



Unidad 4.- JavaScript

Lenguaje de Marcas y SGL.
Miguel Ángel Martí Ferrer

Índice

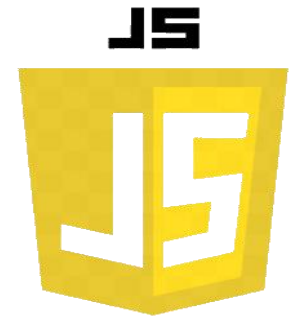
1. Introducción.
2. Historia.
3. Inserción de scripts.
4. Sintaxis.
5. Variables y valores.
6. Entrada y salida de datos.
7. Operadores.
8. Bucles y estructuras.
9. Funciones.
10. Depuración.
11. Objetos.
12. Objetos predefinidos.
13. DOM (Document Object Model)
14. El Objeto Window
15. El Objeto Event
16. El Objeto Document
17. El Objeto Element

Este documento se proporciona con licencia Creative Commons: CC BY-NC-ND.

Se puede usar siempre y cuando se cite al autor, y no se permite obras derivada de ella, ni su uso en proyectos comerciales o con ánimo de lucro.



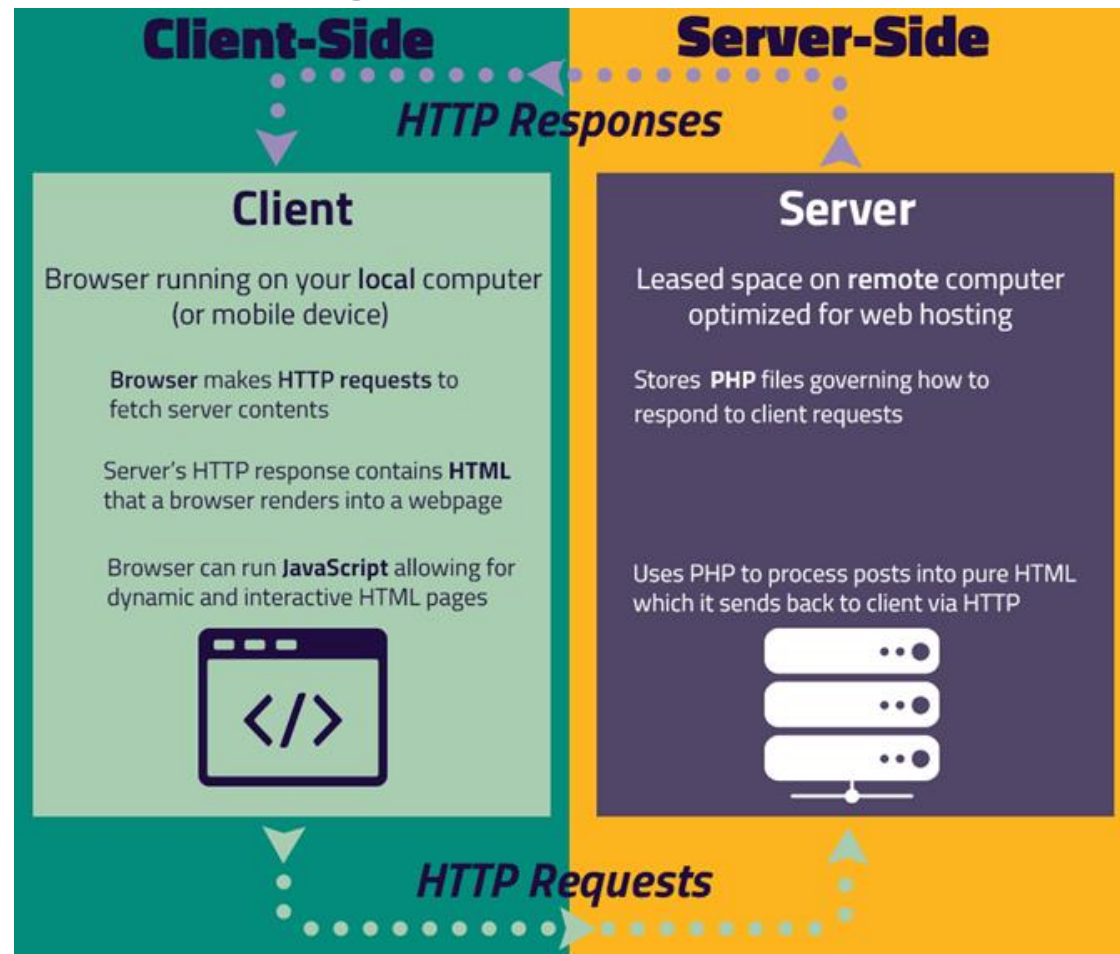
1.- Introducción



- ✓ **Javascript** es un lenguaje de programación que permite ejecutar código en el cliente (nuestro navegador) ampliando la funcionalidad nuestros sitios web.
 - ✓ Permite crear páginas web dinámicas.
 - ✓ no guarda relación con el lenguaje de programación Java.
 - ✓ se integra directamente en las páginas HTML.
 - ✓ es interpretado (sin estar compilado) por el cliente (navegador).
 - ✓ está basado en objetos (que no es lo mismo que POO).
- ✓ Cada navegador incluye su intérprete de JavaScript (Chakra en el caso de Edge, SpiderMonkey en Mozilla Firefox, o V8 de Google Chrome).

1.- Introducción

- ✓ De JavaScript se dice que es un lenguaje del **lado del cliente**, es decir que los scripts son ejecutados por el navegador (cliente).
- ✓ Esto difiere de los llamados lenguajes de script del **lado del servidor** que son ejecutadas por el servidor web (como PHP), cuyo propósito es "crear" la página web que se envía al navegador.





2.- Historia

- ✓ Javascript fue creado por Brendan Eich y lanzado en diciembre de 1995 con el navegador Netscape Navigator 2.
- ✓ ECMA (European Computer Manufacturers Association) estandarizó el lenguaje con el nombre de ECMAScript.
- ✓ Ha tenido varias versiones hasta la actual versión 11 con ECMAScript 2020.

3.- Inserción de scripts

- ✓ Los códigos JavaScript son insertados a través del elemento `<script>`. Con un atributo de tipo que se utiliza para indicar el tipo de lenguaje que vamos a utilizar (no necesario en HTML5)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <link rel="stylesheet" href="css/estilos.css">
</head>
<body>
  Página en HTML4.1
  <script src="js/scripts.js" type="text/javascript"></script>
</body>
</html>
```

- ✓ En `<script>` podemos indicar la fuente del archivo externo o poner directamente código JavaScript.
- ✓ La etiqueta `<script>` se suele poner antes de cerrar `</body>` de esta manera será lo último en cargar.
- ✓ **Ejercicio: crea un esqueleto HTML5 con `<script>`**

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="css/estilos.css">
</head>
<body>
  Página en HTML5
  <script src="js/scripts.js"></script>
</body>
</html>
```

4.- Sintaxis.

- ✓ Instrucciones deben estar separadas por un punto y coma que se coloca al final de cada instrucción:
- ✓ El punto y coma no es necesario si la instrucción siguiente está en la siguiente línea, pero es obligatorio si escriben varias instrucciones en una sola línea.
- ✓ Al igual que HTML o CSS, JavaScript no es sensible a los espacios o tabulaciones.
- ✓ Para los comentarios de una línea se utiliza la doble barra // ... y para comentar varias líneas /* ...*/

```
/*  
Comentario  
multilínea  
*/  
  
sentencia_1;  
sentencia_2;  
sentencia_3; //Comentario de línea
```


5.- Variables y valores.

- ✓ Para declarar las variables en JavaScript se utiliza la instrucción **var**.
- ✓ La instrucción var es opcional, pero se recomienda utilizarla, por ejemplo para buscar variables en el código.
- ✓ El nombre de la variable sólo puede contener caracteres alfanuméricos (sin acentos), “_” o “\$”, debe comenzar por una letra y no puede ser una palabra clave. Además distingue mayúsculas de minúsculas.

```
var ejemplo;  
var resultado;  
ejemplo= "Hola";  
resultado=3+7;
```

```
var miVariable = 5.5; // Los números decimales se separan con un punto  
var miVariable1, miVariable2 = 4, miVariable3;  
var miVariable1, miVariable2;  
miVariable1 = miVariable2 = 2;
```

- ✓ Es posible declarar una variable con **let** en lugar de var. Let asegura que la variable no será redeclarada.

```
let x = "Una cadena";  
let x = 5;  
    alert(x);  
// SyntaxError: Identifier 'x' has already been declared
```


5.1- Tipos de Variables.

- ✓ JavaScript es un lenguaje de tipado dinámicamente (el tipo se determina al darle valor a la variable y puede cambiar)
- ✓ Los tipos de valores que puede contener una variable JavaScript son:
 - ✓ Números: puede contener cualquier tipo de número reales (0.3,1.7,2.9) o enteros (5,3,-1).
 - ✓ Operadores lógicos o boolean: true, false, 1 y 0.
 - ✓ Cadenas de caracteres o String: cualquier combinación de caracteres (letras, números, signos especiales y espacios). Se delimitan mediante comillas dobles o simples ("Cadena",cadena'). Para concatenar cadenas se usa el operador +.

```
//Forma la cadena "Esto es una cadena"  
var cadena = "Esto " + 'es una ' + "cadena";  
alert(cadena);
```

Esta página dice
Esto es una cadena

Aceptar

- ✓ Nota: alert() es una función que saca en una ventana flotante el contenido entre paréntesis. Muy útil para hacer depuración.
- ✓ Ejercicio: declara una variable numérica y otra con una cadena y sácalas en dos ventanas flotantes consecutivas.

5.1- Tipos de Variables.

- ✓ Ejemplos de tipos y tipado dinámico.

```
var edad=23;  
var nombre="Rosa García";  
var frase = nombre+ " tiene "+ edad +" años";  
alert(frase);  
  
var numero = 5.5;  
var numero = 3.65e5; //en notación científica  
var numero= 0x391; // en hexadecimal  
  
var EsVerdadero = true;  
var EsFalso = false;
```

5.2- Arrays

- ✓ Un array (o vector) es una variable que contiene diversos valores.
- ✓ Se crea con “**new Array()**” o inicializando los elementos.
- ✓ Podemos referenciar esos valores indicando su posición en el array con “[]” (el primer elemento tiene la posición 0).
- ✓ Los arrays poseen una propiedad llamada “**length**” que podemos utilizar para conocer el número de elementos del array.

```
// Array definido elemento a elemento
var miVector=new Array();
miVector[0]=1;
miVector[1]=2;
miVector[2]=3;

//Array definido en una linea
var otroArray=[1,2,"tres"];

// Valores dentro del array
alert(miVector[1]);      //muestra: 2
alert(otroArray[2]);     //muestra: tres

// Valor completo del array
alert(miVector);         //muestra: 1,2,3

// Tamaño del array
alert(miVector.length); //muestra: 3

//Una matriz
var matriz = [miVector,miVector,miVector];
alert(matriz[0]);        //muestra: 1,2,3
alert(matriz[0][0]);     //muestra: 1
```

5.3- Conversión de tipos

parseInt ();

- ✓ Función que permite convertir una cadena numérica en un número.

parseFloat ();

- ✓ Función que permite convertir una cadena numérica en un número en coma flotante.

