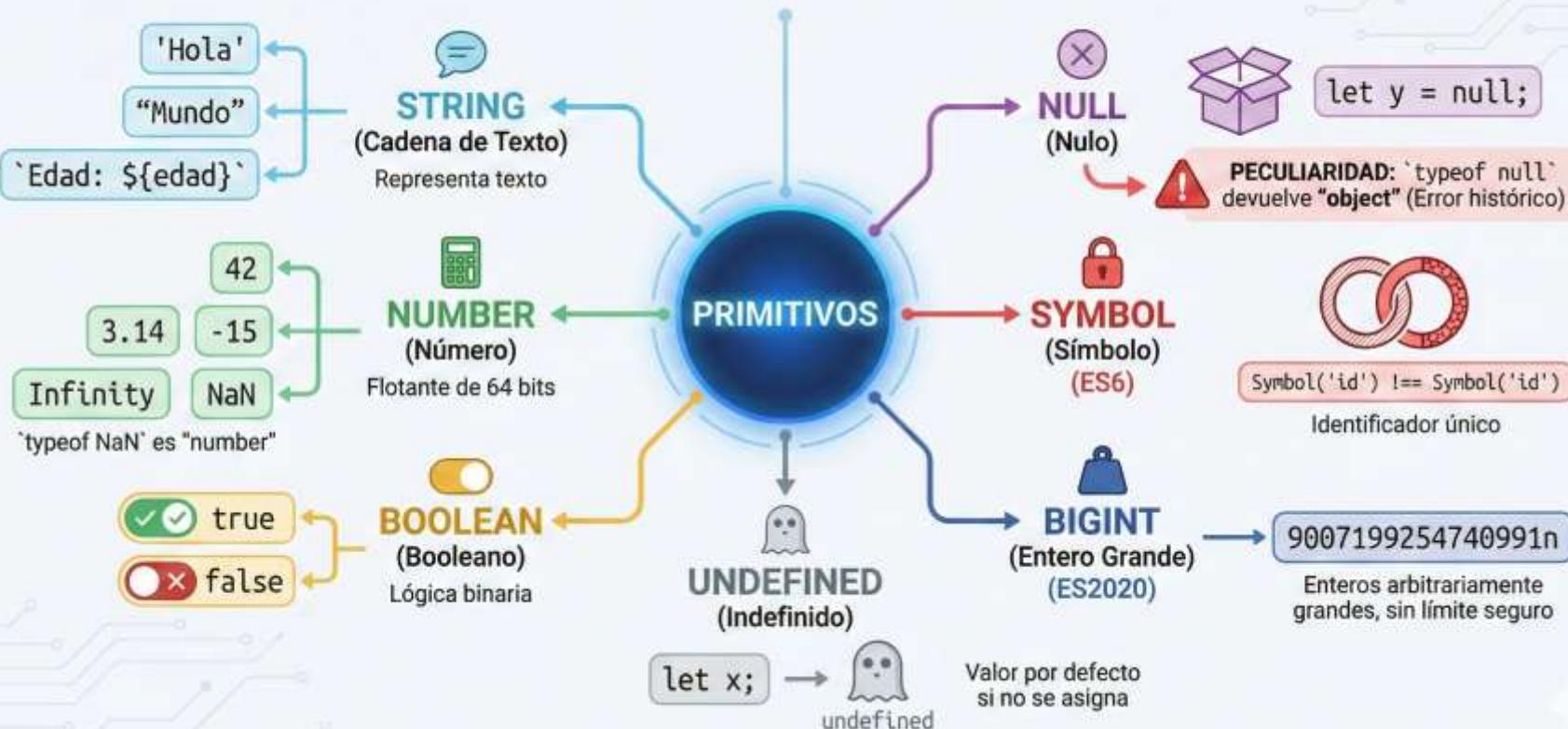


A screenshot of a browser's developer tools console. The tabs at the top are 'Elements' and 'Console'. The 'Console' tab is active, showing the following interactions:

```
> x = 32;
< 32
> x;
< 32
> whereAmI = "Santa Barbara, CA";
< "Santa Barbara, CA"
> x = 45;
< 45
> whereAmI = "Los Angeles"
< "Los Angeles"
> whereAmI = 75
< 75
> |
```

# TIPOS DE DATOS PRIMITIVOS DE JAVASCRIPT

Son inmutables y no son objetos



# JAVASCRIPT - INTRODUCCIÓN



## TIPOS DE DATOS PRIMITIVOS EN JAVASCRIPT

JS

Los primitivos **son los tipos de dato más básicos en JavaScript**. Son inmutables y contienen un único valor.

NUMBER

Para almacenar números.

```
let year = 2020;
```

UNDEFINED

Existe cuando **no se le asigna un valor** a una variable.



```
let b
```

```
let isPremium = true;
```

STRING

Se usa para **representar texto** (con comillas: simples ', dobles "" o acentos invertidos ` `).

```
let name = "EDteam";
```

NULL



Es cuando un dato **no existe**.

```
if (userName !== null)  
    alert(`Bienvenido  
${userName}`)
```

BOOLEAN

Almacena un **valor lógico**, puede ser **true** o **false**.



