

LMSGI02.2 JavaScript



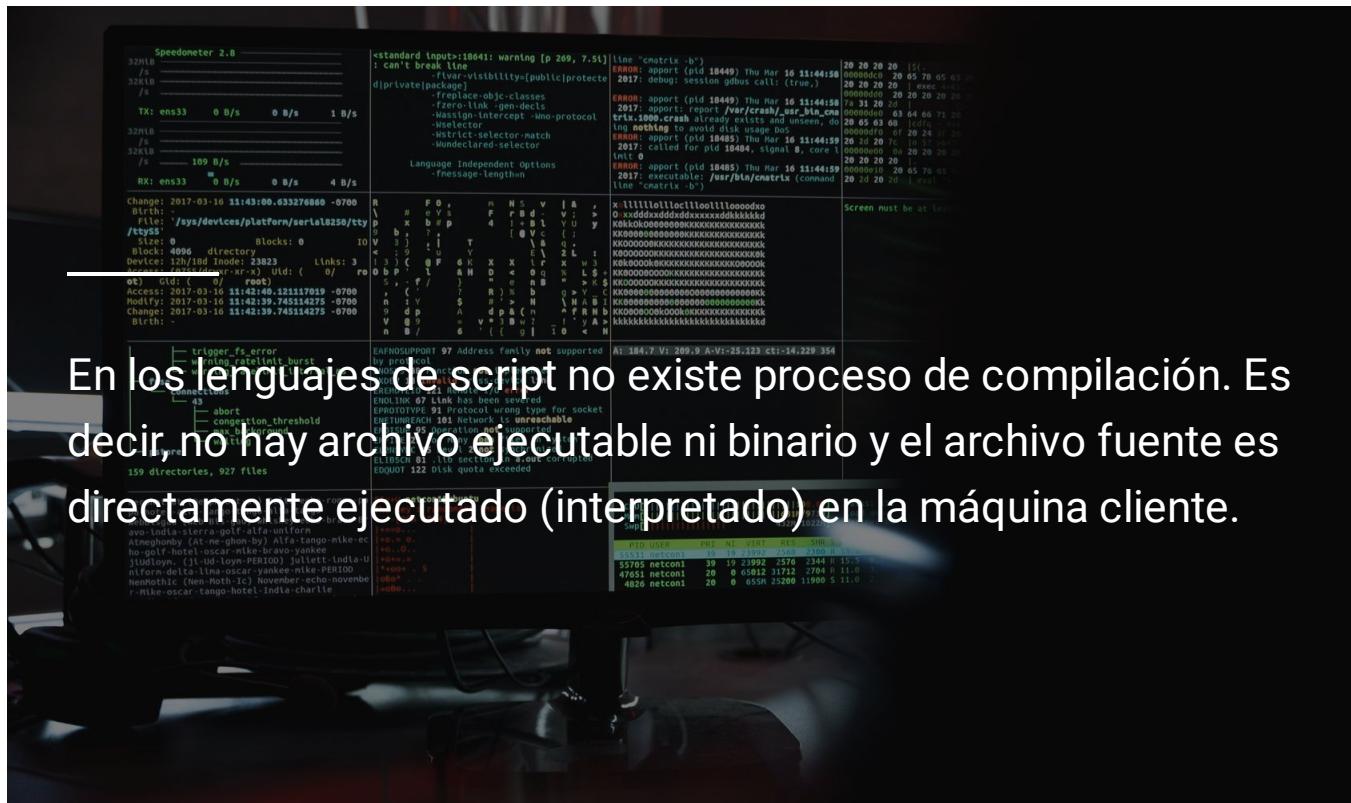
-  Utilización de lenguajes de script del lado cliente
-  Primeros pasos
-  Instrucciones básicas
-  Funciones
-  Objetos de JavaScript
-  Eventos de JavaScript
-  Validación de formularios
-  Autoevaluación

```
  const crypto = req.body.crypto;
  const fiat = req.body.fiat;

  const baseURL = "https://api2.bitcoinaverage.com/indices/global/ticker/";
  const finalURL = baseURL + crypto + fiat;
  request(finalURL, (error, response, body) => {
    const data = JSON.parse(body);
    const price = data.last;

    const currentdate = data.display.timestamp;
    res.write("<p>The current date is " + currentdate + "</p>");
    res.write("<br>The current price of " + crypto + " is " + price + fiat + "<br>");
    res.end();
  });
});
```