.toString();

- ✓ Método de los tipos numéricos para convertir la variable en una cadena de caracteres.

```
//Conversión a número
var texto = '5432', numero;
alert(texto+1);           // Muestra: « 54321 »
numero = parseInt(texto);
alert(numero+1);          // Muestra: « 5433 »

//Conversión a cadena
var cadena;
cadena = numero +1;
alert(cadena);             // Muestra: « 5433 »
cadena = numero.toString()+1;
alert(cadena);             // Muestra: « 54321 »
```

6- Entrada y salida de datos.

alert(message);

- ✓ Permite mostrar al usuario información literal o el contenido de variables en una ventana independiente.
- ✓ La ventana contendrá la información a mostrar y el botón Aceptar.

```
//Método alert()  
alert("Hello World");
```

Esta página dice

Hello World

Aceptar

6- Entrada y salida de datos.

confirm(message);

- ✓ Activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar.
- ✓ Aceptar devuelve el valor true y cancelar devuelve el valor false.

```
//Método confirm()  
var respuesta;  
respuesta=confirm("¿Desea cancelar la suscripción?");  
alert("Usted ha contestado que "+respuesta)
```

Esta página dice
¿Desea cancelar la suscripción?

Aceptar

Cancelar

Esta página dice
Usted ha contestado que true

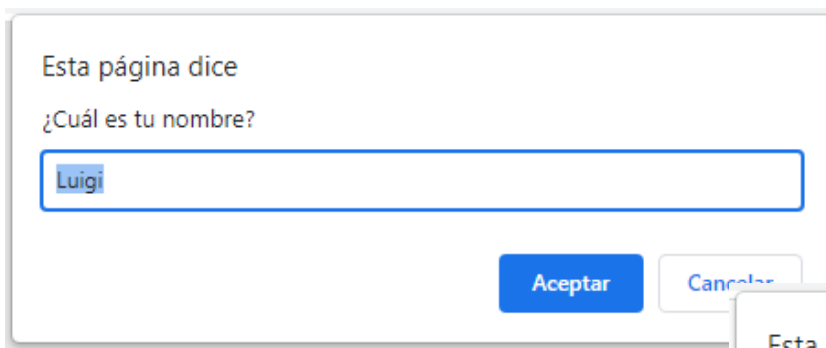
Aceptar

6- Entrada y salida de datos.

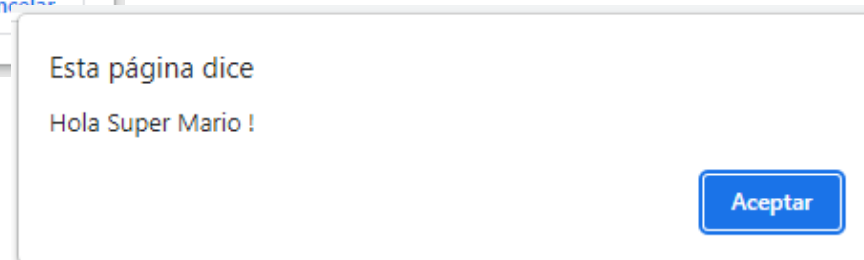
prompt(message, defaultText);

- ✓ Abre un cuadro de diálogo en pantalla en el que se pide al usuario que introduzca algún dato.
- ✓ Si se pulsa el botón Cancelar, el valor de devolución es false/null.
- ✓ Si se pulsa el botón aceptar los datos introducidos se guardan en una variable.
- ✓ defaultText es opcional.

```
//Método prompt()  
var inicio = 'Hola ', nombre, final = ' !';  
nombre= prompt("¿Cuál es tu nombre?", "Luigi");  
alert(inicio + nombre + final);
```



- ✓ **Ejercicio: Pregunta por el nombre de usuario, luego por el apellido y sácalo en una ventana.**



7- Operadores

Operadores aritméticos

- ✓ Los operadores aritméticos se utilizan para hacer cálculos aritméticos.

Operator	Description
+	Suma
-	Resta
*	Multiplicación
**	<u>Exponenciación</u> (ES2016)
/	División
%	Módulo (Resto de la división)
++	Incremento (+1 por defecto)
--	Decremento (-1 por defecto)

7- Operadores

```
//OPERADORES ARITMÉTICOS
//Suma
var resultado = 3 + 2 -1;
alert (resultado) // Muestra « 4 »

//Multiplicación
var numero1=3, numero2 = 2, resultado;
resultado = numero1 * numero2;
alert (resultado); // Muestra: « 6 »

//División y resto
var divisor = 3, resultado1, resultado2, resultado3;
resultado1 = (16 + 8) / 2 - 2; // 10 <-- Si no se ponen paréntesis,
                                //losoperadores se ejecutan de izquierda a derecha.
resultado2 = resultado1 / divisor;
resultado3 = resultado1 % divisor;
alert (resultado2) ; // Resultado de la división: 3,33
alert (resultado3) ; // Resto de la división: 1

//Varias operaciones
var numero = 3;
numero = numero + 5;
alert (numero); // Muestra: « 8 »
alert(8**2); // Muestra: « 64 » (8^2)
alert(numero++); // Muestra: « 8 » (muestra y luego incrementa)
alert(++numero); // Muestra: « 10 » (incrementa y luego muestra)
var hola= 'Hola', nombre = 'tu', resultado;
resultado = hola + nombre;
alert (resultado); // muestra: «Holatu»
```

7- Operadores

Operadores de comparación

- ✓ Los operadores de asignación se utilizan para asignar valores a las variables. Alguno de ellos también incluyen operaciones. En la tabla x=5.

Operator	Description	Comparing	Returns
==	Igual a	x == 8	false
		x == 5	true
		x == "5"	true
===	Igual a en valor y tipo	x === 5	true
		x === "5"	false
!=	no igual	x != 8	true
!==	No gual en valor ni tipo	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	Mayor que	x > 8	false
<	Manor que	x < 8	true
>=	Mayor o igual que	x >= 8	false
<=	Menor o igual que	x <= 8	true

7- Operadores

Operadores lógicos

- ✓ se utilizan para el procesamiento de los valores booleanos. A su vez el valor que devuelven también es booleano: true o false.

Operator	Description
&&	AND (Y lógico)
	OR (O lógico)
!	NOT (NO lógico)

```
document.write("<br>Operación 'false&&false': "+(false&&false));
document.write("<br>Operación 'false&&true': "+(false&&true));
document.write("<br>Operación 'true&&false': "+(true&&false));
document.write("<br>Operación 'true&&true': "+(true&&true));
document.write("<br>Operación 'false||false': "+(false||false));
document.write("<br>Operación 'false||true': "+(false||true));
document.write("<br>Operación 'true||false': "+(true||false));
document.write("<br>Operación 'true||true': "+(true||true));
document.write("<br>Operación '!false': "+(!false));
```

Operación 'false&&false': false
Operación 'false&&true': false
Operación 'true&&false': false
Operación 'true&&true': true
Operación 'false||false': false
Operación 'false||true': true
Operación 'true||false': true
Operación 'true||true': true
Operación '!false': true

7- Operadores

Operadores de bit

- ✓ Realizan operaciones de bit con números de 32 bits. Los operandos numéricos se convierten antes de hacer la operación a un número de 32 bits y se vuelven a convertir en numérico tras la operación.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	1	1
	OR	5 1	0101 0001	101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	10	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	10	2

- ✓ Nota: La operación XOR.

$$A \oplus B: A \cdot \overline{B} + \overline{A} \cdot B \equiv (A + B) \cdot (\overline{A} + \overline{B})$$

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

7- Operadores

Operadores de asignación

- ✓ Los operadores de asignación se utilizan para asignar valores a las variables. Alguno de ellos también incluyen operaciones.

```
//OPERADORES de asignación
var num1=3;
var num2=5;
num2+=num1; //devuelve 8 (5+3)
num2-=num1; //devuelve 5 (8-3)
num2*=num1; //devuelve 15 (5*3)
num2/=num1; //devuelve 5 (15/3)
num2%=num1; //devuelve 2 (resto 5/3)
num2<<=1; //devuelve 4 (10 -> 100)
num2>>=1; //devuelve 2 (100 -> 10)
num2&=3; //devuelve 2 (10AND11 -> 10)
num2|=5; //devuelve 7 (0100R101 -> 111)
num2^=5 //devuelve 2 (111XOR101 ->010)
num2**=2; //devuelve 4 (2^2 -> 4)
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
=	x = y	x = x y
**=	x **= y	x = x ** y

7- Operadores

Operador speed

- ✓ El “speed operator” ... permite copiar todo o parte de un array en otro array.

```
//speed operator
const numeros1 = [1, 2, 3];
const numeros2 = [4, 5, 6];
const numeros = [...numeros1, ...numeros2];

const numeros = [1, 2, 3, 4, 5, 6];
const [uno, dos, ...resto] = numeros;
// uno será =1; dos será =2; el resto será = [3, 4, 5, 6]
```

- ✓ Ejercicio: introduce tu edad y saca la edad que tendrás dentro de 10 años y la edad que tenías hace la mitad de tu vida (haciendo un desplazamiento a la derecha).

8- Bucles y estructuras

if - else

- ✓ La estructura if else permite ejecutar código según la condición entre paréntesis.
- ✓ Las llaves sólo son necesarias si se ha de ejecutar más de una instrucción.
- ✓ Else es opcional.
- ✓ Es posible anidar varios condicionales `if (...) {...} else if (...) {...}`

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false and condition2 is false  
}
```

8- Bucles y estructuras

```
var dia;  
dia=prompt("Introduce el día de la semana ", "");  
if (dia == "domingo")  
    alert("Hoy es festivo");  
else if (dia == "sábado")  
    alert("Hoy no es festivo, pero probablemente no trabajes");  
else  
    alert ("Hoy no es festivo, es muy probable que tengas que trabajar");
```

8- Bucles y estructuras

switch (

- ✓ La estructura switch es una estructura condicional, similar a if. Suele utilizarse cuando se comprueba una condición con múltiples posibilidades.
- ✓ Las sentencia **break** es necesaria para salir del switch. Si no se pone se continúan comprobando case's.
- ✓ No es obligatorio pero sí recomendable utilizar la sentencia **default:** por si no se cumple ninguna condición.

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

8- Bucles y estructuras

```
switch (new Date().getDay()) {  
    case 0:  
        day = "Sunday";  
        break;  
    case 1:  
        day = "Monday";  
        break;  
    case 2:  
        day = "Tuesday";  
        break;  
    case 3:  
        day = "Wednesday";  
        break;  
    case 4:  
        day = "Thursday";  
        break;  
    case 5:  
        day = "Friday";  
        break;  
    case 6:  
        day = "Saturday";  
        break;  
    default:  
        day = "Sunday";  
}
```


8- Bucles y estructuras

Operador ternario

```
variablename = (condition) ? value1:value2
```

- ✓ El **operador ternario** permite ejecutar una condición en una única línea de código.
- ✓ Se ejecuta la condición y si esta es satisfecha devuelve el primer valor, de lo contrario el segundo.

```
var edad=prompt("Introduce tu edad");  
document.write((edad < 18) ? "Menor de edad":"Mayor de edad");
```

Operador OR

- ✓ El **operador OR** puede utilizarse para devolver la primera variable con un valor evaluado true.

```
var conditionTest1='', conditionTest2 = 'Una cadena de caracteres';  
alert (conditionTest1 || conditionTest2); //Devuelve: Una cadena de caracteres
```

- ✓ Nota: document.write genera un nuevo documento HTML.