JS

# JAVASCRIPT - INTRODUCCIÓN



## DECLARACION DE VARIABLES

- let
- var
- const

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



# JAVASCRIPT - INTRODUCCIÓN



A screenshot of a browser developer tools interface showing the "Consola" tab selected. The console output is as follows:

```
» let mensaje;
← undefined
» console.log(typeof(mensaje))
undefined
← undefined
»
```

# JAVASCRIPT - INTRODUCCIÓN



Inspector



Consola



Depurador



Filtrar salida

```
» mensaje = 'Hola'; //Tipo cadena
```

```
← "Hola"
```

```
» console.log(typeof(mensaje));
```

```
string
```

```
← undefined
```

```
» console.log(mensaje)
```

```
Hola
```

```
← undefined
```



JS

# JAVASCRIPT - INTRODUCCIÓN



Inspector



Consola



Depurador



Red



Filtrar salida

```
» let user = 'John', age = 25, mensaje2 = 'Hola';
← undefined
» console.log(typeof(user) + " " + typeof(mensaje2));
      string string
← undefined
» console.log(typeof(age));
      number
← undefined
»
```



JS

# JAVASCRIPT - INTRODUCCIÓN



```
1 let user = 'John', age = 25, message = 'Hola';
```

La versión de líneas múltiples es un poco más larga, pero se lee más fácil:

```
1 let user = 'John';
2 let age = 25;
3 let message = 'Hola';
```

Algunas personas también definen variables múltiples en estilo multilínea:

```
1 let user = 'John',
2     age = 25,
3     message = 'Hola';
```



# JAVASCRIPT - INTRODUCCIÓN



## Number

```
1 let n = 123;  
2 n = 12.345;
```

El tipo *number* representa tanto números enteros como de punto flotante.

`Infinity` representa el Infinito matemático  $\infty$ . Es un valor especial que es mayor que cualquier número.

Podemos obtenerlo como resultado de la división por cero:

```
1 alert( 1 / 0 ); // Infinity
```



JS

# JAVASCRIPT - INTRODUCCIÓN



`Nan` representa un error de cálculo. Es el resultado de una operación matemática incorrecta o indefinida, por ejemplo:

```
1 alert( "no es un número" / 2 ); // Nan, tal división es errónea
```



`Nan` es "pegajoso". Cualquier otra operación sobre `Nan` devuelve `Nan`:

```
1 alert( NaN + 1 ); // Nan
2 alert( 3 * NaN ); // Nan
3 alert( "not a number" / 2 - 1 ); // Nan
```



Por lo tanto, si hay un `Nan` en alguna parte de una expresión matemática, se propaga a todo el resultado (con una única excepción: `Nan ** 0` es `1`).



JS

# JAVASCRIPT - INTRODUCCIÓN



`Nan` representa un error de cálculo. Es el resultado de una operación matemática incorrecta o indefinida, por ejemplo:

```
1 alert( "no es un número" / 2 ); // Nan, tal división es errónea
```



`Nan` es "pegajoso". Cualquier otra operación sobre `Nan` devuelve `Nan`:

```
1 alert( NaN + 1 ); // Nan
2 alert( 3 * NaN ); // Nan
3 alert( "not a number" / 2 - 1 ); // Nan
```



Por lo tanto, si hay un `Nan` en alguna parte de una expresión matemática, se propaga a todo el resultado (con una única excepción: `Nan ** 0` es `1`).



JS

# JAVASCRIPT - INTRODUCCIÓN



## BigInt

En JavaScript, el tipo "number" no puede representar de forma segura valores enteros mayores que  $(2^{53}-1)$  (eso es 9007199254740991), o menor que  $-(2^{53}-1)$  para negativos.

Para ser realmente precisos, el tipo de dato "number" puede almacenar enteros muy grandes (hasta  $1.7976931348623157 * 10^{308}$ ), pero fuera del rango de enteros seguros  $\pm(2^{53}-1)$  habrá un error de precisión, porque no todos los dígitos caben en el almacén fijo de 64-bit. Así que es posible que se almacene un valor "aproximado".



JS

# JAVASCRIPT - INTRODUCCIÓN



- **BigInt** se agregó recientemente al lenguaje para representar enteros de longitud arbitraria.

- Un valor `BigInt` se crea agregando `n` al final de un entero:

```
1 // la "n" al final significa que es un BigInt  
2 const bigInt = 1234567890123456789012345678901234567890n;
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Browser compatibility

[Report problems with this compatibility data on GitHub](#) ↗

|        | Chrome | Edge | Firefox | Opera | Safari | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | WebView Android | Deno | Node.js |
|--------|--------|------|---------|-------|--------|----------------|---------------------|---------------|---------------|------------------|-----------------|------|---------|
| BigInt | ✓      | ✓    | ✓       | ✓     | ✓      | ✓              | ✓                   | ✓             | ✓             | ✓                | ✓               | ✓    | ✓       |
|        | 67     | 79   | 68      | 54    | 14     | 67             | 68                  | 48            | 14            | 9.0              | 67              | 1.0  | 10.4.0  |



JS

# JAVASCRIPT - INTRODUCCIÓN



## String

Un *string* en JavaScript es una cadena de caracteres y debe colocarse entre comillas.

```
1 let str = "Hola";
2 let str2 = 'Las comillas simples también están bien';
3 let phrase = `se puede incrustar otro ${str}`;
```

En JavaScript, hay 3 tipos de comillas.



JS

# JAVASCRIPT - INTRODUCCIÓN



- En JavaScript, hay 3 tipos de comillas.

- 1. Comillas dobles: "Hola".
- 2. Comillas simples: 'Hola'.
- 3. Backticks (comillas invertidas): `Hola`.

- Las comillas **dobles** y **simples** son comillas “**sencillas**” (es decir, funcionan igual). No hay diferencia entre ellas en JavaScript.



JS

# JAVASCRIPT - INTRODUCCIÓN



- Los backticks son comillas de “funcionalidad extendida”. Nos permiten incrustar variables y expresiones en una cadena de caracteres encerrándolas en  `${...}`, por ejemplo:

