

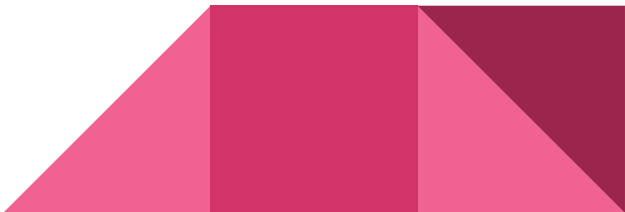
# UD2. Manejo de la sintaxis de JS

Desarrollo Web en Entorno Cliente

2º DAW - Curso 2021/2022

IES Doñana - Sanlúcar de Barrameda (Cádiz)

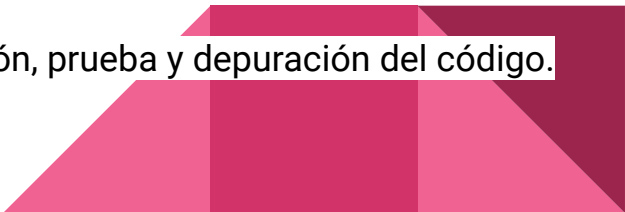
# 0. Objetivos de la unidad

1. Utilizar los distintos **tipos de variables y operadores** disponibles en el lenguaje
  2. Identificar los **ámbitos** de utilización de las variables.
  3. Realizar **conversiones** entre distintos tipos de datos.
  4. Añadir de forma correcta los **comentarios** al código
  5. Añadir **mecanismos de decisión** para la creación de bloques de sentencias.
  6. Añadir **bucles** y verificar su funcionamiento
  7. Utilizar **herramientas** de programación y depuración.
- 

# 0. Objetivos de la unidad

## Resultados de aprendizaje y criterios de evaluación

**RA2. Escribe sentencias simples, aplicando la sintaxis del lenguaje y verificando su ejecución sobre navegadores Web.**

- a) Se ha seleccionado un lenguaje de programación de clientes Web en función de sus posibilidades.
  - b) Se han utilizado los distintos **tipos de variables y operadores** disponibles en el lenguaje.
  - c) Se han identificado los **ámbitos** de utilización de las variables.
  - d) Se han reconocido y comprobado las peculiaridades del lenguaje respecto a las **conversiones** entre distintos tipos de datos.
  - e) Se han añadido **comentarios** al código.
  - f) Se han utilizado **mecanismos de decisión** en la creación de bloques de sentencias.
  - g) Se han utilizado **bucles** y se ha verificado su funcionamiento.
  - h) Se han utilizado **herramientas** y entornos para facilitar la programación, prueba y depuración del código.
- 

# 0. Objetivos de la unidad

Apuntes hechos a mano de Majo Ledesma:

<https://twitter.com/MajoLedes/status/1232696860948729858>

Particularidades JS:

[https://www.youtube.com/watch?v=3se2-thqf-A&ab\\_channel=yuoyubemoneycoin](https://www.youtube.com/watch?v=3se2-thqf-A&ab_channel=yuoyubemoneycoin)



# 0. Objetivos de la unidad

## Particularidades JS:

> typeof NaN	> true==1
< "number"	< true
> 9999999999999999	> true===1
< 10000000000000000	< false
> 0.5+0.1==0.6	> (!+[[]+[]+![]).length
< true	< 9
> 0.1+0.2==0.3	> 9+"1"
< false	< "91"
> Math.max()	> 91-"1"
< -Infinity	< 90
> Math.min()	> []==0
< Infinity	< true
> []+[]	
< ""	
> []+{}	
< "[object Object]"	
> {}+[]	
< 0	
> true+true+true===3	
< true	
> true-true	
< 0	



# ÍNDICE


1. Historia de JS
2. Características de JS
3. El lenguaje JS: sintaxis
4. Entrada y salida de datos
5. Variables
6. Tipos de datos
7. Operadores
8. Estructuras de control
9. Control de errores
10. Buenas prácticas



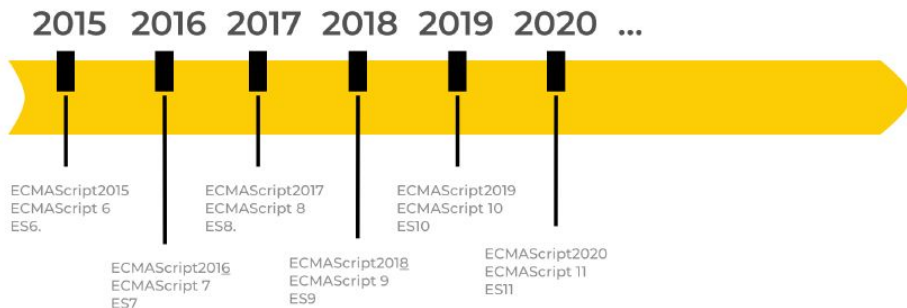
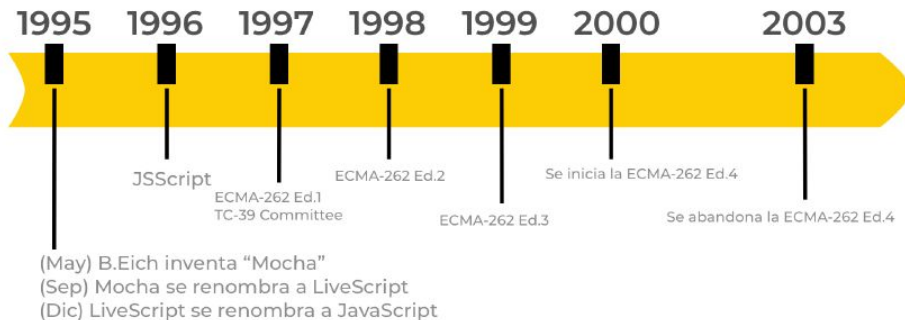
# 1. Historia de JS