8- Bucles y estructuras

for

- ✓ El bucle for realiza un conjunto instrucciones un numero de veces.
- ✓ statement1 es ejecutado una vez al inicio.
- ✓ statement2 define la condición para ejecutar el bloque de código. Si no se cumple, sale del bucle.
- ✓ statement3 es ejecutado cada vez después de que el bloque de código se haya ejecutado.

```
for (statement1; statement2; statement3) {  
    // code block to be executed  
}
```

8- Bucles y estructuras

```
document.write("<br>Números pares del 0 al 30: ");
for (i=2; i<30; i+=2) {
    document.write(i+" - ");
}

document.write("<br>Números pares del 30 al 0: ");
for (i=30; i>0; i-=2) {
    document.write(i+" - ");
}

document.write("<br>Potencias de 2 hasta el 3000: ");
for (i=0; (2**i)<3000; i++) {           //Idem: for (i=2; i<3000; i*=2) {
    document.write(2**i+" - ");         //      document.write(i+" - ");
}                                       //      }
```

Números pares del 0 al 30: 2 - 4 - 6 - 8 - 10 - 12 - 14 - 16 - 18 - 20 - 22 - 24 - 26 - 28 -

Números pares del 30 al 0: 30 - 28 - 26 - 24 - 22 - 20 - 18 - 16 - 14 - 12 - 10 - 8 - 6 - 4 - 2 -

Potencias de 2 hasta el 3000: 1 - 2 - 4 - 8 - 16 - 32 - 64 - 128 - 256 - 512 - 1024 - 2048 -

8- Bucles y estructuras

while

- ✓ El bucle ejecuta un grupo de instrucciones mientras se cumpla una condición.
- ✓ Si la condición se cumple siempre, se produce un bucle infinito, que pueden llegar a colapsar el navegador, o incluso el ordenador.

```
while (condition) {  
    // code block to be executed  
}
```

8- Bucles y estructuras

```
var i;
document.write("<br>Números pares del 0 al 30: ");
i = 2;
while (i<30) {
    document.write(i+" - ");
    i+=2;
}

document.write("<br>Números pares del 30 al 0: ");
i=30;
while (i>0) {
    document.write(i+" - ");
    i-=2;
}

document.write("<br>Potencias de 2 hasta el 3000: ");
i=0; //Idem: i=2
while ( (2**i)<3000) { // while (i<3000; ) {
    document.write(2**i+" - "); // document.write(i+" - ");
    i++; // i*=2;
} // }
```

Números pares del 0 al 30: 2 - 4 - 6 - 8 - 10 - 12 - 14 - 16 - 18 - 20 - 22 - 24 - 26 - 28 -

Números pares del 30 al 0: 30 - 28 - 26 - 24 - 22 - 20 - 18 - 16 - 14 - 12 - 10 - 8 - 6 - 4 - 2 -

Potencias de 2 hasta el 3000: 1 - 2 - 4 - 8 - 16 - 32 - 64 - 128 - 256 - 512 - 1024 - 2048 -

8- Bucles y estructuras

do while

- ✓ Es similar al bucle while pero ejecuta al menos el grupo de instrucciones una vez antes de comprobar la condición y esta se comprueba al final.

```
do {  
    // code block to be executed  
}  
while (condition);
```


8- Bucles y estructuras

```
//Bucle do while
//Preguntar por una clave hasta que se introduzca la correcta
var auxclave;
do {
    auxclave=prompt("introduce la clave ","pista: es anónimo")
} while (auxclave!="anónimo")
document.write ("<br>Has acertado la clave");
```

Esta página dice

introduce la clave

Aceptar Cancelar

Has acertado la clave



8- Bucles y estructuras

break; continue;

- ✓ En los bucles for y while se pueden utilizar las instrucciones break y continue para modificar el comportamiento del bucle.
- ✓ La instrucción **break** dentro de un bucle hace que éste se interrumpa inmediatamente, aun cuando no se haya ejecutado todavía el bucle completo. Al llegar la instrucción, el programa se sigue desarrollando inmediatamente a continuación del bucle. Suele tener más sentido en los bucles for, ya que en los while se les puede añadir la condición de parada al inicio.
- ✓ La instrucción **continue** hace retornar a la secuencia de ejecución a la cabecera del bucle, volviendo a ejecutar la condición o a incrementar los índices cuando sea un bucle for. Esto permite saltarse recorridos del bucle.

8- Bucles y estructuras

```
//Pregunta por la clave y permitir tres respuestas incorrectas
var auxclave=true;
var numveces=0;
while (auxclave!= "anónimo")
{
    auxclave=prompt("Introduce la Clave!");
    if ((numveces++)==2) break;
}
if (auxclave=="anonimo") document.write("<br>La clave es correcta");
else document.write("<br>La clave no es correcta correcta");
```

Esta página dice

Introduce la Clave!

La clave no es correcta correcta

```
//Presenta todos los números pares del 0 al 50 excepto los que sean múltiplos de 3
for (s = 0; s < 50; s++) {
    if (s%3==0) continue;
    document.write(s + "- ");
}
```

1- 2- 4- 5- 7- 8- 10- 11- 13- 14- 16- 17- 19- 20- 22- 23- 25- 26- 28- 29- 31- 32- 34- 35- 37- 38- 40- 41- 43- 44- 46- 47- 49-

- ✓ Ejercicio: introduce palabras hasta introducir exit, luego sácalas todas por pantalla una a una.



9- Funciones

- ✓ Una función es un conjunto de instrucciones que se agrupan bajo un nombre de función.
- ✓ Sólo cuando es llamada por su nombre en el código del programa se provoca la ejecución de las órdenes que contiene.
- ✓ Las funciones:
 - ✓ Ayudan a estructurar los programas para hacerlos su código más comprensible y más fácil de modificar.
 - ✓ Permiten repetir la ejecución de un conjunto de órdenes todas las veces que sea necesario sin necesidad de escribir de nuevo las instrucciones.
- ✓ La definición de una función se puede realizar en cualquier lugar del programa, pero se recomienda hacerlo en la cabecera del código HTML o al inicio del código javascript. (justo después de <script>)
- ✓ La llamada a una función se realizará cuando sea necesario, es decir, cuando se demande la ejecución de las instrucciones que hay dentro de ella.

9- Funciones

- ✓ Una función consta de las siguientes partes básicas:
 - ✓ **function** con el nombre de función
 - ✓ Los parámetros pasados a la función separados por comas y entre paréntesis
 - ✓ Si la función debe devolver un valor se hace con **return**.
 - ✓ Las llaves de inicio y final de la función.

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
  
    //if the function must return a value, then:  
    return value;  
}
```

9- Funciones

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <script>

      //Función que devuelve la suma de dos valores que se pasan por parámetros.
      function suma (a,b){ //definición de la función
        return a+b;
      }
      function autora (){
        document.write ("La autora es: Rosa Rodríguez");
      }

      var op1=5;op2=25;
      var resultado;
      resultado=suma(op1,op2); //llamada a la función
      document.write ("op1"+"+"+"op2"+"="+resultado+"<br>");

    </script>
    <a href="javascript:autora()">Ver autora</a>
  </body>
</html>
```

op1+op2=30

Ver autora

La autora es: Rosa Rodríguez

9- Funciones

- ✓ Ahora que ya conocemos las funciones es muy importante diferenciar entre variables globales y locales:
 - ✓ **Variables globales:** pueden utilizarse en cualquier parte del código y son declaradas fuera de toda función con la instrucción var.
 - ✓ **Variables locales:** se definen con la instrucción var dentro de una función y sólo

```
var varGlobal1=10, varGlobal2=20; //Definición variable globales
function prueba(){
  var varLocal1=1; //Definición de variable local
  var varLocal2=varGlobal1+varLocal1;
  var varGlobal2=22;
  //En la función se puede acceder a las variables globales y las locales
  //definidas dentro de ella
  alert ("La suma de la variable local (1) y la variable global 1 (10) es "+ varLocal2+
    " pero hemos sobrescrito la variable gobal 2 a: "+varGlobal2);
}
prueba();
alert ("La variable global 1 es "+varGlobal1+ "y la 2 es "+varGlobal2);
alert ("La variable local 1 es "+varLocal1)
```

Esta página dice

La suma de la variable local (1) y la variable global 1 (10) es 11 pero
hemos sobrescrito la variable gobal 2 a: 22

Aceptar

Esta página dice

La variable global 1 es 10 y la 2 es 20

Aceptar

✖ Uncaught ReferenceError: varLocal1 is not defined [4.9 Variables Globales Locales.html:19](#)
at [4.9 Variables Global... Locales.html:19:71](#)



10- Depuración

- ✓ Para depurar código tenemos las siguientes opciones:
 - ✓ **alert:** mostrar un mensaje flotante. Es la opción menos útil.
 - ✓ **console.log:** Extraer la información que queremos depurar con console.log.
 - ✓ **debugger:** Utilizar la instrucción debugger; Esto indicará al navegador que debe parar la ejecución para depuración.
 - ✓ **breakpoints:** una vez el navegador ha parado la ejecución es posible añadir breakpoints en el código que muestra el navegador.
 - ✓ **Ide:** Es posible que nuestro IDE permita la depuración de JavaScript, probablemente instalando algún plugin para algún navegador específico.

10- Depuración

The screenshot displays the Chrome DevTools interface with the 'Console' tab selected. The 'Sources' panel shows a file named '4.10 Depuracion.html' with the following code:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5   </head>
6   <body>
7     <script>
8
9       function pruebaDebugger(){
10         var variable = 5;
11         alert(variable); variable = 5
12
13         console.log(variable); variable = 5
14
15         debugger;
16
17         variable+=5;
18
19         return variable;
20       }
21       document.write(pruebaDebugger());
```

The code is paused at line 15, which contains the `debugger;` statement. A context menu is open over this line, with the option 'Add breakpoint' highlighted. The right sidebar shows the 'Debugger paused' state with the following information:

- Debugger paused**
- Watch**
- Breakpoints**: No breakpoints
- Scope**
 - Local**
 - `this: Window`
 - `variable: 5`
 - Global**: Window
- Call Stack**
 - `pruebaDebugger` at 4.10 Depuracion.html:15
 - `(anonymous)` at 4.10 Depuracion.html:21
- XHR/fetch Breakpoints**
- DOM Breakpoints**
- Global Listeners**
- Event Listener Breakpoints**
- CSP Violation Breakpoints**

The bottom status bar shows 'Coverage: n/a'.

A decorative vertical bar on the left side of the slide, composed of numerous overlapping circles of various sizes and colors, including blue, green, yellow, orange, and pink, creating a vibrant, bubbly effect.

Práctica

- ✓ Realiza la práctica 4.1

11. Objetos

- ✓ Los objetos son los más similar a las clases en Java, que tienen métodos y propiedades.
- ✓ Las propiedades del objeto se definen como conjuntos de **clave:valor**, separados por “,”
- ✓ Los métodos también se definen como conjunto de clave:valor, solo que la clave es el nombre de la función y el valor es la función **function(){...}**
- ✓ Es común definir los objetos como const. El siguiente es un ejemplo de definición del objeto persona.