```
1 let name = "John";
2
3 // incrustar una variable
4 alert(`Hola, ${name}!`); // Hola, John!
5
6 // incrustar una expresión
7 alert(`el resultado es ${1 + 2}`); //el resultado es 3
```



La expresión dentro de  `${...}` se evalúa y el resultado pasa a formar parte de la cadena. Podemos poner cualquier cosa ahí dentro: una variable como `name`, una expresión aritmética como `1 + 2`, o algo más complejo.



JS

# JAVASCRIPT - INTRODUCCIÓN



## Boolean (tipo lógico)

El tipo *boolean* tiene sólo dos valores posibles: `true` y `false`.

Este tipo se utiliza comúnmente para almacenar valores de sí/no: `true` significa "sí, correcto, verdadero", y `false` significa "no, incorrecto, falso".

Por ejemplo:

```
1 let nameFieldChecked = true; // sí, el campo name está marcado  
2 let ageFieldChecked = false; // no, el campo age no está marcado
```

```
1 let isGreater = 4 > 1;  
2  
3 alert( isGreater ); // verdadero (el resultado de la comparación es "sí")
```



# JAVASCRIPT - INTRODUCCIÓN



## • El valor “null” (nulo)

El valor especial `null` no pertenece a ninguno de los tipos descritos anteriormente.

Forma un tipo propio separado que contiene sólo el valor `null`:

```
1 let age = null;
```

En JavaScript, `null` no es una “referencia a un objeto inexistente” o un “puntero nulo” como en otros lenguajes.

Es sólo un valor especial que representa “nada”, “vacío” o “valor desconocido”.

El código anterior indica que el valor de `age` es desconocido o está vacío por alguna razón.



JS

# JAVASCRIPT - INTRODUCCIÓN



## El valor “undefined” (indefinido)

- El valor especial `undefined` también se distingue. Hace un tipo propio, igual que `null`.
- El significado de `undefined` es “valor no asignado”.
- Si una variable es declarada, pero no asignada, entonces su valor es `undefined`:

```
1 let age;  
2  
3 alert(age); // muestra "undefined"
```

```
1 let age = 100;  
2  
3 // cambiando el valor a undefined  
4 age = undefined;  
5  
6 alert(age); // "undefined"
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Object y Symbol

El tipo `object` (objeto) es especial.

Todos los demás tipos se llaman “primitivos” porque sus valores pueden contener una sola cosa (ya sea una cadena, un número o lo que sea). Por el contrario, los objetos se utilizan para almacenar colecciones de datos y entidades más complejas.



JS

# JAVASCRIPT - INTRODUCCIÓN



## El operador `typeof`

El operador `typeof` devuelve el tipo de dato del operando. Es útil cuando queremos procesar valores de diferentes tipos de forma diferente o simplemente queremos hacer una comprobación rápida.



# JAVASCRIPT - INTRODUCCIÓN



```
1  typeof undefined // "undefined"
2
3  typeof 0 // "number"
4
5  typeof 10n // "bigint"
6
7  typeof true // "boolean"
8
9  typeof "foo" // "string"
10
11 typeof Symbol("id") // "symbol"
12
13 typeof Math // "object" (1)
14
15 typeof null // "object" (2)
16
17 typeof alert // "function" (3)
```



# JAVASCRIPT - INTRODUCCIÓN



## Tareas

### Comillas ↗

importancia: 5

¿Cuál es la salida del script?

```
1 let name = "Ilya";
2
3 alert(`Hola ${1}`);
4 // ?
5 alert(`Hola ${"name"} `);
6 // ?
7 alert(`Hola ${name}`); // ?
```



JS

# JAVASCRIPT - INTRODUCCIÓN



**let vs var vs const**

DIFFERENCE



JS

# JAVASCRIPT - INTRODUCCIÓN



**i** `var` en vez de `let`

En scripts más viejos, a veces se encuentra otra palabra clave: `var` en lugar de `let`:

```
1 var mensaje = 'Hola';
```



JS

# JAVASCRIPT - INTRODUCCIÓN



- **“var” no tiene alcance (visibilidad) de bloque.**

Las variables declaradas con `var` pueden tener a la función como entorno de visibilidad, o bien ser globales. Su visibilidad atraviesa los bloques.

- Por ejemplo:

```
1 if (true) {  
2     var test = true; // uso de "var" en lugar de "let"  
3 }  
4  
5 alert(test); // true, la variable vive después del if
```



JS

# JAVASCRIPT - INTRODUCCIÓN



Si usáramos `let test` en vez de `var test`, la variable sería visible solamente dentro del `if`:

```
1 if (true) {  
2     let test = true; // uso de "let"  
3 }  
4  
5 alert(test); // ReferenceError: test no está definido
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## “var” tolera redeclaraciones

Declarar la misma variable con `let` dos veces en el mismo entorno es un error:

```
1 let user;  
2 let user; // SyntaxError: 'user' ya fue declarado
```



# JAVASCRIPT - INTRODUCCIÓN



Con `var` podemos redeclarar una variable muchas veces. Si usamos `var` con una variable ya declarada, simplemente se ignora:

```
1 var user = "Pete";
2
3 var user = "John"; // este "var" no hace nada (ya estaba declarado)
4 // ...no dispara ningún error
5
6 alert(user); // John
```



JS

# JAVASCRIPT - INTRODUCCIÓN



- Las variables “var” pueden ser declaradas debajo del lugar en donde se usan

```
1 function sayHi() {  
2     phrase = "Hello";  
3  
4     alert(phrase);  
5  
6     var phrase;  
7 }  
8 sayHi();
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Resumen

- Hay dos diferencias principales entre `var` y `let/const`:
  1. Las variables `var` no tienen alcance de bloque: su visibilidad alcanza a la función, o es global si es declarada fuera de las funciones.
  2. Las declaraciones `var` son procesadas al inicio de la función (o del script para las globales) .



JS

# JAVASCRIPT - INTRODUCCIÓN



## ¿Cuándo usar JavaScript var?

Declare siempre las variables de JavaScript con `var`, `let` o `const`.