- 1995: Desarrollado por **Brendan Eich** de Netscape con el nombre de **Mocha**. Para solucionar el problema de navegar por internet con una velocidad tan lenta. Propuso crear un lenguaje que se ejecutara en el cliente (navegador web). Renombrado posteriormente a **LiveScript**. Finalmente quedó como **JavaScript**: El cambio de nombre coincidió con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape Navigator en la versión 2.002 en diciembre de 1995. Produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java.
- 1996: Microsoft lanzó su propia versión JavaScript «**JScript**» Los dialectos pueden parecer tan similares que los términos «JavaScript» y «JScript» a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos.
- 1997: En respuesta a estas incompatibilidades, la ECMA (European Computer Manufacturers Association) emprendió un esfuerzo de estandarización que desembocó en la publicación del estándar **ECMAScript**.

# 1. Historia de JS

- 2015: ECMAScript 6 (ES6) quedó cerrado en junio 2015. Aportaciones:
    - Mejoras de sintaxis: parámetros por defecto, let, plantillas...
    - Módulos para organización de código
    - Verdaderas clases para programación orientada a objetos
    - Promesas para programación asíncrona
    - Mejoras en programación funcional: expresiones de flecha, iteradores...
  - 2016: ECMAScript 7, cuyo nombre oficial es ECMAScript 2016. Añade:
    - Operador de exponenciación
    - Método nuevo para las matrices que permite comprobar si existen ciertos elementos dentro de éstas.
  - 2017: ES8 Constructores async/await para generadores y promesas
  - 2018: ECMAScript 2018, incluye operadores rest/spread para variables (tres puntos: ...identificador), iteración asíncrona, Promise.prototype.finally()
  - 2019: La 10.<sup>a</sup> edición, incorporó Array.flat(), Array.flatMap(), String.trimStart(), String.trimEnd(), errores opcionales en el bloque catch, Object.fromEntries()
- 





Fuente:  
Aprendiendo JavaScript,  
de Carlos Azaustre

## 2. Características de JS

- Lenguaje de programación **interpretado**. Cualquier navegador web actual incorpora un intérprete para código JS.
- Está basado en el concepto de objeto, pero no es un lenguaje orientado a objetos puro. Sus objetos utilizan herencia **basada en prototipos** → los objetos no son creados mediante la instanciación de clases sino mediante la clonación de otros objetos o mediante la escritura de código. Los objetos ya existentes pueden servir de prototipos para los que el programador necesite crear posibilitando añadir propiedades y métodos a cualquier objeto de forma dinámica.
- No se declaran los tipos de datos de las variables (Es un lenguaje **débilmente tipado y de tipado dinámico**).
- Fue diseñado de forma que se ejecutará en un **entorno limitado** que permitiera a los usuarios confiar en la ejecución de los scripts. Los scripts de JS no pueden comunicarse con recursos que no pertenezcan al mismo dominio desde el que se descargó el script.

### 3. El lenguaje JS: sintaxis

- La sintaxis de JavaScript es muy similar a la de Java o C++.
- No es necesario terminar cada sentencia con el carácter de **punto y coma (;)**:
  - En la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter “;”. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (;)
- **Mayúsculas y minúsculas:**
  - Es case sensitive a diferencia de por ejemplo HTML
  - No es lo mismo utilizar `alert()` que `Alert()`




### 3. El lenguaje JS: sintaxis

- **Tabulación y saltos de línea:**

- JavaScript ignora los espacios, las tabulaciones y los saltos de línea con algunas excepciones.
- Emplear la tabulación y los saltos de línea mejora la presentación y la legibilidad del código.

- **Palabras reservadas:**

- Algunas palabras no se pueden utilizar para definir nombres de variables, funciones o etiquetas.
  - Es aconsejable no utilizar tampoco las palabras reservadas para futuras versiones de JavaScript.
  - En ecma-internacionales posible consultar todas las palabras reservadas de JavaScript.
- 

### 3. El lenguaje JS: sintaxis

- **Comentarios en el código:**

Los comentarios no se interpretan por el navegador. Existen dos formas de insertar comentarios:

- Doble barra (//) – Se comenta una sola línea de código.
- Barra y asterisco (/ \* al inicio y \*/ al final) – Se comentan varias líneas de código.

```
<script>  
    // Este modo permite comentar una sola línea  
    /* Este modo permite realizar comentarios de  
    varias líneas */  
</script>
```

Se usan para:

- Para que el código sea más claro e informar del uso de variables y funciones
- Para desactivar un trozo de código que no queremos que se use
- Para dejar nuestra información de contacto


### 3. El lenguaje JS: sintaxis

- **Identificadores:**

El nombre que se da a las variables y otros elementos del lenguaje se conocen como identificador y debe cumplir dos condiciones:

- El primer carácter debe ser una letra aunque también puede ser un guión bajo (\_) o un dólar (\$).
- El resto de caracteres pueden ser letras, números, guiones bajos.

Además, JS es sensible a mayúsculas y no se pueden usar palabras reservadas.




## 4. Entrada y salida de datos

### Consola (Salida de datos)

Se pueden utilizar la consola como salida de datos que no se consideran oportunos que aparezca en el HTML para el usuario pero sí que pueden ser útiles como programadores.