```
const coche = {  
  marca: "Tesla",  
  modelo: "Model S",  
  años: 50,  
  marca_modelo : function() {  
    return this.marca + " " + this.modelo;  
  }  
};
```

OJO: los objetos en JavaScript son mutables. Si definimos `const x = coche;` y hacemos cambios en `x`, también se harán en `coche`.

11. Objetos

Creación de objetos. Hay diferentes formas de crear nuevos objetos:

- ✓ Crear un solo objeto, utilizando un objeto literal.
- ✓ Crear un solo objeto, con la palabra clave new.
- ✓ Definir un constructor de objetos y luego crear objetos del tipo construido con **this**. (Nota: this hará referencia a diferentes objetos dependiendo del contexto).
- ✓ Crear un objeto usando Object.create().

```
//utilizando un objeto literal
const coche = {marca: "Tesla", modelo : "Model S", años: 3, color:"azul"};
//utilizando un objeto literal vacío y añadiendo propiedades.
const coche = {};
coche.marca: "Tesla";
coche.modelo : "Model S";
coche.años: 50;
coche.color:"azul";
//Utilizando la palabra clave new. (Similiar al anterior)
const coche = new Object();
coche.marca: "Tesla";
coche.modelo : "Model S";
coche.años: 50;
coche.color:"azul";
//Utilizando un constructor
function coche(marca, modelo, años, color) {
    this.marca = marca;
    this.modelo = modelo;
    this.años = años;
    this.color = color;}
//Utilizando Object.create. (No es necesario inicializar variables)
const coche = {marca: "", modelo : "", años: 0, color:""};
const yo = Object.create(coche);
yo.marca = "Tesla";
```


11. Objetos

Acceso a los atributos y métodos

- ✓ Se puede acceder a las propiedades con “.” o con “[...]”. A los métodos se accede con “.”.

```
//Acceder a las propiedades y métodos:  
coche.marca = "Tesla";  
coche["marca"] = "Tesla";  
coche.marca_modelo();
```

- ✓ Es posible acceder a todas la propiedades con un bucle for ... in:

```
for (let x in coche) {  
    document.write(coche[x]);  
}
```

11. Objetos

Acceso a los atributos y métodos

- ✓ Es posible agregar nuevas propiedades y métodos a un objeto existente simplemente dándole un valor. También es posible borrar las propiedades (no los métodos) con la palabra clave **delete**. (delete no tiene efecto sobre variables o funciones, y no debe usarse en propiedades de objetos de JavaScript predefinidos.)

```
coche.matricula = "2719-DRY";  
delete coche.matricula; //idem: delete coche[matricula];
```

- ✓ Además permite el uso de palabras clave **get** y **set** para fijar y recoger valores de atributos. Son similares al uso de métodos, pero no utilizan “()” al acceder a ellos. También es posible fijarlos con `Object.defineProperty`

```
//get y set  
const coche = {  
  //...  
  idioma: "";  
  set leng(leng) {this.idioma = leng;}  
  get leng() {return this.idioma;}  
};  
coche.leng = "es";  
document.write(coche.leng);  
  
//Otra forma de hacer get/set  
Object.defineProperty(coche, "leng", {  
  get : function () {return this.language;}  
});
```

11. Objetos

Creación de objetos de un tipo

- ✓ Hasta ahora hemos visto cómo crear objetos. Es posible crear varios objetos del mismo tipo (similar a instanciar clases Java).
- ✓ La forma de crear un tipo de objeto es utilizar un constructor.

```
//Creación de tipos de objetos
function coche(marca, modelo, años, color) {
  this.marca = marca;
  this.modelo = modelo;
  this.años = años;
  this.color = color;
  this.marca_modelo = function() {
    return this.coche + " " + this.modelo;
  }
}

const tesla = new coche("Tesla", "Model S", 1, "blanco");
const bmw = new coche("BMW", "Serie 1", 2, "azul");
```

- ✓ Es posible añadir propiedades al objeto después de crearlo, aunque sólo se añade la objeto, no al tipo de objeto. Para añadir métodos es necesario hacerlo en el constructor.



11. Objetos

- ✓ Dejamos aquí los objetos. No se profundizará en los siguientes aspectos de los objetos.
 - ✓ Prototipos: permiten añadir propiedades y métodos en el tipo de objeto.
 - ✓ Iterables: objetos que tienen un conjunto de elementos y permiten el uso del bucle **for of**. Necesitan tener definidos los métodos `next()`, y las propiedades `done` y `value`.
 - ✓ Set: objetos que tienen un conjunto de elementos únicos y pueden ser recorridos con la función **forEach()**;
 - ✓ Map: objetos que tienen un conjunto de elementos a los cuales pueden accederse mediante clave-valor y pueden ser recorridos con la función **forEach()**;



12- Objetos predefinidos

- ✓ Entre otras posibilidades, Javascript es un lenguaje que de forma nativa posee gran cantidad de funciones y objetos predefinidos.
- ✓ Como su nombre indica, los objetos predefinidos son objetos que ya están definidos en javascript con sus funciones y propiedades que pueden utilizarse.
- ✓ Estos objetos con sus funciones y propiedades nos pueden ser útiles para realizar un código mas eficiente y claro:
 - ✓ String
 - ✓ Date
 - ✓ Array
 - ✓ Math

12- Objetos predefinidos

Funciones predefinidas.

Son funciones que pueden ser utilizadas sin definir las por el programador. Habíamos visto algunas de ellas:

- ✓ **parseInt(cadena)**: convierte una cadena de texto entero a entero .
- ✓ **parseFloat(cadena)**: convierte una cadena de texto a decimal.
- ✓ **isNaN(cadena)**: NaN es la abreviación de “Not a Number”. Esta función comprueba si una cadena de caracteres puede ser considerada un número (false) o no (true).
- ✓ **eval(cadena)**: recibe una cadena y la interpreta como código Javascript. También vale para expresiones numéricas, donde te devuelve el resultado. Permite por ejemplo evaluar expresiones de cadenas sin pasarlas a numéricos.

```
var x = prompt("Introduce un número");  
var y = prompt("Introduce un número");  
document.write("Resultado de la suma: "+eval(x+"+"+y));
```


12- Objetos predefinidos

string

Cuando creamos una cadena, implícitamente esta cadena se convierte en un objeto String, con sus propiedades y métodos predefinidos. Los métodos de cadenas de Javascript, no modifican al objeto actual, sino que devuelven el resultado de aplicar el método. Estos son algunos métodos y propiedades (hay más, se pueden consultar en el documento de referencia de JavaScript)

- ✓ **length**: devuelve el tamaño del array.
- ✓ **toLowerCase()/toUpperCase()**: devuelve la cadena convertida a minúsculas/mayúsculas.
- ✓ **concat(cadena)**: devuelve el objeto con el valor de cadena concatenado al final.
- ✓ **charAt(posicion)**: devuelve el carácter que se encuentre en la posición (comenzando desde cero).
- ✓ **indexOf(texto, [indice])**: devuelve la primera posición donde se encuentra el texto buscado, empezando (opcionalmente) a buscar desde la posición "indice".
- ✓ **lastIndexOf (texto, [indice])**: como la anterior. Busca "hacia atrás" la primera ocurrencia del texto buscado, empezando (opcionalmente) a buscar desde la posición "indice".
- ✓ **replace(texto1,texto2)**: busca ocurrencias de la cadena texto1 y las reemplaza por texto2.
- ✓ **split(caracter, [separaciones])**: separa la cadena mediante un carácter separador, opcionalmente cuántas separaciones debe devolver.
- ✓ **substring(inicio, [fin])**: devuelve la sub cadena comprendida entre la posición inicio y el final (o la posición fin si se especifica).

12- Objetos predefinidos

```
var cad="Sonic:The hedgehog:123456";
var tfo;
cad=cad.toUpperCase();
alert(cad);                //muestra: SONIC:THE HEDGEHOG:123456
splitTodosCampos=cad.split(":");
alert(splitTodosCampos);   //muestra: SONIC,THE HEDGEHOG,123456
split1Campo=cad.split(":",1);
alert(split1Campo);        //muestra: SONIC
tfo=splitTodosCampos[2];
//Cambio en el telefono los números 3 por 9s
tfo=tfo.replace("2","9");
alert(tfo);                //muestra: 193456
//Muestro el quinto numero del teléfono
alert(tfo.charAt(4));      //muestra: 5
```

12- Objetos predefinidos

date

Date es un objeto predefinido que nos permite trabajar con fechas. Para crear un objeto date se admiten múltiples formatos. Algunos de los métodos mas importantes del objeto Date:

- ✓ **set/get:** son métodos que permite cambiar el valor de alguna parte de la fecha (set) u de obtener el valor de alguna parte de la fecha (get):
setMonth(mes), getMonth(); setDate(dia), getDate(),
setHours(hora,minuto,segundo), getHours(), etc.
- ✓ **toString():** convierte la fecha del objeto a una cadena en objeto fecha.
- ✓ **toUTCString():** convierte la fecha del objeto en una cadena con formato de fecha UTC.
- ✓ **toLocaleTimeString():** Devuelve la parte de la hora de un objeto Date como una cadena, utilizando las convenciones locales.
- ✓ **toLocaleString()** Convierte un objeto de fecha en una cadena, usando convenciones locales.
- ✓ Nota: para get/set, los meses en Date se numeran del 0 al 11 (siendo el 0 Enero y el 11 Diciembre). Los días se numeran del 1 al 31. Los días de la semana se numeran del 0 al 6, siendo el 0 domingo y el 6 sábado.

12- Objetos predefinidos

```
var d;  
// Crea una fecha basándose en la cadena de fecha especificada  
d=new Date("January 13, 2019 11:13:00");  
// Crea una fecha indicando año, mes, día, horas, minutos, segundos milisegundos  
d=new Date(2019,1,30,11,33,30,0);  
// Crea una fecha indicando año, mes, día.  
d=new Date(2019,1,30);  
// Crea una fecha con la fecha y hora del sistema  
var d=new Date();  
alert(d); //muestra: Sat Apr 02 2022 12:22:20 GMT+0200  
           //(hora de verano de Europa central)  
  
alert(d.getDay()); //muestra: 6  
d.setDate(1);  
alert(d.getDay()); //muestra: 5  
alert(d.toUTCString()); //muestra: Fri, 01 Apr 2022 10:22:20 GMT  
alert(d.toLocaleTimeString()); //muestra: 12:23:52  
alert(d.toLocaleDateString()); //muestra: 1/4/2022
```

12- Objetos predefinidos

array

array es un objeto para el manejo de arrays. Permite albergar: date, string, números enteros, decimales, otros Arrays, etc...). Generalmente los métodos aplicados en el Array **sí** modifican el objeto original, a no ser que solamente devuelvan un resultado. Estos son algunos de los métodos más utilizados.

- ✓ **join([separador]):** devuelve una cadena con todos los elementos del array, separados por el texto que se incluya en separador.
- ✓ **concat(array):** concatena dos o más arrays y devuelve una copia.
- ✓ **push(elemento1, elemento2, ...):** añade al final los elementos en el orden proporcionado.
- ✓ **shift()/pop():** devuelve y elimina el primer/último elemento del array.
- ✓ **reverse():** invierte el orden de los elementos de un array.
- ✓ **sort():** ordena los elementos de un array alfabéticamente. Ojo, no numéricamente, 10 va antes que 2.
- ✓ **slice(inicio, [final]):** devuelve los elementos de un array comprendidos entre inicio y ultimo elemento, o el final especificado (opcional).
- ✓ **every():** comprueba que todos los elementos satisfacen una condición
- ✓ **filter():** devuelve una array con todos los elementos satisfacen una condición.
- ✓ **find()/findIndex():** devuelve el primer elemento que satisface una condición y su índice.
- ✓ Nota: para comprobar si se satisface una condición se hace mediante una función a la que se le pasa el valor (obligatorio), y el índice o el array (opcional)

12- Objetos predefinidos