La `var` palabra clave se usa en todo el código JavaScript desde 1995 hasta 2015.

Las palabras clave `let` y `const` se agregaron a JavaScript en 2015.

Si desea que su código se ejecute en navegadores más antiguos, debe usar `var`.



JS

# JAVASCRIPT - INTRODUCCIÓN



## Constantes

Para declarar una variable constante (inmutable) use `const` en vez de `let`:

```
1 const myBirthday = '18.04.1982';
```



JS

# JAVASCRIPT - INTRODUCCIÓN



Las variables declaradas utilizando `const` se llaman "constantes". No pueden ser alteradas. Al intentarlo causaría un error:

```
1 const myBirthday = '18.04.1982';
2
3 myBirthday = '01.01.2001'; // ¡error, no se puede reasignar la constante!
```



# JAVASCRIPT - INTRODUCCIÓN



## Constantes mayúsculas

- Existe una práctica utilizada ampliamente de utilizar constantes como alias de valores difíciles-de-recordar y que se conocen previo a la ejecución.
- Tales constantes se nombran utilizando letras mayúsculas y guiones bajos.



JS

# JAVASCRIPT - INTRODUCCIÓN



Por ejemplo, creamos constantes para los colores en el formato "web" (hexadecimal):

```
1 const COLOR_RED = "#F00";
2 const COLOR_GREEN = "#0F0";
3 const COLOR_BLUE = "#00F";
4 const COLOR_ORANGE = "#FF7F00";
5
6 // ...cuando debemos elegir un color
7 let color = COLOR_ORANGE;
8 alert(color); // #FF7F00
```



# JAVASCRIPT - INTRODUCCIÓN



Por ejemplo, creamos constantes para los colores en el formato "web" (hexadecimal):

```
1 const COLOR_RED = "#F00";
2 const COLOR_GREEN = "#0F0";
3 const COLOR_BLUE = "#00F";
4 const COLOR_ORANGE = "#FF7F00";
5
6 // ...cuando debemos elegir un color
7 let color = COLOR_ORANGE;
8 alert(color); // #FF7F00
```



# JAVASCRIPT - INTRODUCCIÓN



## Ventajas:

- `COLOR_ORANGE` es mucho más fácil de recordar que `"#FF7F00"`.
- Es mucho más fácil escribir mal `"#FF7F00"` que `COLOR_ORANGE`.
- Al leer el código, `COLOR_ORANGE` tiene mucho más significado que `#FF7F00`.



# JAVASCRIPT - INTRODUCCIÓN



## JavaScript y sus variables var, let y const

| Propiedad       | var     | let    | const  |
|-----------------|---------|--------|--------|
| scope (alcance) | función | bloque | bloque |
| re-asignación   | ✓       | ✓      | ✗      |
| re-declaración  | ✓       | ✗      | ✗      |



JS

# JAVASCRIPT - INTRODUCCIÓN



## OPERADORES



# JAVASCRIPT - INTRODUCCIÓN



## OPERADORES NUMÉRICOS

| Operador | Descripción    |
|----------|----------------|
| +        | Suma           |
| -        | Resta          |
| *        | Multiplicación |
| /        | División       |
| %        | Residuo        |
| ++       | Incremento     |
| --       | Decremento     |

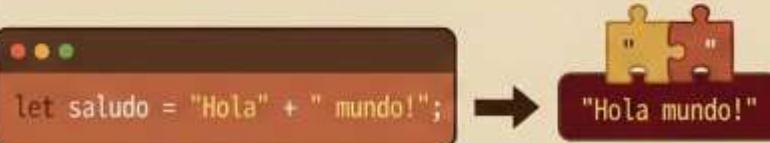


# OPERADORES DE CADENAS (STRINGS) EN JAVASCRIPT

## Uniendo y modificando texto

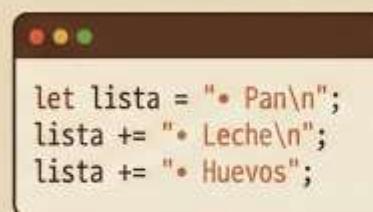
### 1. OPERADOR DE CONCATENACIÓN (+)

Une dos o más cadenas para formar una nueva.  
Realiza coerción de tipos si se usa con no-cadenas.



### 2. OPERADOR DE ASIGNACIÓN DE CONCATENACIÓN (+=)

Añade el valor de la derecha al final de la variable de cadena existente.



JS

NOTA IMPORTANTE: JavaScript convierte automáticamente números u otros tipos a cadenas cuando se usa '+' con una cadena.

# JAVASCRIPT - INMUTABILIDAD CADENAS



## STRINGS EN JAVASCRIPT: INMUTABILIDAD

The screenshot shows a browser's developer tools console with the title "F12 - Chrome DevTools". The code examples demonstrate that strings are immutable in JavaScript:

```
let mensaje = "Hola Mundo";
let mensaje = "Hola Mundo";
let mensaje;
let mensaje = "Adios";
let mensaje = "Hola Mundo";
```

The console output shows the original string "Hola Mundo" followed by nine identical entries of the mutated string "Hola Mundo", illustrating that the original string was not modified.