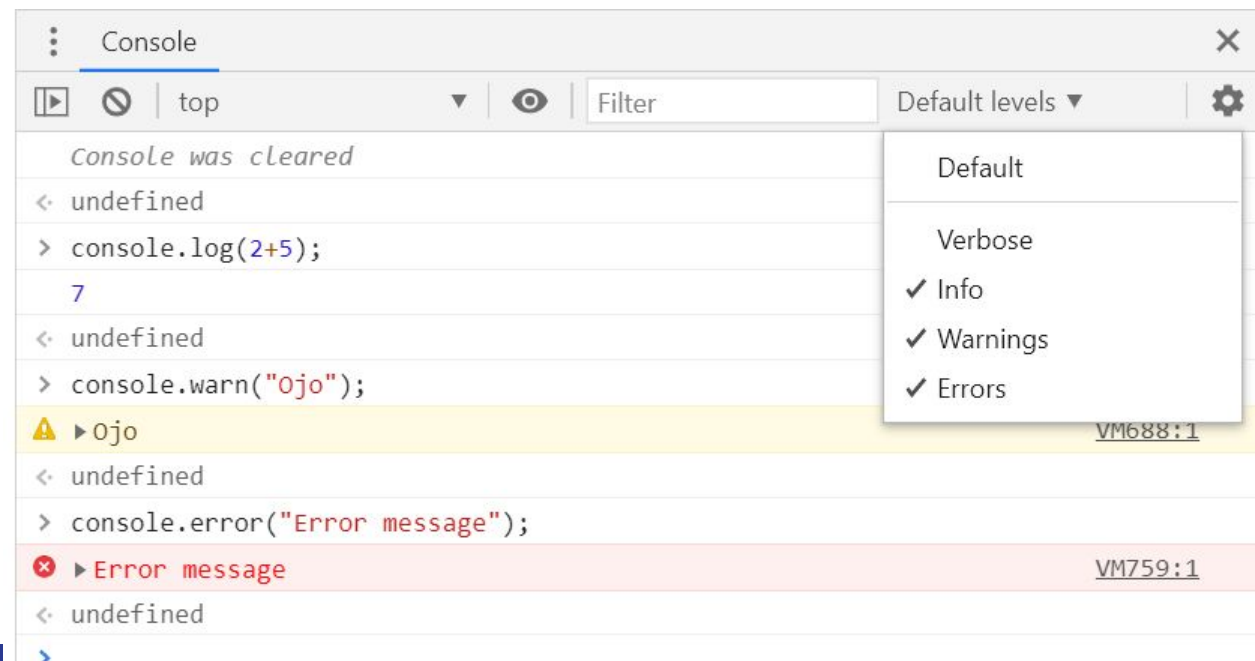
Además la consola permite sentencias en JS

- `console.log("texto" o expresión);`
  - `console.info("texto");`
  - `console.warning("mensaje warning");`
  - `console.error("mensaje de error");`
- 

## 4. Entrada y salida de datos

### Consola (Salida de datos)

Se pueden filtrar el tipo de mensajes desde la consola:





## 4. Entrada y salida de datos

### Ventana alert() (Salida de datos)

Se trata de una ventana estándar que usamos para mostrar información en pantalla. Se puede mostrar texto, variables y texto en conjunto con variables. Si se quiere mostrar texto se pone entre comillas.

Sintaxis:

```
alert("texto de la ventana");  
alert(variable);  
alert("texto"+variable);
```



## 4. Entrada y salida de datos

### **Método document.write() (Salida de datos)**

Opción para mostrar una salida directamente dentro del documento HTML. Se recomienda su uso solo para pruebas. Introduce texto, aunque admite expresiones y código HTML:

Sintaxis:

```
document.write("<p>Esto es un párrafo</p>");
```



## 4. Entrada y salida de datos

### **Atributo innerHTML (Salida de datos)**

Opción más recomendable. También admite código HTML:

Sintaxis:

```
document.getElementById("div2").innerHTML=  
("<p>Esto es tu nombre: "+nombre+"</p>");
```



## 4. Entrada y salida de datos

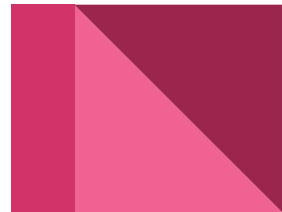
```
<body>
  <div>
    <h1 id="cabecera">Hola a todos</h1>
    <div id="div1"></div>
    <div id="div2"></div>
  </div>
  <script>
    document.write("<p>Esto es un párrafo introducido por write</p>");
    document.getElementById("div1").innerHTML=("Sección 1");
    var nombre="Damián León";
    alert("Hola " + nombre);
    document.getElementById("div2").innerHTML=("<p>Este es tu nombre: "+nombre+"</p>");
  </script>
</body>
```

**Hola a todos**

Sección 1

Este es tu nombre: Damián León

Esto es un párrafo introducido por write



## 4. Entrada y salida de datos

### Ventana Confirm (Entrada y Salida de datos)

Muestra una ventana de confirmación.

- Si el usuario acepta → devuelve true
- En caso contrario → devuelve false

```
var c=confirm("Pulsa un botón");  
var texto;  
if (c){  
    texto="aceptar"  
} else {  
    texto="cancelar"  
}  
document.getElementById("div3").innerHTML=("Has pulsado: "+texto);
```



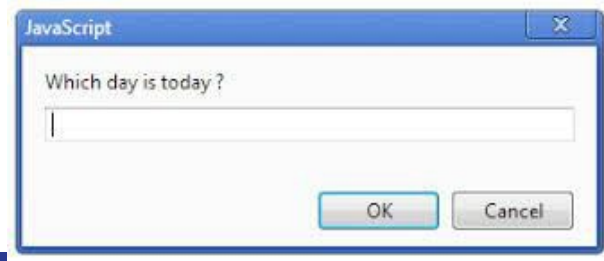
## 4. Entrada y salida de datos

### Ventana Prompt (Entrada y Salida de datos: texto)

- Esta ventana permite interactuar al usuario por medio de la introducción de datos . Siempre devuelve un string.