Utilización de lenguajes de script del lado cliente



Introducción

En el contexto de los lenguajes de script del lado cliente, el navegador actúa como cliente ante el servidor web que aloja el sitio solicitado, transfiriendo toda la información relevante, como código HTML, CSS, imágenes y videos, al navegador del usuario. Este último puede mostrar las páginas web y ejecutar código en el propio navegador. Por otro lado, los lenguajes del lado servidor suelen ejecutar acciones como acceso a bases de datos y conexiones de red antes de enviar las páginas al navegador, especialmente en aplicaciones dinámicas con mucha interactividad.

Es importante destacar que los lenguajes del lado cliente no reemplazan a los del lado servidor, sino que los complementan. Una opción recomendable es trabajar con lenguajes que combinen ambas tecnologías, como AJAX (Asynchronous JavaScript and XML), que integra HTML, CSS, JavaScript, DOM, XML, XSLT y utiliza el objeto XMLHttpRequest para la comunicación con los servidores en segundo plano. Esto permite transferencias asíncronas de datos entre el cliente y el servidor, atendiendo las peticiones del cliente casi en tiempo real sin necesidad de recargar la página.

Entre los lenguajes del lado cliente más utilizados se encuentran JavaScript, VBScript (compatible solo con Internet Explorer o Chrome), además de lenguajes de script de consola como Bash y PowerShell. Aunque estos lenguajes facilitan el trabajo del desarrollador al no requerir un proceso de compilación, no generando archivos binarios ni ejecutables, el código fuente se interpreta directamente en la máquina cliente, lo que puede sobrecargar el procesador. Suelen emplearse para acciones más simples o programas pequeños.

Características

JavaScript es un lenguaje de programación de scripts orientado a objetos. Esto quiere decir:

- **Javascript es un lenguaje de programación:** permite a los programadores escribir código fuente que será analizado por un ordenador. Donde:
 - Un programador es una persona que desarrolla programas y puede ser un profesional (un ingeniero, matemático, programador informático o analista) ó un aficionado.
 - El código fuente está escrito por el programador y es un conjunto de acciones, llamadas instrucciones, que permiten dar órdenes al ordenador para hacer lo que el programador ha pensado. El código fuente no es algo que se vea cuando el programa está funcionando, al igual que una lavadora en funcionamiento, vemos el efecto de las instrucciones que el programador ha pensado y luego escrito. No se ve pero está ahí.
- **JavaScripts utiliza scripts:** como regla general, hay tres formas de usar el código fuente:
 - Lenguaje compilado: el programa o código fuente escrito por el programador se pasa a otro programa llamado compilador que lee el código fuente y lo convierte ó traduce en un lenguaje que el ordenador será capaz de interpretar, que es el lenguaje binario, escrito por 0 y 1. Lenguajes como C son lenguajes compilados muy conocidos.

- Lenguaje precompilado: el programa o código fuente se compila en parte, por lo general en un código más fácil de leer para el ordenador, pero que aún no es binario. Este código intermedio es para ser leído por lo que se llama una "Máquina Virtual", que ejecutará el código. Lenguajes como Java se llaman precompilados.
 - Lenguaje interpretado: aquí no hay compilación. El código fuente del programa se mantiene sin cambios, y si se quiere ejecutar este código, hay que buscar un intérprete que lea el programa y realice las acciones solicitadas. Los scripts son en su mayoría interpretados y cuando decimos que JavaScript es un lenguaje interpretado, necesitamos un intérprete para ejecutar código JavaScript, y este intérprete que se utiliza viene incluido en los navegadores de páginas web. Así, cada navegador tiene un intérprete JavaScript que varía en función del navegador.
-
- **JavaScript está orientado a objetos:** un lenguaje de programación orientado a objetos es un lenguaje que contiene elementos, llamados objetos y los objetos diferentes tienen características específicas y formas de uso diferente.

