# JAVASCRIPT.

Carcterísticas de **Javascript**:

- Se ejecuta en local.

- Es interpretado, no compilado.

- Es de respuesta inmediata.

- Agrega interactividad a los sitios web.

- Proporciona efectos visuales dinámicos.

¿Dónde va **Javascript**?.

Para “linkar” una página **javascript** a un **html**, se puede poner en el **head**, en cualquier etiqueta o “linkar” la página entera (como **CSS**);

En el head: Entre etiquetas <script></script>.

<head>

<script>

alert("Hola mundo ");

</script>

</head>

Nota: Si nuestra página va a tener en el head tanto **CSS** como **Javascript**, lo habitual es poner primero el código **CSS**, ya que es muy frecuente que el **script** haga referencia a elementos **CSS**.

En una etiqueta:

<p onClick="alert('Hola mundo');">Página de prueba Javascript</p>

También podemos sacar contenido en la consola del navegador con:

**console.log**("Esto se muestra en la consola");

Linkar una página Javascript a nuestro **HTML**:

Creo la página **Javascript** y en el **head** de nuestra página **HTML** pongo:

<script src="nombre de la página Javascript.js"></script>

## ESTRUCTURAS BÁSICAS.

**Sentencias**: Llamamos sentencia a cada “orden” que programemos en **Javascript**. Terminan con “**;**”.

alert("Esto es una sentencia ")**;**

document.write("Esta es otra ")**;**

**Funciones predefinidas**: Comandos que ya vienen por defecto en **Javascript** listos para ser utilizados. Siempre llevan paréntesis.

Además de estas funciones predefinidas, nosotros podremos crear las nuestras .

alert();

document.write();

**Comentarios**: En **Javascript** los comentarios son como en **Java**.

**//** Comentario de una sola línea.

**/\***…**\*/**  Comentario de varias líneas.

**Tipos de datos: Javascript** tiene solo tres tipos básicos:

* Valores numéricos.
* Strings.
* Booleans.

**Comandos de consola**: Podemos ejecutar comandos en la consola:

**console.log**("Escribo en la consola.");

## MODO ESTRICTO.

En javascript tenemos un modo que al activarlo nos permite ser más restrictivo en nuestro modo de programar, para ello, al principio de nuestra página pondremos:

‘use strict’

Por ejemplo, sin este modo activo yo podría poner:

num=5;

Con el modo estricto activo tendría que poner:

var num=5;

Además, existen determinadas funcionalidades que sólo funcionan con el modo estricto activado.

## DECLARACIÓN DE VARIABLES.

Se pueden declarar las variables de dos formas, con **let** y con **var**:

**let** permite declarar variables limitando su alcance al bloque o declaración donde se está usando.

**var** define una variable global o local (en una función) sin importar el ámbito del bloque.

Ejemplo:

var num=5;

document.write(num); //num vale 5

if(true){

var num = 10;

document.write(num); //num vale 10

}

document.write(num); //num vale 10

Con **let**:

let num=5;

document.write(num); //num vale 5

if(true){

let num = 10;

document.write(num); //num vale 10

}

document.write(num); //num vale 5

**let** lo que realmente hace es crearme una nueva variable local a nivel de bloque, en este caso dentro del **if**. Fuera no tiene validez.

var puntuación;

puntuación=500;

var puntuación=500, record=1000, jugador= "Angel";

Si declaramos una variable entre comillas, se entiende que es un string:

var num1=5; Aquí num1 es un número (int).

var num2= "5"; Aquí num2 es un string.

Para modificar una variable ya declarada no ponemos var, simplemente el nombre y el valor:

var num=5; //Declaracion y asignación

num=10; //Reasignación

Con prompt introducimos la variable mediante una ventana.

var nombre=prompt("Introduce nombre " , "nombre por defecto");

**TODO** lo introducido con promt se considera un String.

Para comprobarlo: alert(**typeof**(num1));

Con typeof() me indica el tipo de una variable.

Podemos poner typeof num; o typeof(num);

Si quiero convertir un string a número tengo dos opciones:

1-Usar la función **Number**():

Var num1=45;

Resultado= num1+**Number**(num2);

2-Usar **parseInt**:

alert(**parseInt**(num2)+23);

De igual forma tenemos disponible la función **parseFloat().**

Con Document.write podemos escribir directamente en el código **HTML** de nuestra página.

Se devuelve un String si el usuario hace clic en "OK ", se devuelve el valor de entrada. Si el usuario hace clic en "cancelar ", se devuelve NULL. Si el usuario hace clic en aceptar sin introducir ningún texto, se devuelve una cadena vacía.

<head>

<meta charset="utf-8">

<title>Documento sin título</title>

<script >

var nombre = "Angel";

var apellido = "García";

document.write("<h1>Tu nombre es</h1>"+nombre+" "+apellido);

</script>

</head>

Podíamos pedir ambos parámetros por ventana:

var nombre=**prompt**("Introduce nombre");

var apellido=**prompt**("Introduce apellido");

### CONSTANTES.

Sintaxis:

**const** nombre=”Angel”;

## STRINGS.

Funciones de los **strings**:

**toString**: Para convertir un número a **string**:

var num=3;

var num2=num.**toString**();

**toUppercase**: Convierte a mayúsculas.

**toLowercase**: Convierte a minúsculas.

**Length**: Calcula la longitud de un texto. Si la usamos con **arrays** me dice el número de elementos del **array**.

**Concat**: Para concatenar texto. Como el +

var texto=texto1.concat(“ ”+texto2);

**indexOF**(): Para buscar un texto dentro de otro. Me saca su existe un texto dentro de otro y en qué posición está. Busca la primera coincidencia.

También tenemos un **LastIndexOf**() que nos saca la última coincidencia.

<script>

var texto="Esto es el texto";

var busqueda= texto.indexOf("texto");

document.write(busqueda);

</script>

En este caso me imprimiría 11.

El método **search**() funciona igual:

var busqueda= texto.search("texto");

Si no encuentra el texto me devolverá -1.

**Match**(): Funciona como **indexOF** o **search**, pero en este caso nos devuelve un array con los resultados. En principio me devuelve la primera coincidencia. Si quiero que me saque todas tengo que usar la expresión regular:

var busqueda= texto.match(/texto/gi);

**Substr**(): Sácame desde el carácter 14, 5 caracteres en adelante:

var busqueda= texto.substr(14,5);

**CharAt**(): Para seleccionar una letra determinada de un string.

var busqueda= texto.CharAt(44);

Me mostraría la letra que se encuentre en la posición 44.

**StartWith():** Para buscar un texto al inicio del string. Devuelve **true** o **false**, dependiendo de si lo ha encontrado o no. Es decir, si el texto comienza por los caracteres que le digamos. Ejemplo:

var texto=”Hola mundo”;

var busqueda= texto.**StarWith**(“Hola”);

En este caso daría true.

var busqueda= texto.**StarWith**(“mundo”);

En este caso daría false.

**endsWith**(); Como el anterior pero mirando si el texto termina con los caracteres que le digamos.

**includes**(); Busca una palabra en un texto. Es case sensitive.

**replace**(); Nos permite reemplazar un texto por otro.

var busqueda= texto.**replace**(“Hola”, “Adios”);

Me busca “hola” y me lo reemplaza por “Adios”.

slice(); Me separa un string a partir del carácter que le digamos.

var busqueda= texto.**slice**(5);

Me corta el string a partir del carácter 5, es decir, me deja sólo “mundo”.