`let valor = prompt("Texto de la ventana","valor inicial caja");`

- Al pulsar el botón aceptar, el contenido de la caja pasa a valor.
- Si se pulsa el botón cancelar, el contenido de la caja se pierde y valor queda con valor null.



## 5. Variables

- Se pueden definir como zonas de la memoria de un ordenador que se identifican con un nombre y en las cuales se almacenan ciertos datos.
- El desarrollo de un script conlleva:
  - **Declaración de variables:** aunque no es obligatorio, sí es recomendable.
  - **Inicialización de variables.**



## 5. Variables

- **Declaración de variables:**

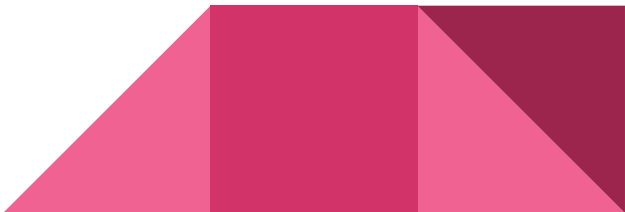
Se declaran mediante la palabra clave *var* seguida por el nombre que se quiera dar a la variable.

```
var mi_variable;
```

Es posible declarar más de una variable en una sola línea.

```
var mi_variable1, mi_variable2;
```

Javascript es un lenguaje de tipado dinámico. Esto significa que no se indica de qué tipo es una variable al declararla e incluso puede cambiar su tipo a lo largo de la ejecución del programa.





## 5. Variables

```
var miVariable;    // declaro miVariable y como no se asignó un valor valdrá undefined
```

```
miVariable='Hola';  // ahora su valor es 'Hola', por tanto contiene una cadena de texto
```

```
miVariable=34;      // pero ahora contiene un número
```

```
miVariable=[3, 45, 2]; // y ahora un array
```

```
miVariable=undefined; // para volver a valer el valor especial undefined
```



## 5. Variables

### Inicialización de variables:

- Se puede asignar un valor a una variable de tres formas:

- Asignación directa de un valor concreto.

```
var mi_variable_1 = 30;
```

```
var x=1, y=2, z=3;
```

- Asignación indirecta a través de un cálculo en el que se implican a otras variables o constantes.

```
var mi_variable_2 = mi_variable_1 + 10;
```

- Asignación a través de la solicitud del valor al usuario del programa.

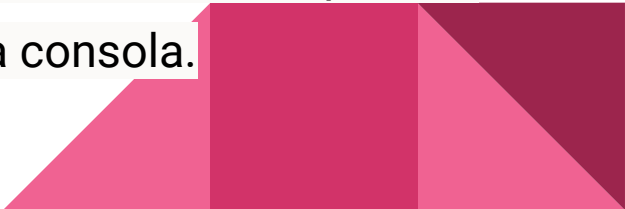
```
var mi_variable_3 = prompt('Introduce un valor:');
```



## 5. Variables

### Ámbito (scope)

Tiempo de vida de las variables en JavaScript.

- Comienza desde que es declarada.
  - Las **variables locales** son eliminadas cuando la función es completada. Solo se pueden recuperar dentro de su ámbito.
  - Las **variables globales** son eliminadas cuando la página se cierra. Se pueden recuperar desde cualquier punto, por ejemplo desde la consola.
- 

## 5. Variables

### Ámbito (scope)

- Cuando utilizamos `var` estamos haciendo que la variable que estamos declarando sea local al ámbito donde se declara.

```
function creaMensaje() {  
  var mensaje = "Mensaje de prueba";  
  alert(mensaje);  
}  
creaMensaje();
```

- Si no utilizamos la palabra `var` para declarar una variable, ésta será global a toda la página, sea cual sea el ámbito en el que haya sido declarada.

```
var mensaje = "Mensaje de prueba";  
  
function muestraMensaje() {  
  alert(mensaje);  
}
```

=

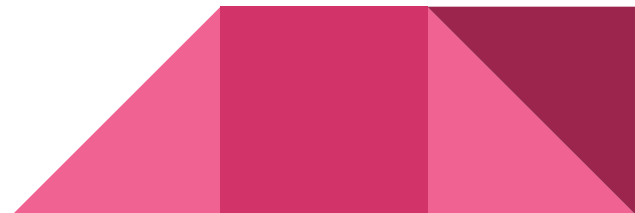
```
mensaje = "Mensaje de prueba";  
  
function muestraMensaje() {  
  alert(mensaje);  
}
```

## 5. Variables

### Ámbito (scope)

¿Qué se visualizará por pantalla?

```
function creaMensaje() {  
    var mensaje = "Mensaje de prueba";  
}  
creaMensaje();  
alert(mensaje);
```



## 5. Variables

### Ámbito (scope)

```
var mensaje = "gana la de fuera";  
function muestraMensaje() {  
  var mensaje = "gana la de dentro";  
  alert(mensaje);  
}  
alert(mensaje);  
muestraMensaje();  
alert(mensaje);
```

#### Resultado

gana la de fuera  
gana la de dentro  
gana la de fuera

```
var mensaje = "gana la de fuera";  
function muestraMensaje() {  
  mensaje = "gana la de dentro";  
  alert(mensaje);  
}  
alert(mensaje);  
muestraMensaje();  
alert(mensaje);
```

#### Resultado

gana la de fuera  
gana la de dentro  
gana la de dentro

## 5. Variables

### Hoisting (Alzamiento)

Mueve declaraciones al inicio de su ámbito

- En Javascript una variable puede ser declarada después de haber sido usada

```
x = 5; // Assign 5 to x
```

```
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                      // Display x in the element
```

//x es 5

```
var x; // Declare x
```

- Este lenguaje permite subir las declaraciones de las variables al principio del ámbito (hoisting) pero no las inicializaciones.

```
var x = 5; // Initialize x
```

```
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y;           // Display x and y
```

```
var y = 7; // Initialize y
```

//x es 5 e y es undefined

## 5. Variables

### Variables `let` y `const` (ES6)

- `let`

Permite declarar variables locales limitando su alcance al **bloque**, declaración o expresión donde se está usando . Dejan de ser globales a la función (`var`).