Primeros pasos

```
isURL    = ( type == "url" )
isElement = ( type == "element" )
isObject  = ( type == "object" )

// Check if boxer is already active, return if true
if ($("#boxer").length > 0) {
    return;
}

// Kill event
killEvent(e);
```

JavaScript es un lenguaje utilizado principalmente junto con el lenguaje HTML y CSS.

```
// Cache internal data
data = extend({}, {
    $window: $(window),
    $body: $("body"),
    $target: $target,
    $object: $object,
    visible: false,
    resizeTimer: null,
    touchTimer: null,
    gallery: {
        active: false
    }
});
```

Sintaxis de Javascript

La sintaxis de Javascript es sencilla y, generalmente, las instrucciones deben estar separadas por un punto y coma (";") que se coloca al final de cada instrucción. Por ejemplo:

```
instrucción_1;  
instrucción_2;  
instrucción_3;
```

En realidad, el punto y coma no es necesario si la instrucción siguiente está en la línea posterior como en el ejemplo. Sin embargo, si se escriben varias instrucciones en una sola línea, como en el siguiente ejemplo, el punto y coma es obligatorio. De todas formas, es una muy buena costumbre utilizar siempre el punto y coma para evitar problemas.

```
instrucción_1;instrucción_2;instrucción_3  
instrucción_4
```

Ejemplo de código:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Hola BIRT!</title>  
  </head>  
  <body>  
    <script>  
      alert('Hola BIRT!');  
    </script>  
  </body>  
</html>
```

Estructura

Un programa escrito en JavaScript consta de:

- **Variables:** Es una buena idea declarar las variables a utilizar en el script, aunque no sea obligatorio.
- **Instrucciones:** son los comandos u órdenes a ejecutar.
- **Resultado:** Puede ser una acción concreta, un texto, un valor numérico,...
- **Comentarios:** Es una buena idea añadir notas al código para indicar qué objetivo se persigue, comentarios de desarrollo y de resultado.

- **Funciones:** Son conjuntos de instrucciones con una finalidad concreta (un cálculo, una comprobación, una acción,...) y que puede devolver ó no un resultado.

Uso de espacios

Javascript no es sensible a los espacios. Esto significa que se pueden alinear las instrucciones siempre que no interfiera con la secuencia de comandos. Por ejemplo:

```
instrucion_1;
instrucion_1_1;
instrucion_1_2;
instrucion_2; instrucion_3;
```

Uso de sangrías

El uso de la sangría en la programación es una manera de estructurar el código para hacerlo más legible. Esto no afecta para nada a la lógica del programa pero el hecho de que las instrucciones estén priorizadas en varios niveles y crear una jerarquía ayuda a entender el programa.

Por ejemplo:

Bien	Mejorable
<pre>function interruptor(elemID) { var elem = document.getElementById(elemID); if (elem.style.display == 'block') { elem.style.display = 'none'; } else { elem.style.display = 'block'; }}</pre>	<pre>function interruptor(elemID) {var elem = document.getElementById(elemID);if (elem.style.display == 'block') {elem.style.display = 'none';} else {elem.style.display = 'block';}}</pre>

Uso de comentarios

Los comentarios son notas que añade el programador al código para explicar el funcionamiento de un script, una instrucción ó un grupo de instrucciones. Los comentarios no interfieren con la ejecución del programa.

Hay dos tipos de comentarios:

Una línea	Multilínea
<pre>instrucción_1; // Este es el comentario de la primera instrucción instrucción_2; // La tercera instrucción es la siguiente: instrucción_3;</pre>	<pre>/* Este programa o script consta de tres instrucciones: - La primera instrucción sirve para hacer algo - La segunda instrucción sirve para otra cosa - La última instrucción sirve para terminar el programa */ instrucción_1; instrucción_2; /* Este es de una sola línea */ instrucción_3; // Fin y comentario de una sola línea</pre>

Posición del código en la página web

Los códigos escritos en JavaScript son insertados a través del elemento <script>. Este elemento tiene un atributo de tipo que se utiliza para indicar el tipo de lenguaje que vamos a utilizar. En nuestro caso es JavaScript, pero podría ser otra cosa, como por ejemplo VBScript.

En HTML5 no es obligatorio utilizarlo por lo que en los ejemplos mostrados, no incluirán este atributo.

Código incrustado en la página web

Por ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mensaje de bienvenida</title>
  </head>
  <body>
    <script>
      alert("Kaixo BirtLH!");
    </script>
  </body>
</html>
```