```
var x=["Lunes","Martes","Miércoles","Jueves","Viernes","Sábado","Domingo"];
var y=[2,3,10,5];
y.reverse();
alert(y);           //muestra: 5,10,3,2
y.sort();
alert(y);           //muestra: 10,2,3,5
y=y.slice(1,3);
alert(y);           //muestra: 2,3

alert(x.join("-")); //muestra: Lunes-Martes-Miércoles-Jueves-Viernes-Sábado-Domingo
alert(x.pop());     //muestra: Domingo
alert(x.shift());   //muestra: Lunes
alert(x);           //muestra: Martes,Miércoles,Jueves,Viernes,Sábado
x=["Lunes"].concat(x,"Domingo"); //Lo devolvemos al estado original

function compruebaTamaño(val){
    return val.length>0?true:false;
}
function contieneE(val){
    return val.includes("e"?true:false;
}

alert("Todos los elemento contienen algo? "+x.every(compruebaTamaño));
//muestra: Todos los elemento contienen algo? true
alert("Días de la semana que contienen e: "+x.filter(contieneE));
//muestra: Días de la semana que contienen e: Lunes,Martes,Miércoles,Jueves,Viernes
alert("Primer día de la semana que contiene e: "+x.find(contieneE)+
    " en la posición: "+x.findIndex(contieneE));
//muestra: Primer día de la semana que contiene e: Lunes en la posición: 0
```

12- Objetos predefinidos

math

El objeto Math es un objeto que contiene una colección métodos que nos ayudarán a trabajar realizar operaciones aritméticas, redondeos, etc.

Algunas de las constantes matemáticas predefinidas más importantes son:

- ✓ **E**: almacena el número de Euler.
- ✓ **PI**: almacena el número Pi.
- ✓ **LN2 / LN10** : logaritmo neperiano de 2 / 10.
- ✓ **LOG2E / LOG10E** : logaritmo en base 2 / 10 del número de Euler.

Algunos de los métodos mas importantes som:

- ✓ **floor(numero)**: redondea hacia abajo un número.
- ✓ **ceil(numero)**: redondea hacia arriba un número.
- ✓ **round(numero)**: redondea al entero mas cercano.
- ✓ **abs(numero)**: devuelve el número en valor absoluto.
- ✓ **max / min (x,y)**: devuelve el mayor / menor de dos números x e y.
- ✓ **pow(x,y)**: devuelve x elevado a y.
- ✓ **random()**: devuelve un número aleatorio con decimales entre 0 y 1.
- ✓ **sqrt(numero)**: devuelve la raíz cuadrada del número indicado.

12- Objetos predefinidos

```
// Obtenemos un entero entre 1 y 11
document.write("<br>" + Math.random()*10+1);
//Redondeos
var x = 4.5;
document.write("<br>" + Math.round(x));
document.write("<br>" + Math.floor(x));
document.write("<br>" + Math.ceil(x));
document.write("<br>" + Math.trunc(x));
//potencias y raices
document.write("<br>" + Math.pow(9, 2));
document.write("<br>" + Math.sqrt(81));
//funciones trigonométricas
document.write("<br>" + Math.sin(90 * Math.PI / 180));
document.write("<br>" + Math.cos(0 * Math.PI / 180));
//máximos y mínimos
document.write("<br>" + Math.min(0, 150, 50, 200, -100, -200));
document.write("<br>" + Math.max(0, 150, 50, 200, -100, -200));
//logaritmos
document.write("<br>" + Math.log(0));
document.write("<br>" + Math.log(10));
document.write("<br>" + Math.log2(2));
document.write("<br>" + Math.log2(8));
```

1.12940007264626941
5
4
5
4
81
9
1
1
-200
200
-Infinity
2.302585092994046
1
3



Práctica

- ✓ Realiza la práctica 4.2

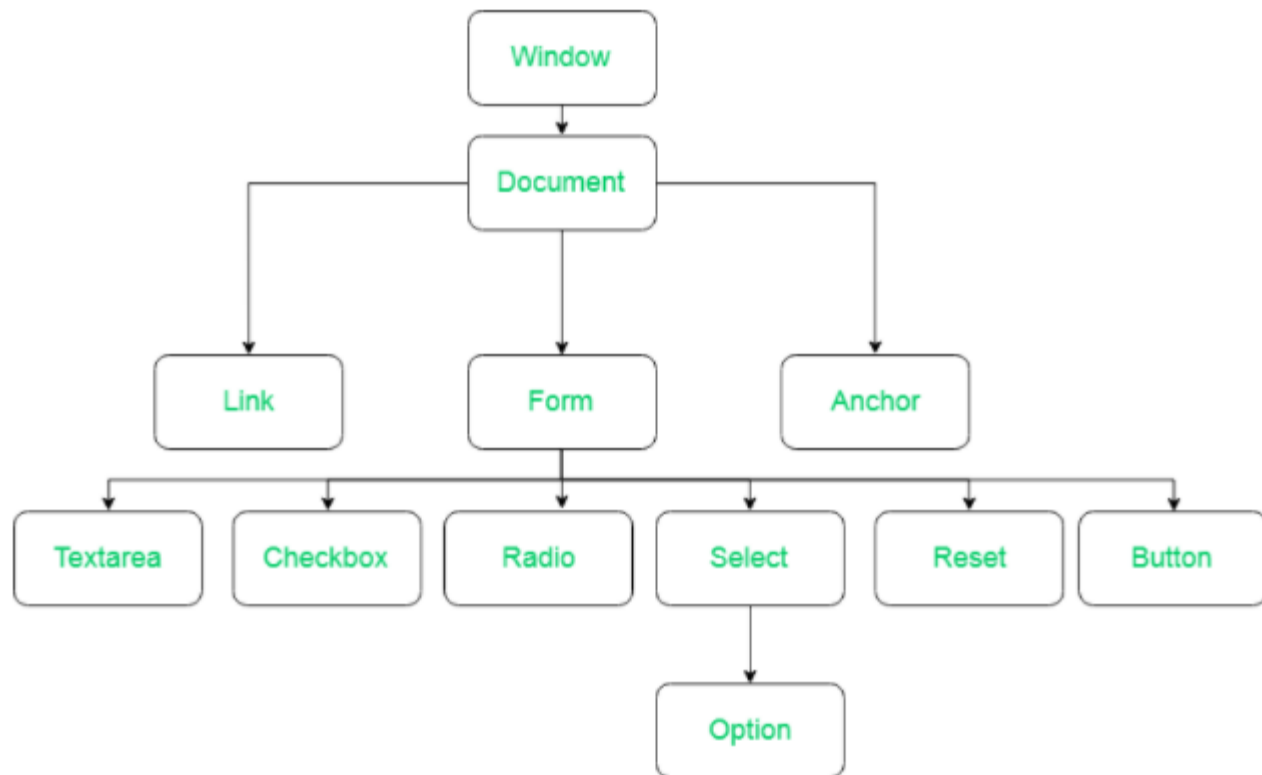


13- DOM (Document Object Model)

- ✓ JavaScript es un lenguaje que permite crear lo que se llama DHTML (Dinamic HTML), las páginas web pueden modificar ellas mismas sus propios contenidos sin cargar una nueva página.
- ✓ El Modelo de Objetos de Documento (**DOM**) es una interfaz de programación (API) que se utiliza con documentos XML y HTML, lo que nos permitirá, a través de Javascript, acceder a documentos en XML y / o HTML y cambiar los elementos HTML, añadir, mover o incluso eliminar.

13- DOM (Document Object Model)

- ✓ Estas son las propiedades del objeto document que se van a poder acceder y modificar.



Representation of the DOM

13- DOM (Document Object Model)

DOM no se recoge en un único documento, sino que consiste en tres niveles, y cada uno de ellos está a su vez compuesto por varias recomendaciones referidas a distintos aspectos de la interfaz:

✓ DOM 1:

- ✓ **DOM Core:** proporciona interfaces de bajo nivel que se pueden utilizar para representar cualquier documento estructurado.
- ✓ **DOM HTML:** Define una serie objetos y métodos específicos de un documento HTML, y que por extensión también pueden aplicarse a un XHTML.

✓ DOM 2:

- ✓ **CORE2:** amplía la funcionalidad de CORE especificada por DOM nivel 1.
- ✓ **VIEWS:** las vistas permiten que los programas accedan y manipulen dinámicamente el contenido del documento.
- ✓ **EVENTS:** Los eventos son scripts que ejecuta el navegador cuando el usuario reacciona a la página web.
- ✓ **STYLE:** permite que los programas accedan y manipulen dinámicamente el contenido de las hojas de estilo.
- ✓ **TRAVERSAL:** Esto permite que los programas atraviesen dinámicamente el documento.
- ✓ **RANGE:** Esto permite que los programas identifiquen dinámicamente un rango de contenido en el documento.



13- DOM (Document Object Model)

✓ DOM 3:

- ✓ **CORE3:** amplía la funcionalidad de CORE especificada por DOM nivel 2.
- ✓ **LOAD y SAVE:** Esto permite que el programa cargue dinámicamente el contenido del documento XML en el documento DOM y guarde el documento DOM en un documento XML por serialización.
- ✓ **VALIDATION:** Esto permite que el programa actualice dinámicamente el contenido y la estructura del documento mientras asegura que el documento sigue siendo válido.
- ✓ **EVENTS:** amplía la funcionalidad de los eventos especificados por DOM Nivel 2.
- ✓ **XPATH:** XPATH es un lenguaje de ruta que se puede usar para acceder al árbol DOM.



14- El Objeto Window

window

- ✓ El objeto window representa una ventana abierta en el navegador.
- ✓ Si la página tiene un `<iframe>`, se crea un objeto window diferente para cada frame.
- ✓ Los métodos que estudiamos en puntos anteriores como `alert`, `prompt`, etc. forman parte del objeto Window, solo que no es necesario nombrar explícitamente Window (lo hace el navegador).

14- El Objeto Window

window

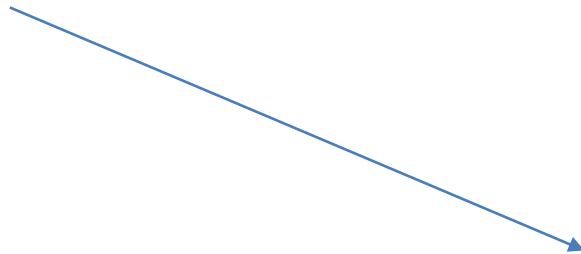
Algunos de los métodos y propiedades más importantes no estudiados previamente son:

- ✓ **setTimeout**("function", milliseconds): ejecuta la llamada a la función (con o sin parámetros) proporcionada por la cadena y la ejecuta pasados X milisegundos. Devuelve un identificador del "setTimeout" por si deseamos cancelarlo con **clearTimeout**(id). Solo ejecuta la orden una vez.
- ✓ **setInterval**("function", milliseconds): similar al anterior, pero se repite cíclicamente cada X milisegundos. Es posible cancelarlo con **clearInterval** (id).
- ✓ **open()**: abre una nueva ventana en una nueva pestaña.
- ✓ **print()**: imprime el contenido del documento.
- ✓ **innerHeight/innerWidth** : Devuelve la altura y anchura del área de contenido de la ventana
- ✓ **screen**: Devuelve el objeto Screen de la ventana para extraer parámetros de resolución, tamaño, etc.
- ✓ **screenLeft/screenTop**: Devuelve la coordenada horizontal/vertical de la ventana relativa a la pantalla.

14- El Objeto Window