También puedo decirle posición de comienzo y fin:

var busqueda= texto.**slice**(5,7);

**split**(); Me mete el string en un array.

var busqueda= texto.**split**();

Podemos indicarle un separador y nos mete las palabras que hay entre espacios, por ejemplo:

var busqueda= texto.**split**(“ ”);

**trim**(): Me quita los espacios por delante y por detrás del string.

### PLANTILLAS EN STRINGS.

Podemos sustituir valores dentro un string sin necesidad de estar concatenándolos, usando una plantilla. Para ello uso las comillas invertidas y la interpolación de variables con $(). Ejemplo:

Modo normal:

var nombre=prompt(“Introduce tu nombre: ”);

var apellidos=prompt(“Introduce tus apellidos: ”);

var texto=“Tu nombre es ”+nombre+” y tus apellidos”+apellidos;

Con plantilla:

var texto= **`**

<h3>Tu nombre es **$(**nombre**)**</h3>

<h3> y tus apellidos **$(**apellidos**)**</h3>

**`**;

## ALERTAS.

**Alerta normal:**

<script>

alert("Esto es una alerta");

</script>

**Alertas de confirmación:**

<script>

var confirmacion = confirm("Estas seguro de continuar?");

</script>

En la variable confirmación almaceno la contestación del usuario que será **true** o **false**.

**Ventana de ingreso de información:**

Para ello utilizamos la función **prompt()**, a la que puedo pasarle dos parámetros. El primer parámetro es el texto a mostrar y el segundo el valor por defecto.

<script>

var edad = **prompt**("Que edad tienes?: ", 18);

</script>

Lo que se introduzca por **prompt()** siempre es un **string**. Si quiero números tengo que hacer la conversión con **parseint**, **parsefloat**…etc.

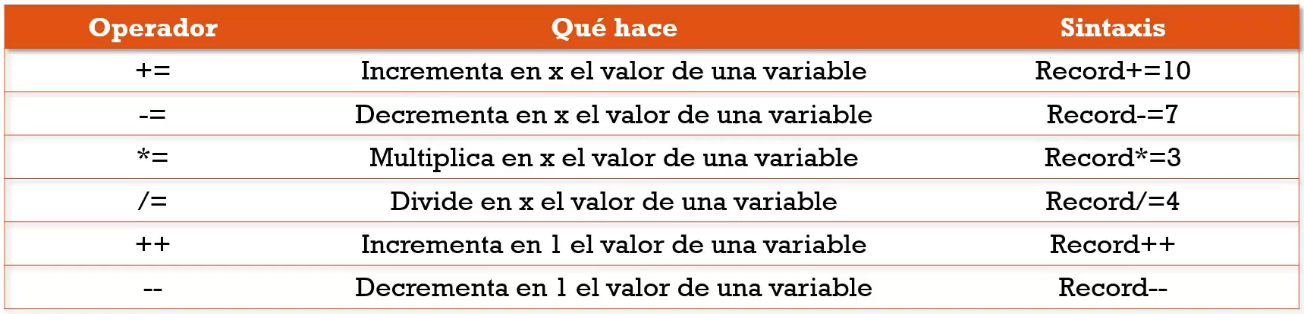
## OPERADORES.

### OPERADORES BÁSICOS.



Falta el operador resto de la división: %

### OPERADORES INCREMENTO Y DECREMENTO.



Ej:

<script >

var edad=prompt("Introduce edad");

var restar=prompt("Cuanto años quieres quitarte?");

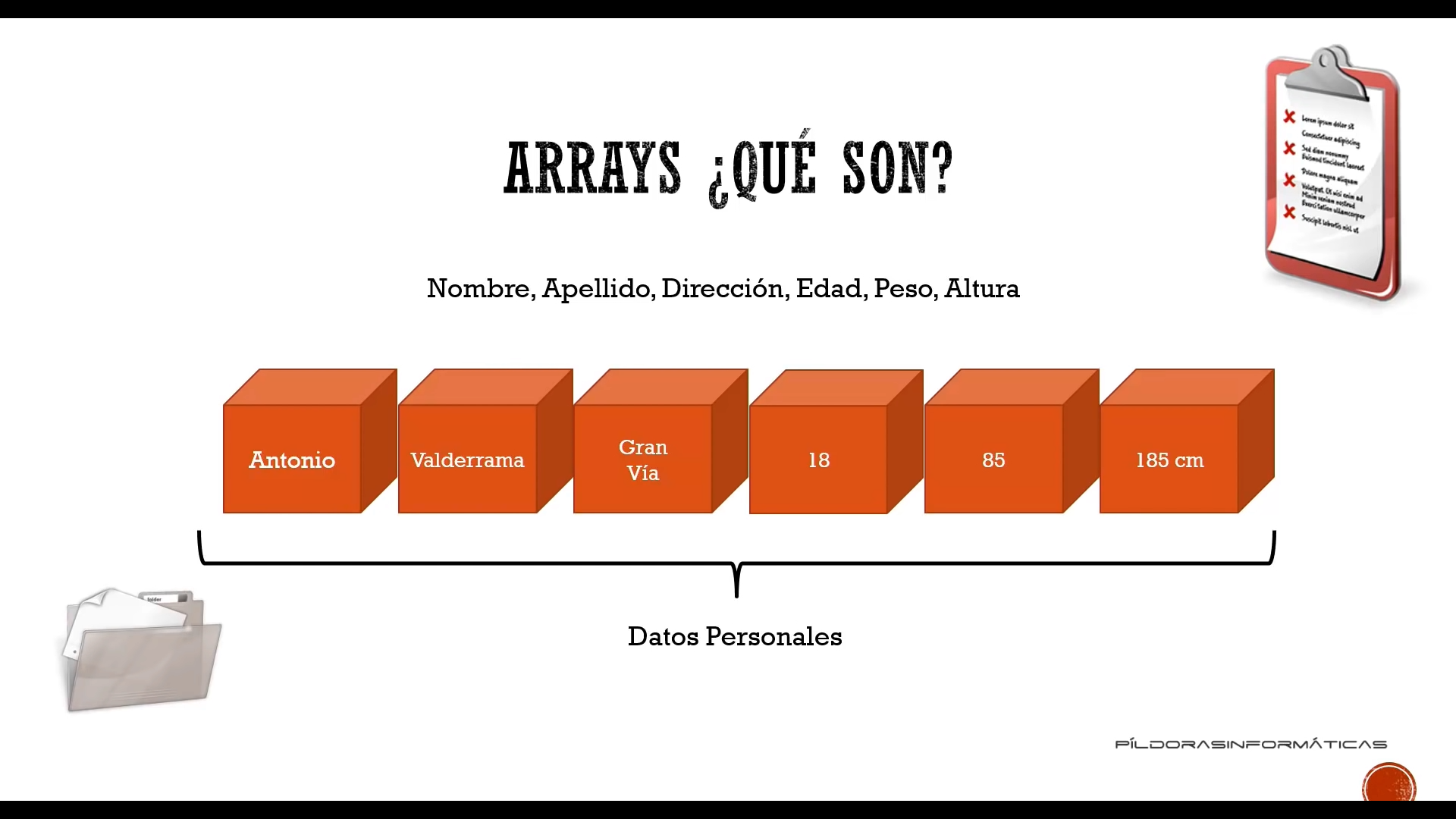
edad-=restar;

document.write("<h1>Edad final: </h1>"+"<br>"+edad);

</script>

## ARRAYS.

En **Javascript** los valores del **array** NO tienen que ser del mismo tipo.