## 5. Variables

### Variables let y const (ES6)

- let

```
//ES5
(function() {
  console.log(x); // x no está definida aún.
  if(true) {
    var x = "hola mundo";
  }
  console.log(x);
  // Imprime "hola mundo", porque "var" hace que sea global
  // a la función;
})();

//ES6
(function() {
  if(true) {
    let x = "hola mundo";
  }
  console.log(x);
  //Da error, porque "x" ha sido definida dentro del "if"
})();
```

## 5. Variables

### Variables let y const (ES6)

- **const**

Se usa para variables que no van a modificar su valor en tiempo de ejecución. Podemos crear constantes que sólo se puedan leer y no modificar a lo largo del código

```
const PI;  
PI = 3.15;  
// ERROR, porque ha de asignarse un valor en la  
// declaración
```

```
const PI = 3.15;  
PI = 3.14159;  
// ERROR de nuevo, porque es sólo-lectura
```

# 5. Variables

## Variables let y const (ES6)

### Resumen

- **var:** declara una variable de scope global o local para la función sin importar el ámbito de bloque. Permite hoisting.
- **let:** declara una variable de scope global, local para la función o de bloque. Es reassignable y no permite hoisting.
- **const:** declara una variable de scope global, local para la función o de bloque. No es reassignable, pero es mutable. No permite hoisting.



## 6. Tipos de datos

- Javascript es un lenguaje no tipado.
- **No tipado == No se declara el tipo de datos.**
- No obstante:
  - Una vez declaradas, las variables tienen un tipo.
  - Una variable puede tener distintos tipos a lo largo de la ejecución (**Tipado dinámico**)
  - En algunas operaciones entre distintos tipos se obtienen un resultado en uno de ellos. (**Débilmente tipado**)

```
x = 1 ;  
x = '1' ;
```

```
"1234" * 1 === 1234  
2 / "bla bla" -> NaN
```

## 6. Tipos de datos

Los tipos de datos especifican qué tipo de valor se guardará en una determinada variable.

- **Primitivos** (se trabaja directamente con el valor del tipo)
  - String
  - Number
  - Boolean
  - Undefined
  - Null
- **Compuestos** (se trabaja con la referencia al valor del tipo)
  - Function
  - Object



## 6. Tipos de datos

### Operador typeof:

- Devuelve el tipo de dato que almacena una variable.
- Los posibles valores de retorno del operador son:
  - undefined, boolean, number, string para cada uno de los tipos primitivos
  - object para los datos complejos: objetos, tipos de referencia (new String...) y también para los valores de tipo null.
- Ejemplo:

```
typeof "John"           // Returns string
typeof 3.14              // Returns number
typeof false            // Returns boolean
typeof [1,2,3,4]         // Returns object
typeof {name:'John', age:34} // Returns object
```

## 6. Tipos de datos

- **undefined:**

Variables que han sido definidas y todavía no se les ha asignado un valor (Ausencia de valor).

```
var variable1;  
typeof variable1; // devuelve "undefined"
```

Nota: El operador `typeof` no distingue entre las variables declaradas, pero no inicializadas y las variables que ni siquiera han sido declaradas



## 6. Tipos de datos

- **null:**

Valor de nada, es decir, está definida pero no tiene valor, es nulo.

Similar a undefined, pero:

```
undefined == null          //true  
pero no undefined===null  //false
```

El tipo null se suele utilizar para representar objetos que en ese momento no existen, Ejemplo:

```
let nombreUsuario = null;
```

Recuerda `typeof(null)=object`





## 6. Tipos de datos

- **number:**

En JavaScript existe sólo un tipo de dato numérico.

Todos los números se representan a través del formato de punto flotante de 64 bits. Este formato es el llamado double en los lenguajes Java o C++.

- Ejemplos:

- Número entero: 45
- Número decimal: 3.1415
- Notación científica: 6.022e+23
- Binario: 0b10001
- Sistema octal: 034 ó 0o34
- Sistema hexadecimal: 0xA3

- Operaciones: Aritméticas (+, -, \*, /, %, \*\*), de comparación(==, >=...)

## 6. Tipos de datos

- **number:**

JavaScript define tres valores especiales :

- **Infinity:**


```
let x = 1/0;  
let y = Infinity;
```

- **-Infinity:**

```
let x = -1/0;
```

- **NaN (Not a Number):** Suele darse cuando una expresión intenta operar de forma numérica un valor que no es un número o con indeterminaciones

```
let x = "Hola"*3;  
let y = 0/0;
```



## 6. Tipos de datos

- **number:**

Para manejar los valores NaN, se utiliza la función **isNaN( )**, que devuelve true si el parámetro que se le pasa NO ES un número:

```
isNaN(NaN); isNaN(100/"Apple"); //Devuelve true
```

```
isNaN(10); isNaN("10"); //Devuelve false
```



## 6. Tipos de datos

- **number:**
  - Objeto Number: Existe el objeto Number. Aunque no se aconseja usarlos así porque ralentizan los programas y pueden producir efectos secundarios no deseados

```
var x = 123;  
var y = new Number(123);
```

```
// typeof x returns number  
// typeof y returns object
```

```
var x = 500;  
var y = new Number(500);
```

```
// (x == y) is true because x and y have equal values
```

```
var x = 500;  
var y = new Number(500);
```

```
// (x === y) is false because x and y have different types
```

```
var x = new Number(500);  
var y = new Number(500);
```

```
// (x == y) is false because objects cannot be compared
```