```
function obtenerResolucion (){  
    document.write("La pantalla tiene: "+screen.width+"pxX"+screen.height+"px");  
}  
  
document.write("La ventana del navegador tiene: "+innerWidth+"pxX"+innerHeight+"px");  
const myTimeout = setTimeout(obtenerResolucion,3000);
```

La ventana del navegador tiene: 570pxX889px



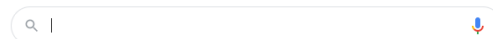
La pantalla tiene: 1280pxX960px

14- El Objeto Window

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function reloj() {
        var fObj = new Date() ;
        var horas = fObj.getHours() ;
        var minutos = fObj.getMinutes() ;
        var segundos = fObj.getSeconds() ;
        if (horas <= 9) horas = "0" + horas;
        if (minutos <= 9) minutos = "0" + minutos;
        if (segundos <= 9) segundos = "0" + segundos;
        document.body.innerHTML = "";
        document.write(horas+":"+minutos+":"+segundos);
      }
    </script>
  </head>
  <body>
    Esta página muestra un reloj y tras 5 segundos carga la página de Google.
    <script>
      var id =setInterval("reloj()",1000);           //Mostraremos un reloj que...
      setTimeout("clearInterval("+id+")",3000);      //...parará de contar a los 3 segundos

      //Además a los 5 segundos cambiará de página
      setTimeout("document.location.href='http://www.google.es';",5000);
    </script>
  </body>
</html>
```

20:41:59





15- El objeto Event

- ✓ Un evento es la consecuencia de una acción del usuario sobre algún objeto de la página web (hover, actívale, focus, etc).
- ✓ Desde el código JavaScript se pueden identificar los eventos y actuar en función de donde y cuando sucedan, por eso podemos decir que los eventos son las interfaces entre el archivo HTML y el código fuente JavaScript .
- ✓ Vamos a ver algunos, pero se proporciona una lista con todos los eventos en documento de referencia de JavaScript.

15- El objeto Event

onLoad y onUnload

- ✓ El evento **onLoad** se ejecuta cuando se termina de cargar completamente el elemento en el que está definido. Puede definirse en: `<body>`, `<frame>`, `<iframe>`, ``, `<input type="image">`, `<link>`, `<script>`, `<style>`.
- ✓ El evento **onUnload** se ejecuta cuando se sale de una página web (pinchando en un enlace, enviando un formulario, cerrando el navegador, recargando la página, etc). Puede definirse en: `<body>`.

15- El objeto Event

```
<!DOCTYPE html>
<html>
  <head>
    <script >
      var nombre;
      function pedirNombre(){
        nombre=prompt("Introduce tu nombre.");
      }
      function despedida(){
        alert("Adios, "+nombre+"esperamos verte pronto!");
      }
    </script>
  </head>
  <body onLoad="pedirNombre()" onUnload="despedida()">
  </body>
</html>
```

Esta página dice

Introduce tu nombre.

Aceptar

Cancelar

✓ Nota: OnUnload no funciona bien con alertas.

15- El objeto Event

onChange

- ✓ El evento **onChange** detecta el cambio de contenido en un elemento de formulario en el momento en que este pierde su foco.
- ✓ Este evento se puede aplicar a los siguientes objetos: `<input type="file">`, `<select>`, `<input type="text">` y `<textarea>`.

```
<!DOCTYPE html>
<html>
  <head>
    <script >
      function MTexto (){
        alert("Ha modificado el campo de texto");
      }
    </script>
  </head>
  <body>
    <form>
      <input type="text" size="40" maxlength="80"
        name="nombre" onChange="MTexto()">
    </form>
  </body>
</html>
```

cambiamos texto y salimos de foco

Esta página dice

Ha modificado el campo de texto

Aceptar

15- El objeto Event

onClick y onDbIClick

- ✓ El evento **onClick** se activa cuando el usuario pulsa el ratón sobre un elemento de formulario o un enlace.
 - ✓ El evento **onDbIClick** se ejecuta cuando el usuario pulsa dos veces con el ratón en un elemento de formulario o enlace.
 - ✓ Funcionan sobre cualquier elemento excepto: <base>, <bdo>,
, <head>, <html>, <iframe>, <meta>, <param>, <script>, <style>, and <title>.
-
- ✓ Nota: <base> especifica una URL base. Otros enlaces pueden hacer referencia a una ruta incompleta cuyo inicio sera <base>. <bdo> (Bi-Directional Override) permite especificar la dirección del texto de derecho a izquierda.

15- El objeto Event

```
<!DOCTYPE html>
<html>
  <head>
    <script >
      function pincha1(){
        alert("Haz un doble click!");
      }

      function pincha2(){
        alert("Te ha tocado un café gratis.");
      }
    </script>
  </head>
  <body>

    <br><br><br><br><br><br>

  </body>
</html>
```



Esta página dice

Haz un doble click!

Aceptar

Esta página dice

Te ha tocado un café gratis.

Aceptar



15- El objeto Event

onmouseenter, onmouseover, onmouseout, onmousemove

- ✓ El evento **onmouseenter** se activa cuando el usuario mueve el ratón en un objeto.
- ✓ El evento **onmouseover** se activa cuando el usuario mueve el ratón en un objeto o en alguno de sus hijos.
- ✓ El evento **onmouseout** se activa cuando el usuario mueve el ratón fuera de un objeto.
- ✓ El evento **onmousemove** se activa cada vez que el usuario mueve el ratón dentro de un objeto.

15- El objeto Event

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      #general{width: 400px;height: 400px;}
      div{width: 200px;height: 105px; border: 3px solid black; text-align: center;margin: auto;}
    </style>

    <script>
      var x = 0;
      var y = 0;
      var z = 0;

      function myMoveFunction() {z+=1;}
      function myEnterFunction() {x+=1;}
      function myOverFunction() {y+=1;}
      function myLeaveFunction(){
        document.write("<br># veces onmousemove:"+z+"<br># veces onmouseenter:"
          +x+"<br># veces onmouseover:"+y);
      }
    </script>
  </head>
  <body>

    <div id="general" onmouseleave="myLeaveFunction()">
      onmouseleave:Al salir de este div general se mostrarán los resultados.
      <div onmousemove="myMoveFunction()">
        <p>onmousemove: se activa cuando se mueve el ratón.</p>
      </div>

      <div onmouseenter="myEnterFunction()">
        <p>onmouseenter: se activa cuando el ratón entra en el elemento.</p>
      </div>

      <div onmouseover="myOverFunction()">
        <p>onmouseover: se activa cuando el ratón se mueve en el elemento o sus hijos.</p>
      </div>
    </div>
  </body>
</html>
```

✓ Mejor ver en directo!



16- El Objeto Document

document

- ✓ Cuando un documento HTML se carga en un navegador web, se convierte en un objeto de documento.
- ✓ El objeto del documento es el nodo raíz del documento HTML.
- ✓ El objeto de documento es una propiedad del objeto de ventana, pero puede ser accedido directamente.

16- El Objeto Document

document

Algunos de los métodos y propiedades más importantes son:

- ✓ **getElementById():** Devuelve el elemento que tiene el atributo ID con el valor especificado.
- ✓ **getElementsByClassName():** Devuelve una HTMLCollection que contiene todos los elementos con el nombre de clase especificado.
- ✓ **getElementsByTagName():** Devuelve una lista de nodos en vivo que contiene todos los elementos con el nombre especificado.
- ✓ **getElementsByTagName():** Devuelve una HTMLCollection que contiene todos los elementos con una etiqueta (<...>) específica.
- ✓ **hasFocus():** Devuelve un valor booleano que indica si el documento tiene foco.
- ✓ **addEventListener()/removeEventListener():** Añade o quita un controlador de eventos al documento. Los eventos se ponen entre “...” y sin on: “change”, “click”, “update”, etc.
- ✓ **Images:** Devuelve una colección de todos los elementos del documento
- ✓ **Links:** Devuelve una colección de todos los elementos <a> y <area> del documento que tienen un atributo href



17- El Objeto Element

element

- ✓ El objeto element representa cualquier nodo elemento definido en el HTML (<p>, <div>, <a>, <table>).
- ✓ Son nodos aunque no elementos el texto o los comentarios.
- ✓ Accediendo a un elemento a través de alguna función del objeto document vamos a poder acceder y modificar sus propiedades.

17- El Objeto Element

element

Algunos de los métodos y propiedades más importantes son:

- ✓ **addEventListener()/ removeEventListener()**: Adjunta/ Elimina un controlador de eventos a un elemento. Los eventos se ponen entre “...” y sin on: “change”, “click”, “update”, etc. Además se pasa la función a ejecutar.
- ✓ **appendChild()**: Agrega (anexa) un nuevo nodo secundario a un elemento
- ✓ **children**: devuelve una lista de elementos hijos.
- ✓ **childNodes**: devuelve una lista de nodos (elementos y no elementos) hijos.
- ✓ **classList**: Devuelve los nombres de clase de un elemento
- ✓ **className/id/nodeName/nodeValue**: Establece o devuelve el valor del atributo clase/id/nombre/valor de un elemento
- ✓ **clientHeight/clientWidth**: Devuelve el alto/ancho de un elemento, incluido el relleno
- ✓ **contains()**: Devuelve verdadero si un nodo es descendiente de un nodo
- ✓ **firstChild/lastChild**: Devuelve el primer/último nodo secundario de un elemento
- ✓ **focus()**: Da foco a un elemento
- ✓ **getAttribute()**: Devuelve el valor del atributo de un elemento
- ✓ **hasAttribute()**: Devuelve verdadero si un elemento tiene un atributo dado
- ✓ **innerHTML**: Establece o devuelve el contenido de un elemento
- ✓ **remove()**: Elimina un elemento del DOM.

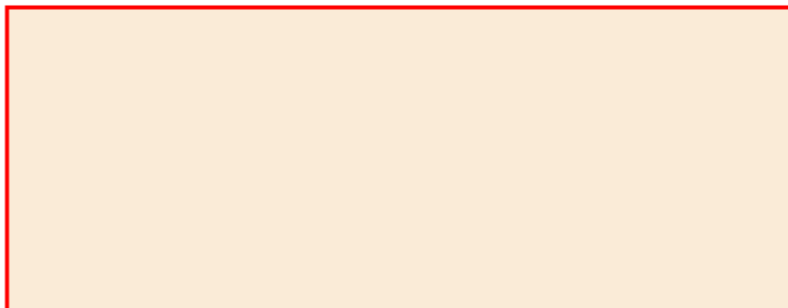
Nota: la función que se ejecuta en **addEventListener** es una callback function. Esto significa que se pasa la función como parámetro, no la llamada a la función. En caso de no tener parámetros sí se puede pasar la llamada. Si se quiere reutilizar funciones se puede llamar a la función en otra función. Veremos un ejemplo más adelante.

17- El Objeto Element