En **Javascript** un **array** se declara de las siguientes formas:

var Articulos=["zapatilla","camiseta","pantalon","calcetines"];

Ó:

Var Articulos =new array("zapatilla","camiseta","pantalon","calcetines");

Ejemplo de acceso a un array:

alert(Articulos[1]); Resultado: camiseta

La posición de los elementos empieza desde cero, como el Java.

<script >

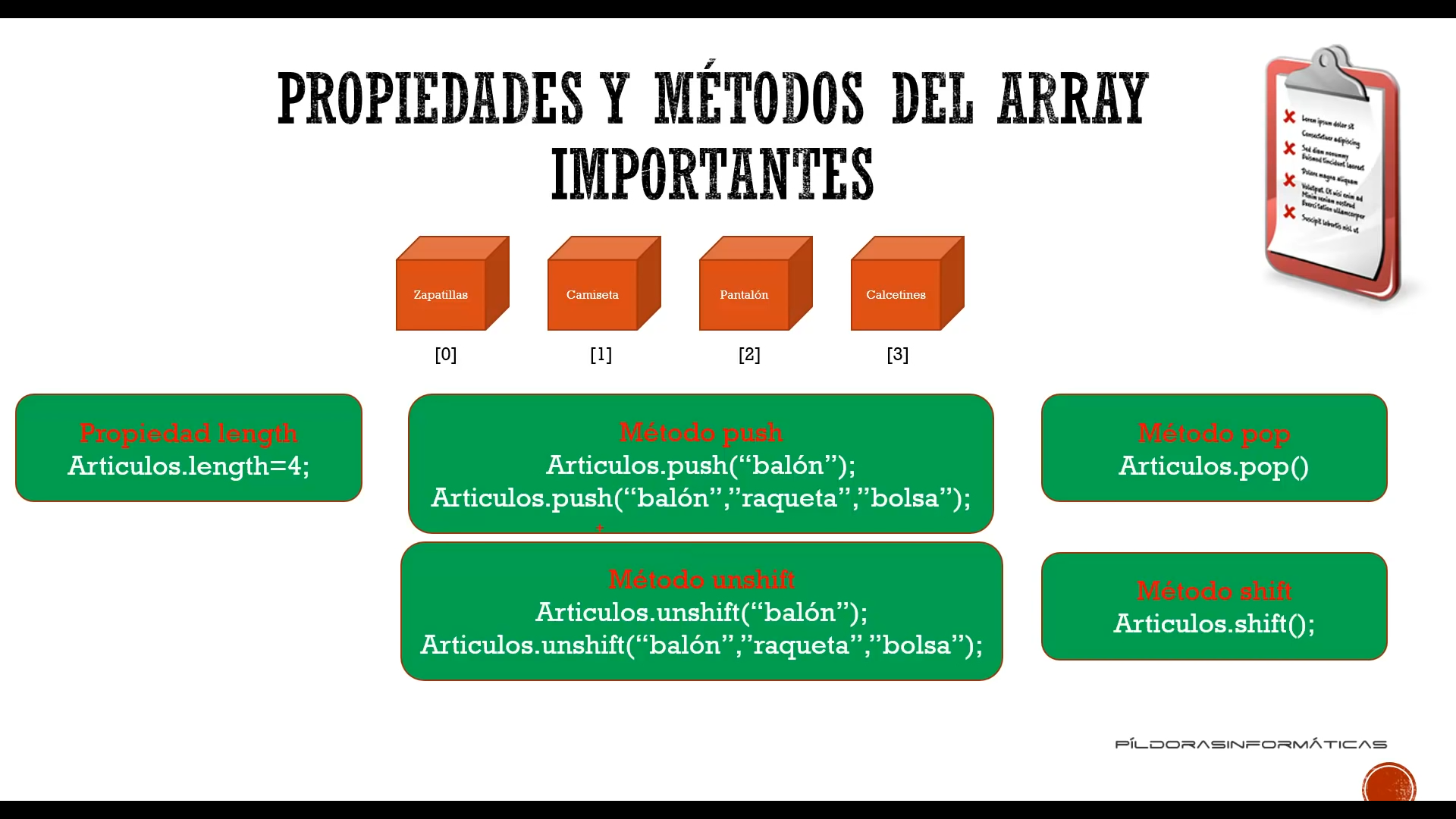
var array1=["objeto1","objeto2","objeto3"];

array1[3]="objeto4";

array1[4]=prompt("Introduce objeto4");

alert(array1[4]);

</script>



**Length**: Devuelve la longitud del array.

**push**: Agrega elementos al final del array.

**unshift**: Agrega elementos al principio del array.

**pop**: Elimina elementos al final del array.

**shift**: Elimina elementos al principio del array.

Ej:

<html>

<head>

<meta charset="utf-8">

<title>Documento sin título</title>

<script >

var articulos=["balon","botas","camiseta","pantalon"];

</script>

</head>

<body>

<script>

document.write("<p>El primer articulo es

<strong>"+articulos[0]+"</strong></p>");

document.write("<P>La longitud del array es "+articulos.length+"</p>");

document.write("<p>El último articulo es <strong>"+articulos[articulos.length-1]+"</strong></p>");

articulos.unshift("pelota");

articulos.push("canasta");

document.write("<p>Añadimos el artículo <strong>"+articulos[0]+"</strong></p>");

document.write("<p>El primer articulo ahora es <strong>"+articulos[0]+"</strong></p>");

document.write("<p>El último articulo ahora es <strong>"+articulos[articulos.length-1]+"</strong></p>");

document.write("<P>La longitud del array ahora es "+articulos.length+"</p>");

document.write(articulos);

articulos.push(prompt("introduce nuevo articulo"));

document.write("<br>");

document.write(articulos);

</script></body></html>

Podemos poner un elemento como **undefined**;

articulos[0]= **undefined**;

Para averiguar la posición de un elemento:

var posicion=articulos.**indexof**(“pantalones”);

Convertir un array a texto:

var miString=artículos.**join**();

Me lo convierte a un **string** separado por comas.

Para hacer lo contrario, es decir convertir un **string** en **array** lo haremos con el método **split**(). Dentro de los parámetros pondremos el carácter que hará de separador.

var cadena=”nombre1,nombre2,nombre3”;

var miArray=cadena.split(“,”);

Para ordenar un array usaremos el método el método **short**(). Para ordenar por orden inverso el método **reverse**();

cadena.short();

cadena.reverse();

Esto nos ordena un array alfabéticamente, aunque sea un array numérico.

Para ordenar un array numérico de forma correcta:

var numeros=[3,43,34,32,345,6];

numeros.short(function(a,b){return a-b});

Y en orden inverso de mayor a menor:

numeros.short(function(a,b){return b-a});

### RECORRER ARRAYS CON FOR.

<script>

document.write("<ul>");

var nombres=["Angel", "Sara", "Manolo", "Ana"];

for(var i=0; i<nombres.length; i++){

document.write("<li>"+nombres[i]+"</li>");

};

document.write("</ul>");

</script>

### RECORRER ARRAYS CON FOR EACH.

Utilizamos la función **foreach** que va a recibir el elemento por parámetro.