Veo



# MÁS OPERACIONES Y MÉTODOS DE CADENAS (STRINGS) EN JAVASCRIPT

## 3. REPETICIÓN (.repeat())

Crea y devuelve una nueva cadena que contiene el número especificado de copias de la cadena original.

```
let risa = "Ja";
let carcajada = risa.repeat(3);
```



"JaJaJa"

## 5. LONGITUD Y BÚSQUEDA (.length y .includes())

.length devuelve el número de caracteres,  
.includes() verifica si una subcadena existe.

```
let texto = "JavaScript";
let longitud = texto.length;
```

JavaScript  
10

```
let contiene = texto.includes("Script");
```



true

## 4. INDEXACIÓN Y ACCESO ([] y .charAt())

Accede a un carácter individual en una posición (índice) específica. Los índices empiezan en 0.

```
let palabra = "Hola";
let segundaLetra = palabra[1];
```



"o"

```
let primeraLetra = palabra.charAt(0);
```



"H"

## 6. EXTRACCIÓN (.slice())

Extrae una sección de una cadena y devuelve una cadena nueva, desde el índice inicial hasta (pero sin incluir) el final.

```
let completo = "Hola Mundo";
let extraido = completo.slice(0, 4);
```



"Hola"

# LOS 10 MÉTODOS DE CADENA (STRING) MÁS USADOS EN JAVASCRIPT

## 1. .length (Propiedad)

Devuelve la longitud de la cadena.

"Hola".length ➔ 4



## 2. .toUpperCase()

Convierte la cadena a mayúsculas.



"mundo".toUpperCase() ➔ "MUNDO"

## 3. .toLowerCase()

Convierte la cadena a minúsculas.



"JAVA".toLowerCase() ➔ "java"

## 4. .includes()

Comprueba si contiene una subcadena.



"Equipo".includes("quip") ➔ true

## 5. .indexOf()

Devuelve el índice de la primera aparición de una subcadena.



"Banana".indexOf("na") ➔ 2

## 6. .slice()

Extrae una parte de la cadena.



"Cangrejo".slice(0, 4) ➔ "Cang"

## 7. .substring()

Similar a slice, pero con reglas diferentes para índices negativos.



"Cangrejo".substring(3, 6) ➔ "gre"

## 8. .replace()

Reemplaza la primera aparición de una subcadena.



"Hola Bob".replace("Bob", "Ana") ➔ "Hola Ana"

## 9. .trim()

Elimina espacios en blanco de ambos extremos.



" Texto ".trim() ➔ "Texto"

## 10. .split()

Divide la cadena en un array de subcadenas.



"a,b,c".split(",") ➔ ["a", "b", "c"]

# JAVASCRIPT - INTRODUCCIÓN

## OPERADORES NUMÉRICOS



| Operador            | Descripción  | Ejemplo  |
|---------------------|--|--|
| Resto (%)           | Operador binario correspondiente al módulo de una operación. Devuelve el resto de la división de dos operandos.  | 12 % 5 devuelve 2.   |
| Incremento (++)     | Operador unario. Incrementa en una unidad al operando. Si es usado antes del operando (++x) devuelve el valor del operando después de añadirle 1 y si se usa después del operando (x++) devuelve el valor de este antes de añadirle 1. | Si x es 3, entonces ++x establece x a 4 y devuelve 4, mientras que x++ devuelve 3 y, solo después de devolver el valor, establece x a 4. |
| Decremento (--)     | Operador unario. Resta una unidad al operando. Dependiendo de la posición con respecto al operando tiene el mismo comportamiento que el operador de incremento.  | Si x es 3, entonces --x establece x a 2 y devuelve 2, mientras que x-- devuelve 3 y, solo después de devolver el valor, establece x a 2. |
| Negación Unaria (-) | Operación unaria. Intenta convertir a número al operando y devuelve su forma negativa.   | -"3" devuelve -3.<br>-true devuelve -1.  |
| Unario positivo (+) | Operación unaria. Intenta convertir a número al operando.  | + "3" devuelve 3.<br>+ true devuelve 1.  |
| Exponenciación (**) | Calcula la potencia de la base al valor del exponente. Es equivalente a $\text{base}^{\text{exponente}}$   | 2 ** 3 devuelve 8.<br>10 ** -1 devuelve 0.1.   |



# JAVASCRIPT - INTRODUCCIÓN



## OPERADORES LÓGICOS

| Operador                                  | Uso                                 | Descripción  |
|---|-------------------------------------|--|
| AND Lógico<br>( <code>&amp;&amp;</code> ) | <code>expr1 &amp;&amp; expr2</code> | Devuelve <code>expr1</code> si puede ser convertido a <code>false</code> de lo contrario devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>&amp;&amp;</code> devuelve <code>true</code> si ambos operandos son <code>true</code> , en caso contrario devuelve <code>false</code> .      |
| OR Lógico<br>( <code>  </code> )          | <code>expr1    expr2</code>         | Devuelve <code>expr1</code> si puede ser convertido a <code>true</code> de lo contrario devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>  </code> devuelve <code>true</code> si alguno de los operandos es <code>true</code> , o <code>false</code> si ambos son <code>false</code> . |
| NOT Lógico<br>( <code>!</code> )          | <code>!expr</code>                  | Devuelve <code>false</code> si su operando puede ser convertido a <code>true</code> , en caso contrario, devuelve <code>true</code> .  |



# JAVASCRIPT - INTRODUCCIÓN



## OPERADORES LÓGICOS-RELACIONALES



Igualdad

== == == == == == = == ==

Desigualdad

!= != != != != != <> != !=

o

or || || || || || Or || or

y

and && && && && && And && and

Negación

not ! ! ! ! ! Not ! !