```
<body>
```

```
<form action="login.php" method="get">
  <label for="nombre">Nombre:</label>
  <input type="text" id="nombre" name="nombre" placeholder="su nombre"><br>
  <label for="email">e-mail:</label>
  <input type="email" id="email" name="email"> <br>
  <label for="dia">Día:</label>
  <input type="date" id="dia" name="dia"> <br>
  <label for="hora">Hora:</label>
  <input type="time" id="hora" name="hora"> <br>
  <label for="maletas">Número de maletas:</label>
  <input type="number" id="carga" name="maletas" min="0" max="3" value=0> <br>
  <p id="resumen"></p>  <!-- rectángulo rojo por CSS y vacío -->
</form>
```

Nombre:	<input type="text" value="su nombre"/>
e-mail:	<input type="email"/>
Día:	<input type="date" value="dd/mm/aaaa"/>
Hora:	<input type="time" value="--:--"/>
Número de maletas:	<input type="number" value="0"/>



17- El Objeto Element

```
<script>
function actualizaResumen(){
    if (document.getElementById("nombre").value.length!=0
        && document.getElementById("email").value!=0
        && document.getElementById("dia").value!=0
        && document.getElementById("hora").value!=0
    )
    {
        var nombre = document.getElementById("nombre").value;
        var email = document.getElementById("email").value;
        var dia = document.getElementById("dia").value;
        var hora = document.getElementById("hora").value;
        var cargo = parseInt(document.getElementById("cargo").value)*20;

        document.getElementById("resumen").innerHTML = "Sr. "+nombre+" lea atentamente este
        resumen. Será también enviado a su correo: "+email+". Le recogeremos el día "+dia+" y
        deberá estar cinco munutos antes de las "+hora+". Se le añadira un cargo de "+cargo+"
        por las maletas.";
    }
    document.getElementById("nombre").addEventListener("change", actualizaResumen);
    document.getElementById("email").addEventListener("change", actualizaResumen);
    document.getElementById("dia").addEventListener("change", actualizaResumen);
    document.getElementById("hora").addEventListener("change", actualizaResumen);
    document.getElementById("cargo").addEventListener("change", actualizaResumen);
}
</script>
```

Nombre:

e-mail:

Día:

Hora:

Número de maletas:

Mr. Anonymous

anonymous@notyourbusiness.com

01/08/2022



14:00



1

Sr. Mr. Anonymous lea atentamente este resumen. Será también enviado a su correo: anonymous@notyourbusiness.com. Le recogeremos el día 2022-08-01 y deberá estar cinco munutos antes de las 14:00. Se le añadira un cargo de 20€ por las maletas.



18- El Objeto Style

style

- ✓ El objeto element representa una declaración única de un elemento definido en el HTML.
- ✓ Accediendo a un elemento a través de alguna función del objeto document vamos a poder acceder y modificar su estilo.
- ✓ El estilo podrá ser accedido siempre y cuando sea coherente con cualquier estilo que pudiera aparecer una hoja CSS.
- ✓ La lista de estilos que se puede acceder y modificar es larga. Puede consultarse el documento de referencia de JavaScript

17- El Objeto Element

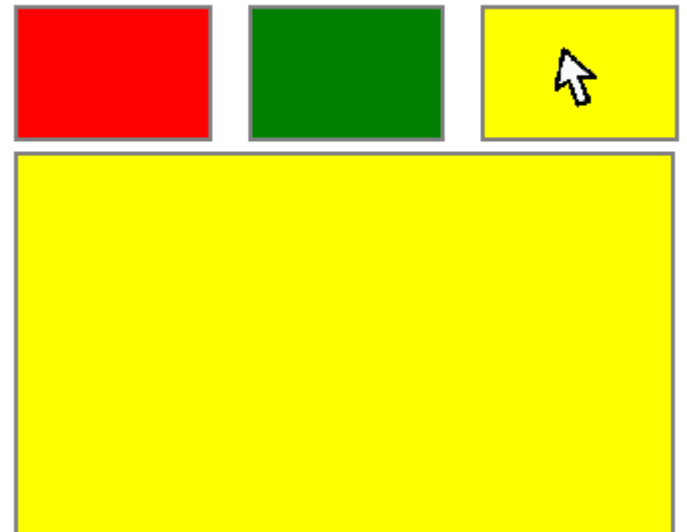
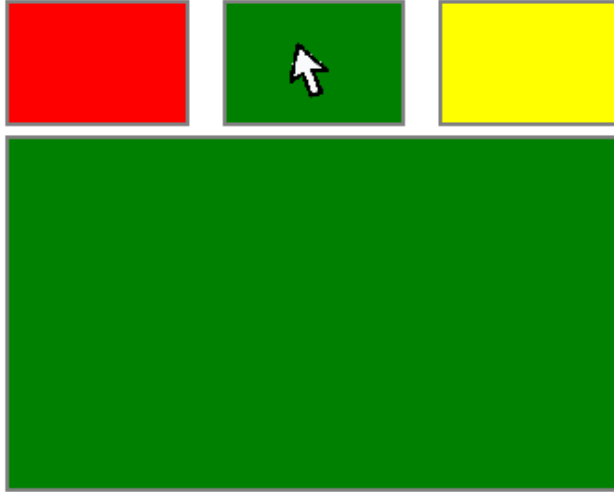
```
<script>
  function cambiaColor(id){
    var color;
    switch (id){
      case 1:
        color="red";
        break;
      case 2:
        color="green";
        break;
      case 3:
        color="yellow";
        break;
      default:
        color="red";
    }
    document.getElementById("divN").style.backgroundColor = color;
  }
  document.getElementById("div1").addEventListener("click", function(event) {
    cambiaColor(1);
  });
  document.getElementById("div2").addEventListener("click", function(event) {
    cambiaColor(2);
  });
  document.getElementById("div3").addEventListener("click", function(event) {
    cambiaColor(3);
  });
</script>
```

```
<style>
  #div1, #div2, #div3 {
    width: 75px;
    height: 50px;
    border: 2px gray solid;
    display: inline-block;
    margin: 0px 5px;
  }
  #div1 { background-color: red;}
  #div2 { background-color: green;}
  #div3 { background-color: yellow;}

  #divN{
    width: 260px;
    height: 150px;
    margin: 0px 5px;
    border: 2px gray solid;
  }
</style>
```

```
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
<div id="divN"></div>
```

17- El Objeto Element





Práctica

- ✓ Realiza la práctica 4.3



19- JS Async

- ✓ JS Async es una librería con un conjunto de herramientas que permiten ejecutar funciones en JS de forma asíncrona.
- ✓ Estas herramientas son:
 - ✓ JS Callbacks
 - ✓ JS Asynchronous
 - ✓ JS Promises
 - ✓ JS Async/Await



19- JS Async

callbacks

- ✓ Una callback de llamada es una función que se pasa como argumento a otra función.
- ✓ Esta técnica permite que una función llame a otra función, pero que podrá variar, se le podrá pasar una u otra función según convenga.
- ✓ Una función callback puede ejecutarse después de que otra función haya finalizado

19- JS Async

callbacks

- ✓ Tenemos dos ejemplos. En el primero hay que llamar a dos funciones para obtener el resultado. En el segundo la primera función puede sólo ejecutar la segunda función y no otra.

```
//Ej1
function mostrar(num) {
    document.writeln(num);
}
function calcular(num1, num2) {
    let sum = num1 + num2;
    return sum;
}
let result = calcular(5, 5);
mostrar(result);
```

```
//Ej2
function mostrar(num) {
    document.writeln(num);
}
function calcular(num1, num2) {
    let sum = num1 + num2;
    mostrar(sum);
}
calcular(5, 5);
```

19- JS Async

callbacks

- ✓ Mediante las callback la función se pasa como parámetro, que será ejecutado por la función que lo recibe.

```
//Ej3. Callback
function mostrar(num) {
    document.writeln(num);
}

function calcular(num1, num2, myCallback) {
    let sum = num1 + num2;
    myCallback(sum);
}
calcular(5, 5, mostrar);
```

19- JS Async

callbacks

- ✓ Las callback son muy útiles al trabajar con funciones asíncronas, donde una función tiene que esperar a que otra función (como esperar a que se cargue un archivo).
- ✓ Las funciones que se ejecutan en paralelo con otras funciones se denominan asincrónicas. Un buen ejemplo es JavaScript `setTimeout()`. En el primer ejemplo se pasa la función como argumento. En el segundo se pasa una función completa que llama a la función.

```
//Ej4. setTimeout con callback
setTimeout(mostrar, 3000);
function mostrar() {
    document.write("Han pasado 3 segundos");
}
```

Han pasado 3 segundos

```
//Ej5. setTimeout con callback con función completa
setTimeout(function() { mostrar("Han pasado 3 segundos"); }, 3000);
function mostrar(value) {
    document.write(value);
}
```

19- JS Async

promises

- ✓ Una promise (promesa) es un objeto que representa la terminación o el fracaso de una operación asíncrona.
- ✓ La sintaxis de una promesa es la siguiente.

```
let miPromesa = new Promise(function(siExito, siFracaso) {  
    // "Código ejecutando" (Puede llevar algún tiempo)  
  
    //Al acabar bajo alguna condición ejecutamos  
    siExito(); // si satisface la condición  
    siFracaso(); //si no la satisface  
});  
  
// "Código esperando" (Debe esperar a que acabe la promesa)  
miPromesa.then(  
    function(value) { /* codigo si tiene éxito */ },  
    function(error) { /* codigo si tiene fracasa */ }  
);
```

- ✓ Promise.then() tiene dos argumentos (éxito y error). Son opcionales, se podría poner sólo uno.

19- JS Async

promises

- ✓ Si continuamos con el ejemplo que veníamos utilizando.