<script>

document.write("<ul>");

var nombres=["Angel", "Sara", "Manolo", "Ana"];

nombres.forEach((elemento)=>{

document.write("<li>"+elemento+"</li>");

});

document.write("</ul>");

</script>

Opcionalmente la función **foreach** puede recibir más parámetros.

Un segundo parámetro puede ser el índice.

nombres.forEach((elemento, index)=>{

document.write("<li>"+index+" – "+elemento+"</li>");

### RECORRER ARRAYS CON FOR IN

Otra forma de recorrer un array es con el **for in.**

for( let nombre **in** nombres){

document.write("<li>"+nombres[nombre]+"</li>");

};

En este caso nombre es el índice.

### BÚSQUEDAS EN UN ARRAY.

Para ello podemos usar el método **find**() que tiene dentro una función de carga.

var busqueda=nombres.**find**(function(nombre){

return nombre==nombre1;

});

busqueda almacena el texto que estamos buscando.

Esto se puede simplificar poniéndolo de la siguiente forma:

var busqueda=nombres.**find**(nombre => nombre==nombre1;);

De forma similar podemos buscar el índice con el método **findIndex**().

var busqueda=nombres.**findIndex**(nombre => nombre==nombre1;);

Otro método interesante es **some**() que nos permite comprobar si alguno de los elementos del array cumple una determinada condición. Por ejemplo, comprobar si hay elementos mayores de 40:

var numeros=[3,45,33,65,776];

var busqueda=numeros.some(numero=>numero>40);

Almacena **true** o **false.**

### ARRAYS MULTIDIMENSIONALES

No son más que un array compuesto de otros arrays.

<script>

var peliculas=['Batman', 'Cars', 'Shameless'];

var categoria=['accion', 'infantil', 'drama'];

var cine=[categoria, peliculas];

console.log(cine);

</script>

Para acceder a un elemento, por ejemplo, categoría infantil:

Console.log(cine[0][1]);

### POO.

En Javascript los objetos tienen propiedades y métodos.

Las propiedades se modifican con la nomenclatura del punto y el valor entre comillas:

*nombreDelObjeto.propiedad=* *"valor* *";*

Renault.ancho= "2000 ";

Botton.style.width= "500px ";

Document.write();

Windows.alert();

Para llamar a los métodos: *nombreDelObjeto.metodo();*

Renault.acelera();

<body>

<input type="button" id="boton">

<input type="button" id="boton2">

<script>

var miBoton=document.getElementById("boton");

miBoton.style.width="300px";

miBoton.style.height="300px";

var miBoton2=document.getElementById("boton2");

miBoton2.style.width="300px";

miBoton2.style.height="300px";

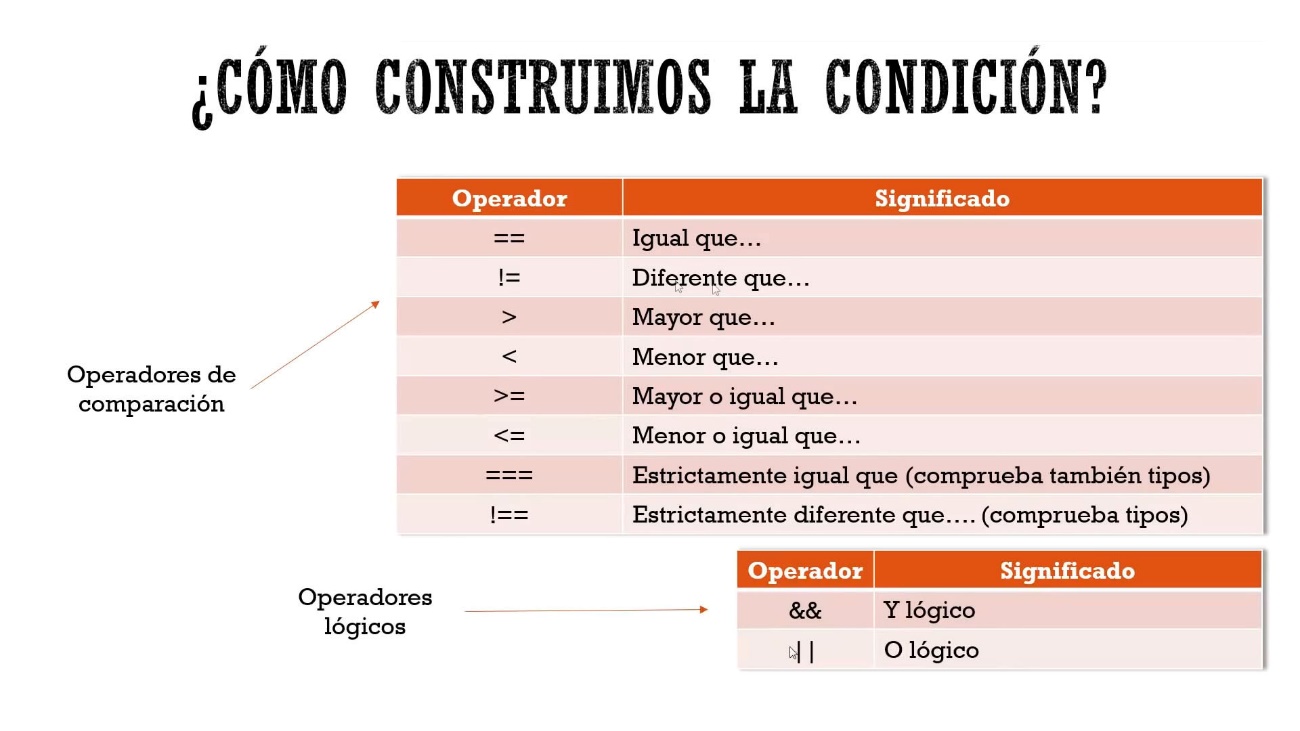
miBoton2.focus();

</script>

</body>

## **CONDICIONALES**.

### COMPARADORES



### IF…ELSE.

Exactamente igual que en Java.

If(…){ if(…){ if(…){

}else{…} {else if{…} {else if{…

{else{…}

Ej:

<body>

<script>

var precio=30000;

var dinero=prompt("Introduce cuanto dinero tienes: ");

if(dinero>precio){

alert("Te puedes comprar el coche");

}else{

alert("Te vas en autobus");

}

</script>

</body>

Ej2 :

<body>

<script>

var precio=30000;

var dinero=prompt("Introduce cuánto dinero tienes: ");

var edad=prompt("Introduce tu edad: ");

if((precio<dinero)&&(edad>=18)){

alert("Te puedes comprar el coche");

}else{

if((precio<dinero)&&(edad<18)){

alert("Tienes el dinero pero no la edad");

}

else if((precio>dinero)&&(edad>=18)){

alert("Tienes la edad pero no el dinero");

}

else if((precio>dinero)&&(edad<18)){

alert("Ni dinero ni edad");

}}

</script>

</body>

Si el usuario me introdujese por ventana un valor que no es un número tendríamos un problema. Para averiguar si un valor es numérico usamos la función isNaN().