JS

# JAVASCRIPT - INTRODUCCIÓN



## OPERADORES DE ASIGNACIÓN

| Nombre                       | Operador abreviado | Significado  |
|------------------------------|--------------------|--------------|
| Operadores de asignación     | $x = y$            | $x = y$      |
| Asignación de adición        | $x += y$           | $x = x + y$  |
| Asignación de sustracción    | $x -= y$           | $x = x - y$  |
| Asignación de multiplicación | $x *= y$           | $x = x * y$  |
| Asignación de división       | $x /= y$           | $x = x / y$  |
| Asignación de resto          | $x %= y$           | $x = x \% y$ |
| Asignación de exponenciación | $x **= y$          | $x = x ** y$ |



# JAVASCRIPT - INTRODUCCIÓN

## OPERADORES DE COMPARACIÓN

| Operador                       | Descripción   | Ejemplos devolviendo true  |
|--------------------------------|---|--|
| Igualdad (==)                  | Devuelve true si ambos operandos son iguales.   | <code>3 === var1</code><br><code>"3" === var1</code><br><code>3 === "3"</code> |
| Desigualdad (!=)               | Devuelve true si ambos operandos no son iguales.  | <code>var1 != 4</code><br><code>var2 != "3"</code>                             |
| Estrictamente iguales (===)    | Devuelve true si los operandos son igual y tienen el mismo tipo.<br>Mira también <code>Object.is</code> y <code>sameness in JS</code> . | <code>3 === var1</code>  |
| Estrictamente desiguales (!==) | Devuelve true si los operandos no son iguales y/o no son del mismo tipo.  | <code>var1 !== "3"</code><br><code>3 !== "3"</code>                            |
| Mayor que (>)                  | Devuelve true si el operando de la izquierda es mayor que el operando de la derecha.  | <code>var2 &gt; var1</code><br><code>"12" &gt; 2</code>                        |
| Mayor o igual que (>=)         | Devuelve true si el operando de la izquierda es mayor o igual que el operando de la derecha.  | <code>var2 &gt;= var1</code><br><code>var1 &gt;= 3</code>                      |
| Menor que (<)                  | Devuelve true si el operando de la izquierda es menor que el operando de la derecha.  | <code>var1 &lt; var2</code><br><code>"2" &lt; 12</code>                        |
| Menor o igual que (<=)         | Devuelve true si el operando de la izquierda es menor o igual que el operando de la derecha.  | <code>var1 &lt;= var2</code><br><code>var2 &lt;= 5</code>                      |



# JAVASCRIPT - INTRODUCCIÓN



## Los valores "falsy" en JavaScript

1. `false`: el valor booleano falso.
2. `0`: el número cero.
3. `-0`: el número cero negativo.
4. `0n`: el BigInt cero.
5. `""`: la cadena de texto vacía.
6. `null`: un valor nulo.
7. `undefined`: un valor no definido.
8. `NaN`: un valor que representa "Not a Number".



# JAVASCRIPT - INTRODUCCIÓN



Cualquier otra cosa que no sea "falsy" es "truthy":

1. `true`: el valor booleano verdadero.
2. `1`: cualquier número diferente de cero se considera verdadero.
3. `"false"`: cualquier cadena de texto no vacía se considera verdadera.
4. `[]`: un arreglo vacío se considera verdadera.
5. `{}`: cualquier objeto vacío se considera verdadero.
6. `function () {}`: cualquier función definida se considera verdadera.
7. `new Date ()`: cualquier objeto de fecha se considera verdadero.
8. `42n`: cualquier BigInt diferente de cero se considera verdadero.



# JAVASCRIPT - INTRODUCCIÓN



Los valores "Nullish" son  
**null** y **undefined**



# JAVASCRIPT - INTRODUCCIÓN



## OPERADORES DE CADENAS

- Operador concatenación
- Template String o Template Literals
- Métodos String



# JAVASCRIPT - INTRODUCCIÓN



## OPERADOR CONCATENACION

Podemos usar la concatenación (representada por el signo +) para construir cadenas formadas por varias cadenas más pequeñas o uniendo cadenas con otros tipos.



# JAVASCRIPT - INTRODUCCIÓN



Inspector Consola Depurador Red



Filtrar salida

```
» > let cad1 = "Aprendiendo JavaScript";
      let cad2 = " Con Campus";
      let cad3 = " En el ciclo ";
      let ciclo = 2;
      ...
```

Aprendiendo JavaScript Con Campus En el ciclo 2



JS

# JAVASCRIPT - INTRODUCCIÓN



## TEMPLATE STRING O LITERALS

Los literales de plantilla son similares a las cadenas normales. Pero a diferencia de las cadenas regulares, los literales de plantilla nos permiten incrustar expresiones en ellos.



# JAVASCRIPT - INTRODUCCIÓN



## TEMPLATE STRING O LITERALS

- Se pueden usar con la comilla atrás (` `).

- Para incluir una expresión, simplemente la escribimos como: \${expresión}.



# JAVASCRIPT - INTRODUCCIÓN



## TEMPLATE STRING O LITERALS

El término "**template string**" se utilizó originalmente para referirse a esta nueva característica de JavaScript, pero el término "**template literal**" se ha vuelto más popular en los últimos años. Ambos términos son correctos, pero el término "**template literal**" es el que se utiliza en la documentación oficial de ECMAScript.



# JAVASCRIPT - INTRODUCCIÓN



## TEMPLATE STRING

```
» console.log("Aprendiendo JavaScript con ${cad2} en el ciclo ${ciclo}");
    Aprendiendo JavaScript con ${cad2} en el ciclo ${ciclo}
← undefined
```

- Una cadena tradicional con doble comillas ("") no reemplaza el valor de la expresión dentro de ella.



# JAVASCRIPT - INTRODUCCIÓN



## TEMPLATE LITERALS

```
>> ▶ const multilinea = `  
    Esta es una  
    cadena con muchas  
    lineas.`;  
  
    console.log(multilinea);
```

Esta es una  
cadena con muchas  
lineas.

Permiten la creación de cadenas de texto multilínea. Para ello, se pueden utilizar caracteres de nueva línea dentro de las cadenas.