## 6. Tipos de datos

- **number:**

Method	Description
toString()	<pre>var myNumber = 128; myNumber.toString(16); // returns 80 myNumber.toString(2);</pre>
toExponential()	<pre>var x = 9.656; x.toExponential(2); // returns 9.66e+0</pre>
toFixed()	<pre>var x = 9.656; x.toFixed(0); // returns 10 x.toFixed(2); // returns 9.66</pre>
toPrecision()	<pre>var x = 9.656; x.toPrecision(); // returns 9.656 x.toPrecision(2); // returns 9.7</pre>
valueOf()	<pre>var x = 123; x.valueOf(); // returns 123 from variable x (123).valueOf(); // returns 123 from literal 123</pre>

Devuelve String

## 6. Tipos de datos

- **number:**

### **Ejercicio Laura Folgado:** [u2e1\\_numeros](#)

Crea un programa en el que crees 5 variables numéricas (entero, decimal, científico, octal y hexadecimal).

A las variables les asignarás los siguientes números: 1357, 135.7, 135e7, 01357 y 0x1357.

Muestra con 5 alerts su valor, escribiendo la siguiente sentencia:

```
alert ("Número entero" + entero);
```

Comenta el código con los comentarios que estimes necesarios.

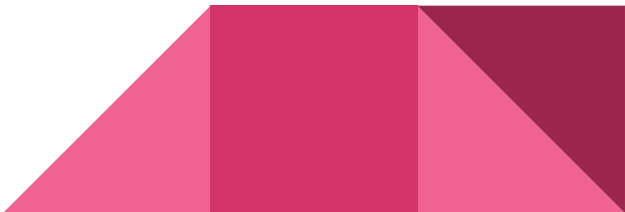


## 6. Tipos de datos

- **string:**
  - Se pueden representar letras, dígitos, signos de puntuación o cualquier otro carácter de Unicode.
  - La cadena de caracteres se debe definir entre comillas dobles o comillas simples.

```
var answer = "He is called 'Johnny'";
```

```
var answer = 'He is called "Johnny"';
```



## 6. Tipos de datos

- **string:**

- Cada carácter de la cadena se encuentra en una posición a la que se puede acceder individualmente. Siendo el primer carácter el de la posición 0.
- Para poder incluir cualquier carácter en las cadenas de caractere es necesario el uso de caracteres de escape.

Secuencia de escape	Resultado
\\	Barra invertida
\n	Salto de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\f	Salto de página
\r	Retorno de carro
\b	Retroceso
\'	Comilla simple
\"	Comilla doble





## 6. Tipos de datos

- **string:**

- template literals (ES6)

Son literales de texto que habilitan el uso de expresiones incrustadas.

Se usan para:

- Usar cadenas de texto de más de una línea
- Interpolación de cadenas de texto con ellas, es decir, otra forma de concatenar texto y variable sin usar el signo +.

```
//ES5
var nombre1 = 'Javascript';
var nombre2 = 'awesome';
console.log(nombre1 + " is " + nombre2);

//ES6
var nombre1 = 'Javascript';
var nombre2 = 'awesome';
console.log(`${nombre1} is ${nombre2}`);
```

## 6. Tipos de datos

- **string:**

- Objeto String: Al igual que con number se pueden crear objetos string aunque es una práctica desaconsejada.

```
var x = "John";  
var y = new String("John");  
  
// typeof x will return string  
// typeof y will return object
```

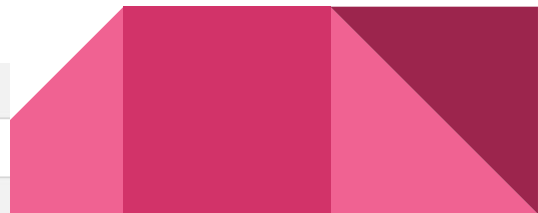


## 6. Tipos de datos

- **string:**

- Métodos

<code>charAt()</code>	Returns the character at the specified index (position)
<code>charCodeAt()</code>	Returns the Unicode of the character at the specified index
<code>concat()</code>	Joins two or more strings, and returns a copy of the joined strings
<code>fromCharCode()</code>	Converts Unicode values to characters
<code>indexOf()</code>	Returns the position of the first found occurrence of a specified value in a string
<code>lastIndexOf()</code>	Returns the position of the last found occurrence of a specified value in a string
<code>localeCompare()</code>	Compares two strings in the current locale
<code>match()</code>	Searches a string for a match against a regular expression, and returns the matches
<code>toUpperCase()</code>	Converts a string to uppercase letters
<code>trim()</code>	Removes whitespace from both ends of a string
<code>valueOf()</code>	Returns the primitive value of a String object



## 6. Tipos de datos

- **string:**

- Métodos

<code>replace()</code>	Searches a string for a value and returns a new string with the value replaced
<code>search()</code>	Searches a string for a value and returns the position of the match
<code>slice()</code>	Extracts a part of a string and returns a new string
<code>split()</code>	Splits a string into an array of substrings
<code>substr()</code>	Extracts a part of a string from a start position through a number of characters
<code>substring()</code>	Extracts a part of a string between two specified positions
<code>toLocaleLowerCase()</code>	Converts a string to lowercase letters, according to the host's locale
<code>toLocaleUpperCase()</code>	Converts a string to uppercase letters, according to the host's locale
<code>toLowerCase()</code>	Converts a string to lowercase letters
<code>toString()</code>	Returns the value of a String object

## 6. Tipos de datos

- **string:**

### EJERCICIO: u2e2\_cadenas

Crea un programa en el que crees 4 variables, 2 cadenas y 2 números, con los siguientes valores: tu nombre, tu apellido, tu edad y tu año de nacimiento.