<script>

var num1=prompt("Introduce numero");

var num2=prompt("Introduce numero2");

if(!isNaN(num1)&&!isNaN(num2)){

alert(parseInt(num1)+parseInt(num2));

}else{

alert("No has introducido números");

}

</script>

### SWITCH…CASE

var edad=18;

var imprime="";

**switch**(edad){

**case** 18:

imprime="Acabas de cumplir la mayoría de edad";

**break**;

**case** 25:

imprime="Eres u adulto";

**break**;

**case** 50:

imprime="Eres maduro";

**break**;

**default**:

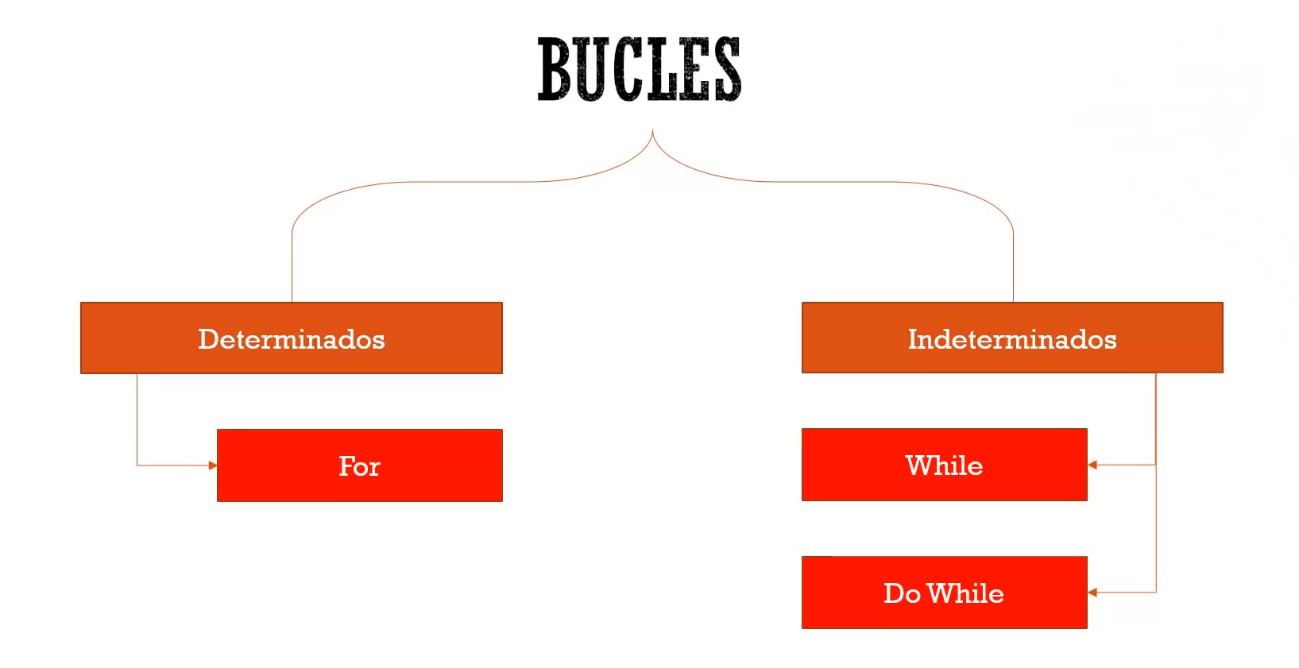
imprime="Otra edad no contemplada"

document.write(imprime);

**break**;

}

## BUCLES.



### WHILE. DO…WHILE.

Exactamente igual que Java. La diferencia entre ambos bucles es que con Do…while nos aseguramos de que el código se ejecuta al menos una vez.

While(…){} do{…}

While(…)

Ejemplo de while:

<script>

var contador=0;

var meses=["Enero","Febrero","Marzo","Abril","Mayo","Junio",

"Julio","Agosto","Septiembre","Octubre","Novienbre","Diciembre"];

while(contador<meses.length){

document.write(meses[contador]+"<br>");

contador++;

}

</script>

Ejemplo de do…while;

<script>

var numero=parseInt(Math.random()\*100);

var numero\_introducido;

var contador=0;

do{

numero\_introducido=prompt("Introduce número: ");

contador++;

if(numero\_introducido>numero){

alert("Demasiado alto");

}

if(numero\_introducido<numero){

alert("Demasiado bajo");

}

}while(numero!=numero\_introducido);

alert("CORRECTO¡¡¡, el número era el "+numero+". Has acertado en "+contador+" intentos.");

</script>

### FOR.

Igual que en Java. Sintaxis:

for(inicio;condición;incremento/decremento){

código a repetir}

continuación del programa

Ej:

<script>

for(var i=0;i<10;i++){

document.write("Hola"+"<br>");

}

</script>

El principal uso del bucle for es recorrer arrays:

<script>

var meses=["Enero","Febrero","Marzo","Abril","Mayo","Junio",

"Julio","Agosto","Septiembre","Octubre","Novienbre","Diciembre"];

for(var i=0;i< meses.length;i++){

document.write(meses[i]+"<br/>");

}

</script>

Ejemplo, rellenar un array y recorrerlo después:

<script>

var trabajadores=new Array();

var contador=0;

var persona;

while(persona!="salir"){

persona=prompt("Introduce nombre");

trabajadores[contador]=persona;

contador++;

}

trabajadores.pop();

for(var i=0;i<trabajadores.length;i++){

document.write(trabajadores[i]+"<br>");

}

</script>

### BREAK.

Nos permite salir de un bucle de ejecución al cumplirse una condición:

var year=2000;

while(year<2010){

document.write(year);

if(year==2005){

break;

year++

}

}

En este caso cuando el año sea igual a 2005, se sale del bucle.

## FUNCIONES.

Conjunto de instrucciones preparadas para ser llamadas en cualquier momento. Sólo se ejecutará si la llamamos. Es algo similar a los métodos de Java.

Sintaxis:

**function** nombre\_funcion(){

//Código a ejecutar

}

//En otro punto de la página **HTML** llamamos a la función

Nombre\_funcion();

Se le pueden pasar parámetros:

**function** suma(**num1**,**num2**){

var resultado=num1+num2;

}

.

.

.

Suma(3,7);

Ejemplo 1:

<head>

<meta charset="utf-8">

<title>Documento sin título</title>

<script>

function fecha(){

document.write(Date());

}

</script>

</head>

<body>

<h1>Hola es día: <script>fecha();</script></h1>

</body>

Ejemplo 2:

<head>

<meta charset="utf-8">

<title>Documento sin título</title>

<script>

var nombre;

function pedir\_nombre(){

nombre=prompt("Introduce nombre");

document.write(nombre);

}

</script>

</head>

<body>

<p>Hola <script>pedir\_nombre();</script> , cómo estás?</p>

</body>

Ejemplo 3:

<head>

<meta charset="utf-8">

<title>Documento sin título</title>

<script>

var num1;

var num2;

function suma(num1,num2){

num1=prompt("Introduce número 1: ");

num2=prompt("Introduce número 1 :");

document.write("La suma de "+num1+" y "+num2 +" es= "+(Number(num1)+Number(num2)));

}

</script>

</head>

<body>

<script>

suma(num1,num2);

</script>

</body>

Opcionalmente se puede pasar un tercer parámetro, opcional, que ya viene inicializado por defecto.