JS

# JAVASCRIPT - INTRODUCCIÓN



## MÉTODOS STRING

ES6 introduce algunos métodos nuevos para cadenas. Echemos un vistazo a ellos:

**repeat(num)**: Devuelve una cadena que contiene num copias de nuestra cadena inicial:

```
» console.log("*".repeat(10));
```

```
*****
```



# JAVASCRIPT - INTRODUCCIÓN



## MÉTODOS STRING

**startsWith(str, index)**: Devuelve verdadero si una cadena comienza con 'str'. El parámetro de índice especifica en qué parte de la cadena comenzar a buscar (el valor predeterminado es 0; el comienzo de la cadena):

```
» str = "Hola estoy aprendiendo JavaScript en Campus.";
  console.log(str.startsWith("Hola"));
  console.log(str.startsWith("Hola", 2));
  console.log(str.startsWith("Script", 27));

true
false
true
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## MÉTODOS STRING

**endsWith(str, length)**: Devuelve verdadero si una cadena termina con 'str'. El parámetro de longitud especifica la longitud de la cadena a buscar (el valor predeterminado es la cadena completa):



# JAVASCRIPT - INTRODUCCIÓN



## MÉTODOS STRING

### endsWith(str, length)

```
JS methods.js ✘
1 const str = 'Hello. My name is Inigo Montoya. You killed my father. Prepare to die. ';
2
3 console.log(str.endsWith('Prepare to die.')); // outputs true
4
5 console.log(str.endsWith('Prepare to die.')); // outputs false (note the space at the end);
6
7 console.log(str.endsWith('Montoya', 31)); // outputs true
8
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## MÉTODOS STRING

**includes(str, index)**: Devuelve verdadero si una cadena contiene 'str'. El parámetro de índice especifica en qué parte de la cadena comenzar a buscar (el valor predeterminado es 0; el comienzo de la cadena):



# JAVASCRIPT - INTRODUCCIÓN



## MÉTODOS STRING

### includes(str, index):

```
const str = 'Hello. My name is Inigo Montoya. You killed my father. Prepare to die.'  
  
console.log(str.includes('Inigo Montoya')); // outputs true  
  
console.log(str.includes('Westley')); // outputs false  
  
console.log(str.includes('Hello', 1)); // outputs false  
|
```



# JAVASCRIPT - INTRODUCCIÓN



```
'midudev'.length // 7
'midudev'[1] // i
'midudev'.includes('dev') // true
'midudev'.indexOf('midu') // 0
'midudev'.startsWith('midu') // true
'midudev'.endsWith('paint') // false
'midudev'.slice(0, 4) // 'midu'
'midudev'.slice(4) // 'dev'
'midudev'.toUpperCase() // 'MIDUDEV'
'MiduDev'.toLowerCase() // 'midudev'
'midudev'.replace('dev', '👨‍💻') // 'midu👨‍💻'
'midu'.repeat(3) // 'midumidumidu'
' mi du '.trim() // 'mi du'
'mi du dev'.split(' ') // [ 'mi', 'du', 'dev' ]
```



JS

# JAVASCRIPT - INTRODUCCIÓN



**Interacción: alert, prompt, confirm**



# JAVASCRIPT - INTRODUCCIÓN



## alert

Ya la hemos visto. Muestra un mensaje y espera a que el usuario presione "Aceptar".

Por ejemplo:

```
1 alert("Hello");
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## **prompt**

- La función `prompt` acepta dos argumentos:

```
1 result = prompt(title, [default]);
```

- Muestra una ventana modal con un mensaje de texto, un campo de entrada para el visitante y los botones OK/CANCELAR.

`title`

El texto a mostrar al usuario.

`default`

Un segundo parámetro opcional, es el valor inicial del campo de entrada.



JS

# JAVASCRIPT - INTRODUCCIÓN



## confirm

- La sintaxis:

```
1 result = confirm(pregunta);
```

- La función `confirm` muestra una ventana modal con una `pregunta` y dos botones: OK y CANCELAR.

- El resultado es `true` si se pulsa OK y `false` en caso contrario.

- Por ejemplo:

```
1 let isBoss = confirm("¿Eres el jefe?");  
2  
3 alert( isBoss ); // true si se pulsa OK
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## CONVERSIÓN DE TIPOS



# JAVASCRIPT - INTRODUCCIÓN



## ToString

La conversión a string ocurre cuando necesitamos la representación en forma de texto de un valor.

Por ejemplo, `alert(value)` lo hace para mostrar el valor como texto.

También podemos llamar a la función `String(value)` para convertir un valor a string:

```
1 let value = true;
2 alert(typeof value); // boolean
3
4 value = String(value); // ahora value es el string "true"
5 alert(typeof value); // string
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## ToNumber

La conversión numérica ocurre automáticamente en funciones matemáticas y expresiones.

Por ejemplo, cuando se dividen valores no numéricos usando `/`:

```
1 alert( "6" / "2" ); // 3, los strings son convertidos a números
```



JS

# JAVASCRIPT - INTRODUCCIÓN



Podemos usar la función `Number(value)` para convertir de forma explícita un valor a un número:

```
1 let str = "123";
2 alert(typeof str); // string
3
4 let num = Number(str); // se convierte en 123
5
6 alert(typeof num); // number
```

Si el string no es un número válido, el resultado de la conversión será `NaN`. Por ejemplo:

```
1 let age = Number("un texto arbitrario en vez de un número");
2
3 alert(age); // NaN, conversión fallida
```



# JAVASCRIPT - INTRODUCCIÓN



## Valor Se convierte en...

undefined NaN

null 0

true and false 1 y 0

string Se eliminan los espacios (incluye espacios, tabs \t , saltos de línea \n , etc.) al inicio y final del texto. Si el string resultante es vacío, el resultado es 0 , en caso contrario el número es "leído" del string. Un error devuelve NaN .



JS

# JAVASCRIPT - INTRODUCCIÓN



Ejemplos:

```
1 alert( Number(" 123  ") ); // 123
2 alert( Number("123z") );      // NaN (error al leer un número en "z")
3 alert( Number(true) );       // 1
4 alert( Number(false) );      // 0
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## ToBoolean

Las reglas de conversión:

- Los valores que son intuitivamente "vacíos", como `0`, `""`, `null`, `undefined`, y `NaN`, se convierten en `false`.
- Otros valores se convierten en `true`.