- Muestra en un alert una frase que incluya comillas simples.
- Muestra en un alert tu nombre y apellidos separados por un salto de línea.
- Muestra en un alert la suma de las variables edad y año de nacimiento.
- Muestra en un alert la suma de todas las variables.

Comenta el código con los comentarios que estimes necesarios.



## 6. Tipos de datos

- **boolean:**

También conocido como valor lógico. Sólo admite dos valores: true o false.

Se recomienda declararlos con texto positivo:

`userIsLogged`

Otros valores pueden tomar el valor de true o false:

- falsy value:

- 0,
- null
- undefined
- Cadena vacía: ""
- NaN

- truthy value:

- 1
- Básicamente cualquier otro valor que no esté en la lista de false.

## 6. Tipos de datos

- **Conversión de tipos**

Al ser JS un lenguaje de tipo dinámico y débilmente tipado permite la conversión de tipos de variables (casting), incluso en ocasiones es el propio intérprete el que realiza esta conversión de manera automática (coertion):

- Con métodos globales:

- String()
- Boolean()
- Number()
- parseInt()
- parseFloat()

```
x = true;
Number(x);    // returns 1
x = false;
Number(x);    // returns 0
x = new Date();
Number(x);    // returns 1404568027739
x = "10"
Number(x);    // returns 10
x = "10 20"
Number(x);    // returns NaN
```

- Otros métodos:

- .toString()

## 6. Tipos de datos

- **Conversión de tipos**

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

Coertion:

- <https://twitter.com/Manz/status/1375436477702750208>





## 6. Tipos de datos

- Actividad de evaluación: Conversión de tipos



## 7. Operadores

JavaScript utiliza principalmente cinco tipo de operadores:

- Aritméticos.
- Lógicos.
- De asignación.
- De comparación.
- Condicionales.



## 7. Operadores

- Operadores aritméticos:
  - Permiten realizar cálculos elementales entre variables numéricas.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation ( <a href="#">ES2016</a> )
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

## 7. Operadores

- Operadores aritméticos:
  - Permiten realizar cálculos elementales entre variables numéricas.

```
JS  var suma = 5 + 9;           // R.  14
    var resta = 10 - 3;        // R.   7
    var multiplicacion = 4 * 3; // R.  12
    var division = 18 / 2;     // R.   9
    var modulo = 53 % 10;      // R.   3
    var agrupado1 = 3 + (4 * 5) - 6; // R.  17
    var agrupado2 = (3 + 4) * (5 - 6); // R.  -7
    var cadenas = '4' + '3';   // R.  43  ojo! son string
```

## 7. Operadores

- Operadores Lógicos:
  - Combinan diferentes expresiones lógicas con el fin de evaluar si el resultado de dicha combinación es verdadero o falso.

Operador	Nombre
& &	Y
	O
!	No

## 7. Operadores

- Operadores Lógicos:
  - Combinan diferentes expresiones lógicas con el fin de evaluar si el resultado de dicha combinación es verdadero o falso.

```
JS // AND(&&)
console.log(true && true); // true
console.log(true && false); // false
console.log(false && false); // false
console.log(false && true); // false
// OR(||)
console.log(true || true); // true
console.log(true || false); // true
console.log(false || false); // false
console.log(false || true); // true

var exp1 = true && true; // R. true
var exp2 = (3 == 3) && (5 >= 8); // R. false
var exp3 = (9 > 11) || (17 <= 40); // R. true
var exp4 = false || false; // R. false
```

## 7. Operadores

- Operadores de Asignación:
  - Permiten obtener métodos abreviados para evitar escribir dos veces la variable que se encuentra a la izquierda del operador.

Operador	Nombre
<code>+=</code>	Suma y asigna
<code>-=</code>	Resta y asigna
<code>*=</code>	Multiplica y asigna
<code>/=</code>	Divide y asigna
<code>%=</code>	Módulo y asigna

## 7. Operadores

- Operadores de comparación o relacionales:
  - Permiten comparar todo tipo de variables y devuelve un valor booleano.

Operador	Nombre
<	Menor que
<=	Menor o igual que
==	Igual
>	Mayor que
>=	Mayor o igual que
!=	Diferente
===	Estrictamente igual
!==	Estrictamente diferente



## 7. Operadores

- Operadores de comparación o relacionales:
  - Permiten comparar todo tipo de variables y devuelve un valor booleano.

JS

```
var mayorQue = 8 > 3;  
var menorQue = 10 < 50;  
var mayorIgual = 100 >= 90;  
var menorIgual = 10 <= 40;  
var igual = (13 == 13);  
var identico1 = (13 === 13); // Ojo! compara valor y tipo de dato  
var identico2 = (13 === '13'); // R. falso  
var distinto = 100 != 10;
```

## 7. Operadores

- Operadores condicionales o ternario:
  - Permite indicar al navegador que ejecute una acción en concreto después de evaluar una expresión.

`test ? expresion1 : expresion2`

- Si la expresión antes del operador es verdadera, se utiliza el primer valor a la derecha. En caso contrario se utiliza el segundo valor a la derecha.

```
let dividendo = prompt("Introduce el dividendo: ");  
let divisor = prompt("Introduce el divisor: ");  
(divisor != 0) ? alert(dividendo/divisor) : alert("No es posible la división por cero");
```

# 7. Operadores

- Orden de prioridad:
  1. Paréntesis
  2. Negación (!) / Incremento (++) / Decremento (--)
  3. Aritméticos:
    - Multiplicación (\*) / División (/) / Resto (%)
    - Suma (+) / Resta (-)
  4. Comparación
    - Mayor - menor (>, <, >=, <=)
    - Igualdad (==) / Desigualdad (!=)
  5. Lógicos
    - And lógico (&&)
    - Or lógico (||)
  6. Asignación (=, +=, -=...)