```
//Ej6
function mostrar(mensaje) {
    document.writeln(mensaje);
}
function mostrarError(mensaje){
    alert(mensaje);
}

let promesa = new Promise(function(siExito, siFracaso) {
    let x = 0;
    // "Código ejecutando" (Puede llevar algún tiempo)
    if (x == 0) {
        siExito("OK");
    } else {
        siFracaso("Error");
    }
});

promesa.then(
    function(mensaje) {mostrar(mensaje);},
    function(error) {mostrarError(error);}
);
```

- ✓ Si no queremos utilizar function en el .then es posible utilizar “=>”

```
promesa.then(
    (mensaje) =>{mostrar(mensaje);},
    (error)   =>{mostrarError(error);}
);
```


19- JS Async

async y await

- ✓ Las palabras clave async y await hacen mucho más fácil trabajar con promesas.
- ✓ La palabra clave async antes de una función hace que la función devuelva una promesa:

```
async function miFuncion() {  
    return "Hola";  
}  
  
//Es lo mismo que  
function miFuncion() {  
    return Promise.resolve("Hola");  
}
```

- ✓ La palabra clave await hace a una función esperar una promesa. Sólo puede ser utilizada dentro de una función async.

```
let valor = await promesa;
```

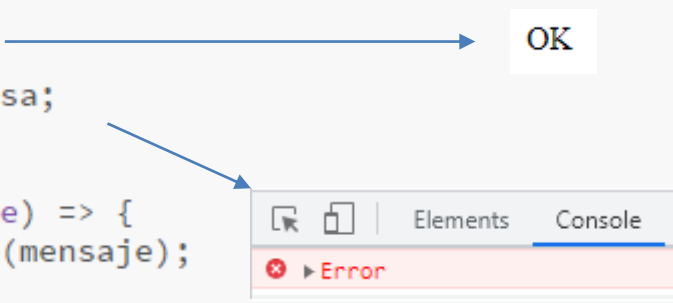
19- JS Async

async y await

- ✓ Siguiendo con nuestro ejemplo async hace que el resultado de mostrar pueda recogerse con **then** y **catch**.

```
//Ej7
async function mostrar() {
  let promesa = new Promise(function(siExito, siFracaso) {
    let x = 0;
    // "Código ejecutando" (Puede llevar algún tiempo)
    if (x == 0) {
      siExito("OK");
    } else {
      siFracaso("Error");
    }
  });
  return await promesa;
}

mostrar().then((mensaje) => {
  document.write(mensaje);
})
.catch((error) => {
  console.error(error);
});
```



- ✓ Nota: en este ejemplo estamos creando la promesa nosotros mismos. Lo habitual, y lo útil al trabajar con async y await es crear funciones que utilicen promesas de librerías, por ejemplo las que esperan a que conteste un servidor.

19- JS Async

async y await

- ✓ En muchos casos la función de rechazo no se necesita.

```
//Ej8
async function mostrar() {
  let promesa = new Promise(function(siExito) {
    let x = 0;
    // "Código ejecutando" (Puede llevar algún tiempo)
    if (x == 0) {
      siExito("OK");
    }
  });
  return await promesa;
}

mostrar().then((mensaje) => {
  document.write(mensaje);
});
```



19- JS Async

async y await

- ✓ Por último si utilizamos setTimeout como una promesa con async y await:

```
//Ej9. Promesa con un setTimeout
async function mostrar() {
    let promesa = new Promise(function(siExito) {
        setTimeout(function(){siExito("Han pasado 3 segundos");}, 3000);
    });
    return await promesa;
}
mostrar().then((mensaje) => {
    document.write(mensaje);
});
```

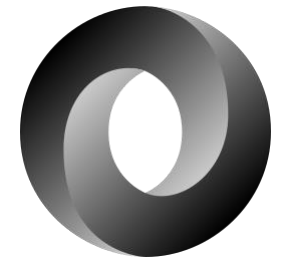
Han pasado 3 segundos

A decorative vertical bar on the left side of the slide, composed of numerous overlapping circles of various sizes and colors, including blue, green, yellow, orange, and pink, creating a vibrant, bubbly effect.

Práctica

- ✓ Realiza la práctica 4.4

20- JSON



- ✓ JSON (JavaScript Object Notation) un formato de texto para almacenar y transportar datos, “autodescriptivo” y fácil de entender.
- ✓ En JSON los datos se engloban entre llaves ({...}), se presentan como pares clave/valor donde la clave va entre comillas dobles y los valores deben ser uno de los siguientes tipos de datos:
 - ✓ un string
 - ✓ un número
 - ✓ un objeto
 - ✓ un array
 - ✓ un booleano
 - ✓ nulo

```
{  
  "code": "I7HF",  
  "name": "John",  
  "children": ["GB32", "H2SS", "A22D"],  
  "age": 30,  
  "car": null,  
  "address": {"country": "USA", "city": "New York", "street": "45", "number": "12"}  
}
```

20- JSON

- ✓ Un uso común de JSON es intercambiar datos desde un servidor web.
- ✓ Al recibir datos de un servidor web, los datos siempre son una cadena.
- ✓ Los datos se analizan y se convierten en un objeto de JavaScript con **JSON.parse()**.

```
const obj = JSON.parse('{ "code": "I7HF", "name": "John", "children":  
["GB32", "H2SS", "A22D"], "age": 30, "car": null, "address":  
{ "country": "USA", "city": "New York", "street": "45",  
"number": "12" } }');
```

```
document.writeln(obj.name);  
document.writeln(obj.children[2]);  
document.writeln(obj.address.country);
```

John A22D USA

20- JSON

- ✓ Otro uso común de JSON es intercambiar datos hacia un servidor web.
- ✓ Al enviar datos a un servidor web, los datos deben ser una cadena.
- ✓ Los datos se convierten en una cadena JSON con **JSON.stringify()**.

```
const person = {  
  code:"I7HF",  
  name:"John",  
  children:["GB32", "H2SS", "A22D"],  
  age:30,  
  car:null,  
  address:{country:"USA", city:"New York", street:"45", number:"12"}  
};  
  
document.writeln(JSON.stringify(person));  
window.location = "deliver_json.php?x=" + myJSON; //window.location es la URL actual  
                                                    //Al cambiarla, carga la nueva página php.
```

```
{"code":"I7HF","name":"John","children":["GB32","H2SS","A22D"],"age":30,"car":null,"address":  
{"country":"USA","city":"New York","street":"45","number":"12"}}
```



21- API Webs con JSON

- ✓ Existen diferentes herramientas de JavaScript que nos permiten trabajar con APIs Web:
 - ✓ Web Forms API: para validar formularios.
 - ✓ Web History API: para acceder al historial del navegador.
 - ✓ Web Storage API: para almacenar datos en el navegador.
 - ✓ Web Worker API: para ejecutar código en segundo plano.
 - ✓ Web Fetch API: para hacer peticiones HTTP.
 - ✓ Web Geolocation API: para acceder a la geolocalización del usuario.
- ✓ De entre ellas Web Fetch API es la que nos permite realizar una petición HTTP y descargar datos ej JSON que luego pueden ser interpretados por el navegador.

21- API Webs con JSON

- ✓ Las peticiones HTTP van a ser consumir un tiempo en ser respondidas. Esto hace que sean asíncronas y como se puede suponer que lo más lógico sea utilizar una callback, o en este caso una promesa.
- ✓ Para ello se utilizan dos funciones asíncronas que devuelven una promesa:
 - ✓ **fetch()**: se le pasa un recurso (normalmente una URL) y devuelve un objeto respuesta de consultar el recurso (**Response**).
 - ✓ **Response**, entre otros tiene las siguientes propiedades y métodos:
 - ✓ **Response.ok**: (true/false), que indica si se ha obtenido respuesta del servidor.
 - ✓ **Response.status**: que devuelve el estado de la respuesta HTTP del servidor(200-ok, 404-Not found, etc)
 - ✓ **Response.text()**: es un método del objeto Response que devuelve una promesa con un string de la respuesta.

```
//Utilización de fetch
fetch(file)
  .then(x => x.text())
  .then(y => document.write(y));

//Utilización de fetch creando una función
async function extraeTextoDeWeb(file) {
  let miResponse = await fetch(file);
  if (miResponse.ok && miResponse.status=="200")
  {
    let miTexto = await miResponse.text();
    return miTexto;
  }
}
extraeTextoDeWeb("http://unaur1.es").then((message)=>{document.write(message)});
```

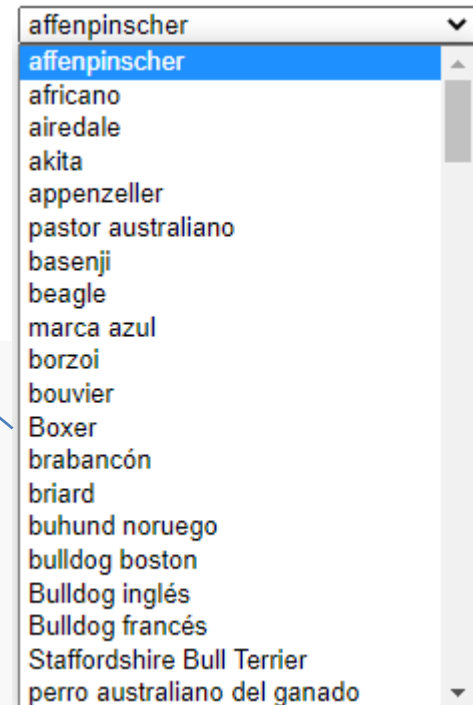
- ✓ Nota: antes de la API Web Fetch se utilizaba la función XMLHttpRequest.

21- API Webs con JSON

- ✓ Un ejemplo real de utilización. Mejor verlo en funcionamiento.
- ✓ <https://dog.ceo/dog-api/> es una api web que devuelve imágenes de perros:
 - ✓ bien random: (<https://dog.ceo/api/breeds/image/random>).
 - ✓ Bien especificando la raza
(https://dog.ceo/api/breed/text_raza/images/random)
- ✓ La forma de hacerlo es devolviendo un objeto JSON con la URL de la imagen, y si la búsqueda ha tenido éxito (no contiene todas la razas).

JSON	Datos sin procesar	Cabeceras
Guardar	Copiar	Adaptar para impresión
<pre>{"message": "https:\\\\images.dog.ceo\\breeds\\basenji\\n02110806_487.jpg", "status": "success"}</pre>		

21- API Webs con JSON



```
<body>
  <select id="selector-perro"> ... </select>
  <img id="imagen_perro">

  <script>
    let direccion = "";
    let objeto_JSON = null;
    async function buscaAPI(file) {
      let respuesta_FETCH = await fetch(file);
      let respuesta_texto = await respuesta_FETCH.text();
      return respuesta_texto;
    }
    function seleccionaPerro(){
      direccion="https://dog.ceo/api/breed/"+document.getElementById("selector-perro").value+"/images/random";
      buscaAPI(direccion).then((mensaje)=>
        {
          objeto_JSON = JSON.parse(mensaje);
          if (objeto_JSON.status!="error")
            document.getElementById("imagen_perro").src=objeto_JSON.message;
          else
            document.getElementById("imagen_perro").src="https://i0.wp.com/learn.onemonth.com/wp-content/uploads/2017/08/1-10.png";
        });
    }
    document.getElementById("selector-perro").addEventListener("change", seleccionaPerro)
  </script>
</body>
```

A decorative vertical bar on the left side of the slide, composed of numerous overlapping circles of various sizes and colors, including blue, green, yellow, orange, and pink, creating a vibrant, bubbly effect.

Práctica

- ✓ Realiza la práctica 4.5

22.- Fuentes

✓ <https://www.w3schools.com/>



✓ <https://developer.mozilla.org>