<script>

function suma(num1, num2, **mostrar=true**){

if(mostrar==true){

return (num1+num2));

}}

document.write(suma(2,3));

</script>

### PARAMETROS REST Y SPREAD.

Cuando no sé con exactitud el número de parámetros que le pueden llegar a mi función, pondré tres puntos para hacer referencia al resto de posibles parámetros. Dichos parámetros me los mete en un **array** con el nombre de mi parámetro.

<script>

function numeros(num1, num2,...otros){

document.write(num1 +" , "+ num2+"<br>");

document.write(otros);

}

numeros(2,3,4,54,332);

</script>

En este ejemplo otros vale 4,54,332. Es un array.

Si no tuviese los tres puntos valdría 4.

Parámetros spread:

De forma similar, en el siguiente ejemplo hemos creado un array con dos elementos y se lo hemos pasado a la función como parámetro. Si no usamos los tres puntos, lo que le estamos diciendo es que num1=[1,10] y num2=3 y otros =[4,54,332]. Pero si le ponemos los tres puntos lo que me coge es num1=1, num2=10 y otros=[3,4,54,332]

<script>

function numeros(num1, num2,...otros){

document.write(num1 +" , "+ num2+"<br>");

document.write(otros);

}

var arrayNumeros=[1,10];

numeros(…arrayNumeros,3,4,54,332);

</script>

### FUNCIONES ANÓNIMAS.

Las funciones anónimas son aquellas que no tienen nombre y yo las puedo guardar dentro de una variable.

var miFuncion=function(nombre);

return "El nombre es " + nombre;

Para llamarla:

miFuncion(“Angel”);

### CALLBACK.

Los Callbacks en JavaScript son como su propio nombre, en inglés, indica, ***llamadas de vuelta***, quiere decir que cuando invoco una función pasándole como parámetro otra función (el callback) esta función parámetro se ejecuta cuando la función principal ha terminado de ejecutarse. O cuando a nosotros nos interese…

Este concepto que parece muy complicado, es en realidad muy sencillo de entender si lo hacemos con casos prácticos, por ejemplo:

function funcionPrincipal(callback){

 alert('hago algo y llamo al callback avisando que terminé');

 callback();

}

funcionPrincipal(function(){

 alert('terminó de hacer algo');

});

Aquí vemos como la función **funcionPrincipal** se ejecuta recibiendo un argumento que es otra función y que se ejecuta después de que termine su labor llamando a callback.

Pero no te preocupes si no lo has entendido, vamos a hacer algunos ejemplos más para ir cogiéndolo, en realidad es muy sencillo.

Siguiendo con el caso anterior, podemos hacer incluso que la **función callback** reciba **argumentos** que se envían desde la función principal…

|  |  |
| --- | --- |
|  | function funcionPrincipal(callback){      alert('hago algo y llamo al callback avisando que terminé');      callback('Miguel');  }    funcionPrincipal(function(nombre){     alert('me llamo ' + nombre);  }); |

Con esto ya vemos la fuerza que puede tener esta forma de programación, pero ahora imaginemos que queremos encadenar diferentes funciones con callbacks, lo cual es muy sencillo siguiendo con la misma lógica…

|  |  |
| --- | --- |
|  | function funcionPrincipal(callback1, callback2, callback3){      //código de la función principal      callback1();      //más código de la función principal      callback2();      //más código de la función principal      callback3();      //más código si fuera necesario  }    funcionPrincipal(      function(){          alert('primer callback');      },      function(){          alert('segundo callback');      },      function(){          alert('tercer callback');      }  ); |

Aquí vemos como al invocar a la función principal se le pasan como argumentos las tres funciones, que se ejecutan, las cuales podrían recibir parámetros… como hemos visto más arriba, pero de este modo el código queda un poco sucio, ya que no podemos ver claramente las funciones, de modo que podemos declarar las funciones que se enviarán como argumentos aparte, lo cual nos permite también utilizarlas en otras partes del código, tal y como vemos en el siguiente ejemplo:

|  |  |
| --- | --- |
|  | function funcionPrincipal(callback1, callback2, callback3){      //código de la función principal      callback1();      //más código de la función principal      callback2();      //más código de la función principal      callback3();      //más código si fuera necesario  }    function callback1(){          alert('primer callback');      }    function callback2(){          alert('segundo callback');      }    function callback3(){          alert('tercer callback');      }    funcionPrincipal(callback1, callback2, callback3); |

Podemos observar cómo se declaran todas funciones y luego se pasan como argumentos de la **función principal** para poder utilizarlas dentro de la misma. Pero aunque esto daría para otra entrada, vamos a ver por encima el uso de otra función de javaScript llamada **setInterval()**, esta función nos sirve para retardar acciones, y necesita dos parámetros para funciones; una función que invoca a la función que vamos a usar y un valor numérico que dice en milisegundos el retraso y actualización de la función invocada.

Esto que parece un lío, vamos a verlo en un código donde mostraremos un alert, un reloj y un texto, y donde el reloj se va modificando cada segundo y el texto tardará en aparecer 3 segundos:

|  |  |
| --- | --- |
|  | <h3 id="demo"></h3>  <h3 id="demo2"></h3>  <script>  function funcionPrincipal(callback1, callback2, callback3){      //código de la función principal      callback1();      //más código de la función principal      var miVar = setInterval(function(){ callback2() }, 1000);      //más código de la función principal      var miVar2 = setInterval(function(){ callback3() }, 3000);      //más código si fuera necesario  }    function callback1(){          alert('primer callback');      }    function callback2(){          var d = new Date();          var t = d.toLocaleTimeString();          document.getElementById("demo").innerHTML = t;      }    function callback3(){          document.getElementById("demo2").innerHTML = 'Esto es el callback3';      }    funcionPrincipal(callback1, callback2, callback3);  </script> |

El ejemplo en esencia es el mismo, cambiando las funcionalidades de las funciones de callback, pero hemos añadido en las **líneas 8 y 10** la función **setInterval()** para que se retrasen e incluso se actualice, aprovechando de paso la forma de hacer un reloj en javascript…

Existe una sintaxis para las funciones de callback llamada funciones de flecha, en las que se omite la palabra function y se sustituye por una flecha puesta tras el nombre de la función:

funcionPrincipal(function(nombre){

   alert('me llamo ' + nombre);

});

Quedaría así:

funcionPrincipal(nombre =>{

   alert('me llamo ' + nombre);

});

### FUNCIONES FLECHA.

Las funciones de flecha son una manera de definir funciones de callback de una manera mucho más clara y limpia. Simplemente sustituyendo la palabra function por una flecha. Si lleva un parámetro no hace falta poner los paréntesis, si lleva dos ya sí.

function sumame(num1, num2, sumaYmuestra, sumaPorDos) {

var suma = num1 + num2;

sumaYmuestra(suma);

sumaPorDos(suma);

return suma;

}

sumame(5, 7, dato => {

console.log('La suma es ', dato);

},

dato => {

console.log('La suma de dos es ', (dato \* 2));

});

### ÁMBITO DE LAS VARIABLES EN FUNCIONES.

Desde dentro de una función se puede acceder a las variables globales, pero desde fuera de una función no podemos acceder a las variables locales.