# 7. Operadores

EJERCICIO: [u2e3\\_comparacion](#)

Crea un programa en el que muestres el resultado de varias operaciones mediante alert, mostrando el texto exacto de la operación realizada y su resultado.

Ej:

- `var operacion1 = (10 == 10);`
- `alert ("La operación 10==10 es"+operacion1`

Las operaciones a realizar son:

- `10 == 10`
- `10 === 10`
- `10 === 10.0`
- `"Laura" == "laura"`
- `"Laura" > "laura"`
- `"Laura" < "laura"`
- `"123" == 123`
- `"123" === 123`
- `parseInt("123") === 123`

Comenta el código con los comentarios que estimes necesarios.



## 8. Estructuras de control

- Permiten modificar el flujo de ejecución de las instrucciones de la aplicación.
- Vamos a dividir estas estructuras en dos tipos:
  - De selección
    - if
    - switch
  - Iterativas
    - while
    - for



## 8. Estructuras de control

- **Estructuras de selección**

- Con las sentencias condicionales se puede gestionar la toma de decisiones y el posterior resultado por parte del navegador.
- Dichas sentencias evalúan condiciones y ejecutan ciertas instrucciones en base al resultado de la condición.
- Las estructuras de selección en JavaScript son:
  - **if:** Permite redirigir un curso de acción según la evaluación de una condición
  - **switch:** De acuerdo con el valor de una variable, permite ejecutar un grupo u otro de sentencias



## 8. Estructuras de control

- **Estructuras de selección**

- if - sintaxis (1)

```
if (condición){  
    instrucciones  
}
```



## 8. Estructuras de control

- **Estructuras de selección**

- if - sintaxis (2)

```
if(expresión){  
    instrucciones_si_true  
} else {  
    instrucciones_si_false  
}
```





## 8. Estructuras de control

- **Estructuras de selección**

- if - sintaxis (3)

```
if (expresión_1){  
    instrucciones_1  
} else if (expresión_2){  
    instrucciones_2  
} else {  
    instrucciones_3  
}
```




## 8. Estructuras de control

- **Estructuras de selección**

- switch - sintaxis

```
switch (variable){  
    case valor1:  
        instrucciones a ejecutar si expresión  
        break;  
    case valor2:  
        instrucciones a ejecutar si expresión  
        break;  
    case valor3:  
        instrucciones a ejecutar si expresión  
        break;  
    default:  
        instrucciones a ejecutar si expresión es diferente a los valores anteriores  
}
```



## 8. Estructuras de control

- Estructuras de selección

- switch - sintaxis

```
switch(color) {  
    case 'blanco':  
    case 'amarillo':    // Ambos colores entran aquí  
        colorFondo='azul';  
        break;  
    case 'azul':  
        colorFondo='amarillo';  
        break;  
    default:            // Para cualquier otro valor  
        colorFondo='negro';  
}
```

## 8. Estructuras de control

- Estructuras de selección

- switch - sintaxis

Javascript permite que el *switch* en vez de evaluar valores pueda evaluar expresiones. En este caso se pone como condición *true*:

```
switch(true) {  
  case age < 18:  
    console.log('Eres muy joven para entrar');  
    break;  
  case age < 65:  
    console.log('Puedes entrar');  
    break;  
  default:  
    console.log('Eres muy mayor para entrar');  
}
```

## 8. Estructuras de control

- **Estructuras iterativas**

- Las estructuras de control iterativas o de repetición, inician o repiten un bloque de instrucciones si se cumple una condición o mientras se cumple una condición.
- Las estructuras de iterativas en JavaScript son:
  - while: Ejecuta un grupo de sentencias solo cuando se cumpla una condición
  - for: Ejecuta un grupo de sentencias un número determinado de veces. JS tiene bastante particularidades con este tipo de bucle, destacaremos:
    - for in
    - for of



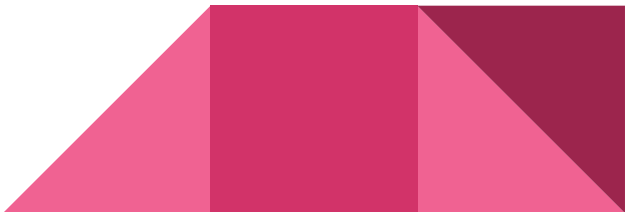
## 8. Estructuras de control

- **Estructuras iterativas**

- Las estructuras de control iterativas o de repetición, inician o repiten un bloque de instrucciones si se cumple una condición o mientras se cumple una condición.
- Las estructuras de iterativas en JavaScript son:
  - while: Ejecuta un grupo de sentencias solo cuando se cumpla una condición
  - for: Ejecuta un grupo de sentencias un número determinado de veces. JS tiene bastante particularidades con este tipo de bucle, destacaremos:
    - for in
    - for of



# 1. Historia de JS

1. Historia de JS
  2. Características de JS
  3. El lenguaje JS: sintaxis
  4. Entrada y salida de datos
  5. Variables
  6. Tipos de datos
  7. Operadores
  8. Estructuras de control
  9. Control de errores
  10. Buenas prácticas
- 

# Bibliografía

- Aprendiendo JavaScript, Carlos Azaustre (2020)
- <https://castilloinformatica.es/>
- <https://sites.google.com/site/desarrollowebcliente/ejercicios/ejercicios-unidad-2?authuser=0>