Por ejemplo:

```
1 alert( Boolean(1) ); // true
2 alert( Boolean(0) ); // false
3
4 alert( Boolean("hola") ); // true
5 alert( Boolean("") ); // false
```



JS

# JAVASCRIPT - INTRODUCCIÓN



**ToBoolean** – Ocurren en operaciones lógicas. Se puede realizar con `Boolean(value)`.

Sigue las reglas:

| Valor                                    | Se convierte en... |
|--|--------------------|
| <code>0, null, undefined, NaN, ''</code> | <code>false</code> |
| <code>cualquier otro valor</code>        | <code>true</code>  |



# JAVASCRIPT - INTRODUCCIÓN



**ToBoolean** – Ocurren en operaciones lógicas. Se puede realizar con `Boolean(value)`.

Sigue las reglas:

| Valor                                    | Se convierte en... |
|--|--------------------|
| <code>0, null, undefined, NaN, ''</code> | <code>false</code> |
| <code>cualquier otro valor</code>        | <code>true</code>  |



# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT

JavaScript puede "mostrar" datos de diferentes maneras:

- Escribir en un cuadro de alerta, usando `window.alert()`.
- Escribiendo en la consola del navegador, usando `console.log()`.
- Escribir en un elemento HTML, usando `innerHTML`.
- Escribir en la salida HTML usando `document.write()`.



# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT

```
<body>
  <h1>Probando document.write e innerHTML</h1>
  <p id="miparrafo"></p>
  <script>
    document.write("<h2>Probando write</h2>");
    document.getElementById("miparrafo").innerHTML = "<h2>Probando inner </h2>";
  </script>
  <script src="miscript.js"></script>
</body>
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT - INTERNO

Para acceder a un elemento HTML, JavaScript puede usar el método ***document.getElementById(id)***.

El **id** atributo define el elemento HTML.



# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT - INTERNO

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT - WRITE

- Para fines de prueba, es conveniente utilizar **document.write()**:



# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT - WRITE

```
<script>  
document.write(5 + 6);  
</script>
```



# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT - WRITE

El uso de `document.write()` después de cargar un documento HTML, eliminará **todo el HTML existente**:



# JAVASCRIPT - INTRODUCCIÓN



## SALIDA DE JAVASCRIPT - WRITE

```
<h2>Tituto</h2>
<p>Párrafo será eliminado por write</p>

<button type="button" onclick="document.write(5 +
6)">Inténtalo</button>
```

El método `document.write()` solo debe usarse para pruebas.



JS

# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Escribe un programa que solicite dos números y posteriormente muestre su suma, resta, multiplicación y división en una ventana y en consola.
- Ejemplo:

### Operadores aritméticos

Sean dos números: 17 y 8

La suma es .....: 25

La diferencia es .....: 9

Su producto .....: 136

Su cociente es .....: 2.125



JS

# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- En la oficina, 2 monitores están conectados a cada computadora. Solicitar el número de campers para calcular la cantidad de monitores a instalar.

## Tarea

Complete el código para calcular y enviar la cantidad de monitores a la consola y en el HTML inserte el número de camper y el número de computadores a instalar.



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Debido a la llegada de nuevos camper a Campus, se ve necesario crear usuarios Linux en cada computadora. Solicitar la cantidad de campers antiguos y de campers nuevos con el fin de calcular cuantos usuarios Linux máximos se deben crear en cada computadora.
- Mostrar el resultado en una ventana emergente.



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Hay muchas situaciones en las que desea verificar la edad de alguien.
- Se le proporciona un programa que toma la edad del usuario como entrada.
- Escribe el código para verificar si el usuario es un adulto y envíe a la consola el valor booleano correspondiente. **NOTA:** No puede usar estructuras ifs.

**Entrada de muestra**

20

**Salida de muestra**

verdadero



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Dado un reloj que mide 24 horas en un día, escriba un programa que tome la hora como entrada. Si la hora está en el rango de 0 a 12, envíe am a la consola y envíe pm si no lo está.

### Entrada de muestra

13

### Salida de muestra

pm



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- Necesitas planear un viaje por carretera. Estás viajando a una velocidad promedio de 40 millas por hora.  
Dada una distancia en millas como entrada (el código para tomar la entrada ya está presente), envíe a la consola el tiempo que le llevará recorrerla en minutos.

**Entrada de muestra:**

150

**Salida de muestra:**

225



# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- ¿Cuáles son los valores finales de todas las variables a, b, c y d después del código a continuación?

```
1 let a = 1, b = 1;  
2  
3 let c = ++a; // ?  
4 let d = b++; // ?
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- ¿Cuáles son los valores de 'a' y 'x' después del código a continuación?

```
1 let a = 2;  
2  
3 let x = 1 + (a *= 2);
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- ¿Cuáles son los resultados de estas expresiones?

```
1  "" + 1 + 0
2  "" - 1 + 0
3  true + false
4  6 / "3"
5  "2" * "3"
6  4 + 5 + "px"
7  "$" + 4 + 5
8  "4" - 2
9  "4px" - 2
10 " -9 " + 5
11 " -9 " - 5
12 null + 1
13 undefined + 1
14 " \t \n" - 2
```



JS

# JAVASCRIPT - INTRODUCCIÓN



## Ejercicio

- ¿Cuál será el resultado de las siguientes expresiones?

```
1 5 > 4
2 "apple" > "pineapple"
3 "2" > "12"
4 undefined == null
5 undefined === null
6 null == "\n0\n"
7 null === +"\\n0\\n"
```



JS



# Programa académico **CAMPUS**

## Ciclo 2