### EVENTOS.

La siguiente tabla resume los eventos más importantes definidos por **JavaScript**:

| **Evento** | **Descripción** | **Elementos para los que está definido** |
| --- | --- | --- |
| onblur | Deseleccionar el elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onchange | Deseleccionar un elemento que se ha modificado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |
| onkeypress | Pulsar una tecla | Elementos de formulario y <body> |
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <body> |
| onload | La página se ha cargado completamente | <body> |
| onmousedown | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| onmousemove | Mover el ratón | Todos los elementos |
| onmouseout | El ratón *"sale"* del elemento (pasa por encima de otro elemento) | Todos los elementos |
| onmouseover | El ratón *"entra"* en el elemento (pasa por encima del elemento) | Todos los elementos |
| onmouseup | Soltar el botón que estaba pulsado en el ratón | Todos los elementos |
| onreset | Inicializar el formulario (borrar todos sus datos) | <form> |
| onresize | Se ha modificado el tamaño de la ventana del navegador | <body> |
| onselect | Seleccionar un texto | <input>, <textarea> |
| onsubmit | Enviar el formulario | <form> |
| onunload | Se abandona la página (por ejemplo al cerrar el navegador) | <body> |

Ejemplo “onClick”:

<head>

<meta charset="utf-8">

<title>Documento sin título</title>

<script>

function aviso(){

alert("Mensaje de alert");

}

</script>

</head>

<body>

<img src="18276540.gif" width="225" height="400" alt="" onClick="aviso()"/>

</body>

## DOM

La finalidad de Javascript es interactuar con el código HTML, para ello tenemos que tener muy claro lo que es el **DOM**. (Document Object Model). Es decir, el árbol de etiquetas que componen nuestro HTML.

Suponiendo que tenemos un elemento HTML con un id=caja podemos referirnos a él con:

var cajas=document.getElementById("caja").innerHTML;

De esta forma cargamos en la variable cajas el valor del elemento con id=caja.

También podemos modificar las propiedades o el valor del elemento con id=caja.

var cajas=document.getElementById("caja");

cajas.innerHTML;

cajas.style.background="red";

Otra forma de seleccionar un elemento de nuestra página es con querySelector.

var cajas=document.querySelector("#caja");

De esta forma selecciono también el elemento con id=caja.

Si quiero seleccionar un elemento cuya class=caja:

var cajas=document.querySelector(".caja");

De esta forma si selecciono el elemento sólo es para seleccionar el nombre de la etiqueta,

var cajas=document.querySelector("caja");

con # para hacer referencia a su ID

var cajas=document.querySelector("#caja");

y con punto para hacer referencia a su clase.

var cajas=document.querySelector(".caja");

### SELECCIONAR CLASES Y ETIQUETAS

Acabamos de ver cómo seleccionar elementos concretos por su etiqueta, nombre o clase.

Para seleccionar todos los elementos de un tipo:

var todos\_los\_div=document.getElementsByTagName('div');

Con esto he seleccionado todos los elementos de tipo **div** y me los mete en un array. Ahora ya podríamos sacar el contenido de un elemento de dicho array:

var contenido=todos\_los\_div[2].textContent();

console.log(contenido);

Me mostraría lo que hay en el elemento [2] del array.

También podríamos haber utilizado el innerHTML:

var contenido=todos\_los\_div[2].innerHTML();

La única diferencia es que con **innerHTML(),** si quiero, puedo añadirle un valor nuevo al contenido.

También podemos seleccionar elementos por su etiqueta .class con:

document.getElemenByClassname();

## BOM

Browser Object Model. Contiene multitud de propiedades que nos permiten trabajar con el navegador.

De igual forma que en el DOM podemos seleccionar objetos del documento, con el BOM podemos seleccionar objetos del navegador y modificarlos. Por ejemplo, podemos seleccionar la ventana:

console.log(window.innerHeight);

console.log(window.innerWidth);

En este caso jugamos con el tamaño de la ventana.

Más cosas que podemos hacer:

console.log(screen.Width); Similar a lo anterior

Sacar la URL cargada y más datos:

console.log(window.location);

Sacar la Href:

console.log(window.location.href);

Abrir una pestaña nueva en el navegador:

window.open(URL\_de\_destino);

## EVENTOS

Son funciones que se ejecutan cada vez que yo hago algo. Por ejemplo, ante un movimiento de ratón, un click…etc.

### EVENTOS DE RATÓN.

Podemos capturar eventos de ratón en HTML, con el comando onclick, por ejemplo:

<button id="boton" **onClick**="alert('Boton presionado')">Presionar</button>

o podemos hacerlo desde javascript con addEvenListener, que tendrá una función de callback. Suponiendo que tenemos definida una función que cambia el color del botón:

var boton=document.querySelector("#boton");

boton.addEventListener('click', function(){

cambia\_color();

});

O puedo cambiar directamente el atributo:

boton.addEventListener('click', function(){

boton.style.background="red" ;

});

### EVENTOD DE TECLADO Y FOCO.

Dichos eventos son:

-Focus.

-Blur.

-Keydown.

-Keypress.

-Keyup.

## Geolocalización.

Existen diferentes formas de localizer a un usuario:

-Localizando la IP.

-En HTML5 con la API Geolocation.

-BBDD IP

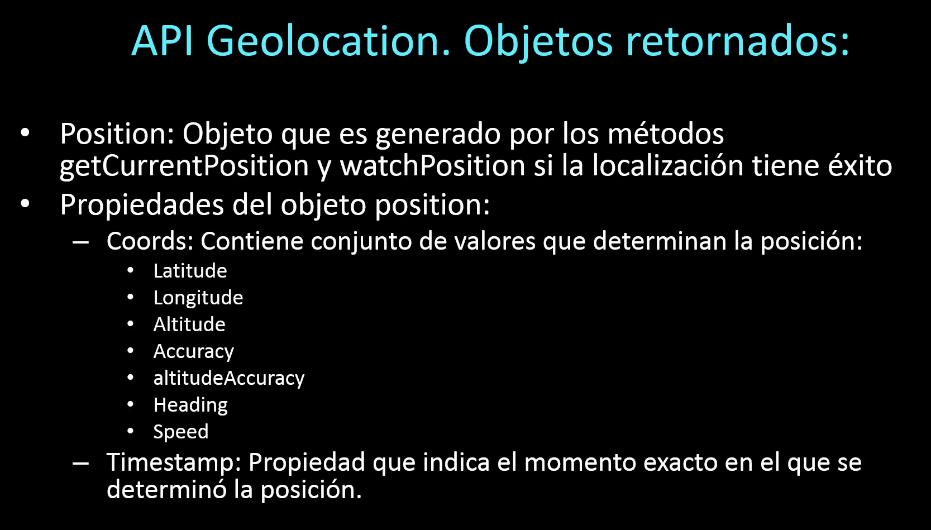
-Otras.

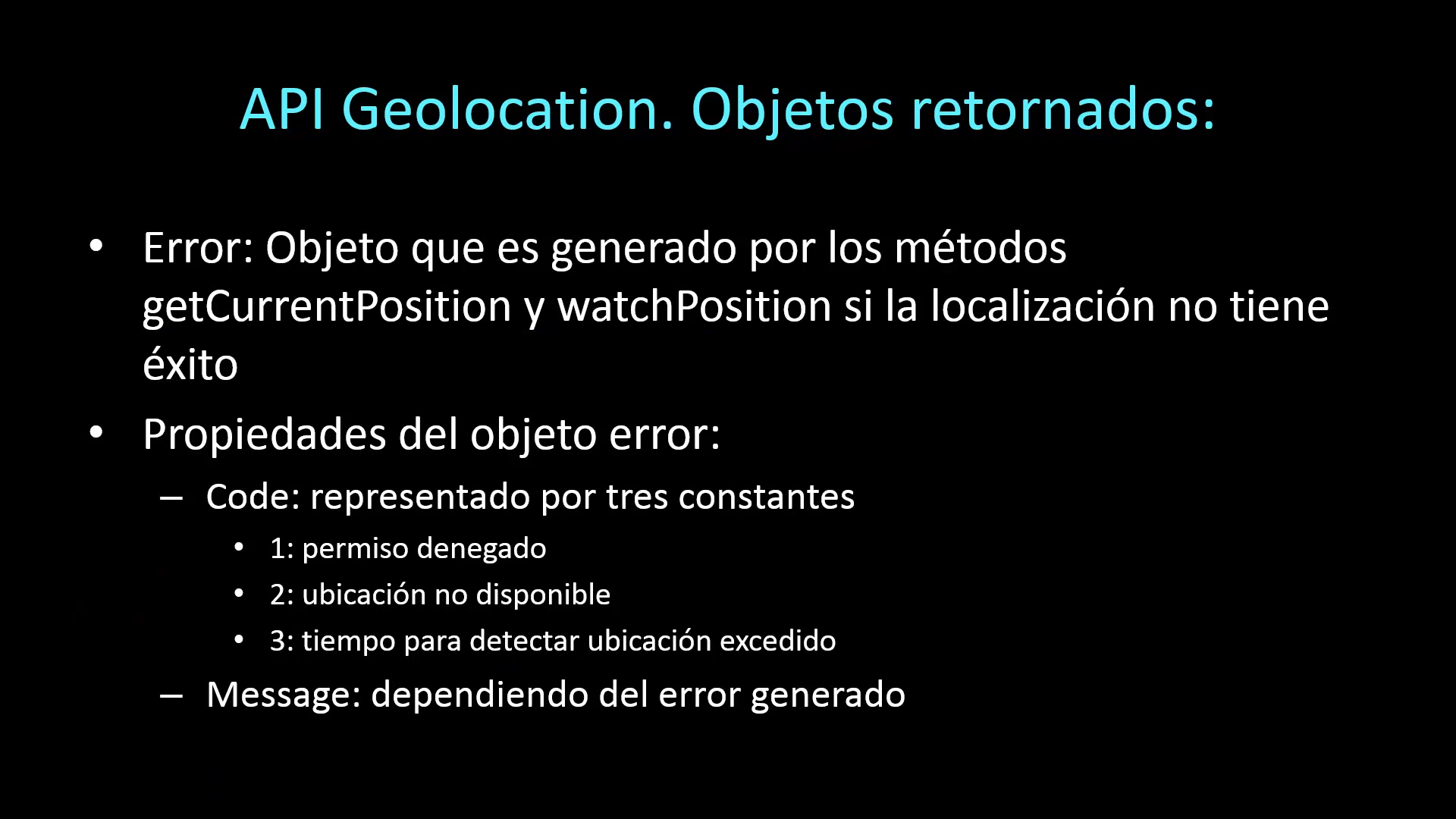
La API Geolocation tiene tres métodos:

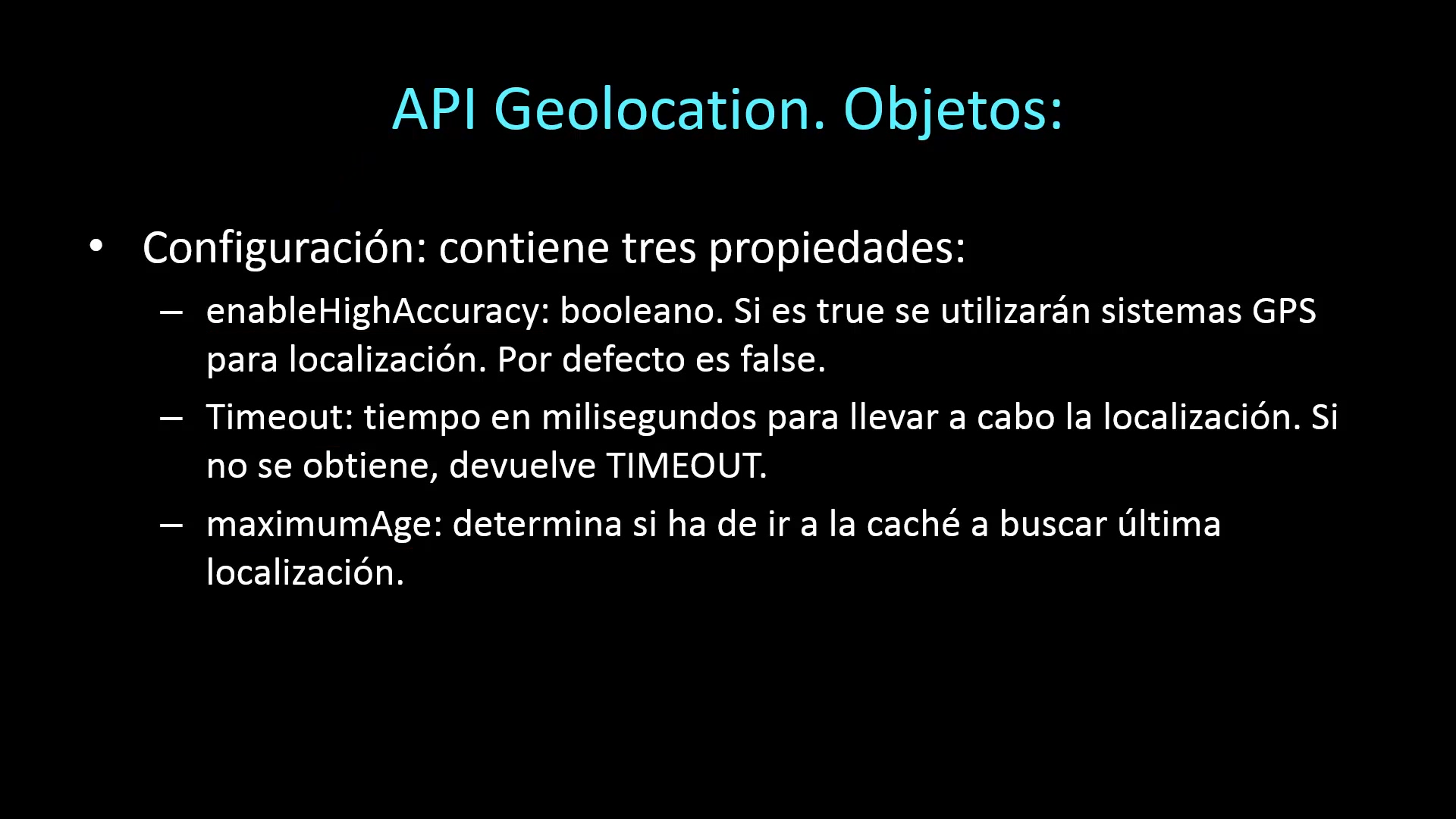
-**GetCurrentPosition**, nos da la posición del usuario. Tiene tres posibles argumentos, ubicación (obligatorio), errores y configuración.

-**WatchPosition**, nos da la posición refrescando cada x tiempo.

-**clearWatch**, resetea WatchPosition.







### Ventanas emergentes