Código externo a la página web

Es posible y conveniente, escribir el código JavaScript en un archivo externo con la extensión ".js". Este archivo se llama desde la página web mediante el elemento `<script>` y su atributo `src` con el camino de la dirección (path) del archivo. Por ejemplo:

Si el contenido del archivo con el código JavaScript "hola.js" es:

```
alert ("Kaixo BirtLH!");
```

Desde nuestra página web es posible utilizarla de la forma siguiente:

```
<! DOCTYPE html>
<html>
```

```
<head>
    <title>Mensaje de bienvenida</title>
</head>
<body>
    <script src="hola.js"></script>
</body>
</html>
```

Posición del elemento <script>

En la gran mayoría de las veces, se coloca el elemento <script> dentro de <head> cuando se utiliza para cargar un archivo JavaScript. Eso es correcto, pero como una página web se lee e interpreta por el navegador de forma lineal, es decir, en primer lugar lee <head> y después los elementos de <body> uno después del otro, si se llama a un archivo JavaScript desde el principio de la carga de la página, el navegador cargará este archivo, y si es grande, la carga de la página se ralentiza.

Para superar este problema, es conveniente colocar los elementos <script> justo antes de cerrar <body>. Además, hay algunos navegadores modernos que lo hacen automáticamente. Por ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <title>¡Hola BIRT!</title>
    </head>
    <body>
        <p>
            <!--
                Contenido de nuestra página web
            -->
        </p>
        // Y ahora cargamos el código correspondiente a JavaScript
        <script src="hola.js"></script>
    </body>
</html>
```

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

Instrucciones básicas

En cualquier lenguaje de programación, para lograr el resultado deseado hay que utilizar una serie de instrucciones básicas.

```
isURL    = ( type == "url" )
isElement = ( type == "element" )
isObject  = ( type == "object" )

// Check if boxer is already active, return
if ($("#boxer").length > 0) {
    return;
}

// Kill event
$(document).off("click", ".boxer");
```

Introducción

Antes de ver las instrucciones disponibles en JavaScript, es necesario saber qué es una variable y cuáles son los operadores disponibles en las diferentes instrucciones.

VARIABLES

Una variable es un espacio de almacenamiento donde guardar información de cualquier tipo de datos. Estos tipos de datos pueden ser:

- una cadena de caracteres
- un valor numérico
- estructuras un poco más complejas

Con la declaración de una variable simplemente reservamos espacio de almacenamiento en memoria para usarlo después. Para declarar una variable:

- Se necesita un nombre con caracteres alfanuméricos: letras (a-z,A-Z), números (0-9), guión bajo (_) y dólar (\$) también son aceptados. Además, el nombre de la variable no puede comenzar con un número y no puede formarse sólo con palabras clave utilizadas por Javascript ("palabras reservadas"). Por ejemplo:

```
var miVariable;
```

- JavaScript es un lenguaje "case sensitive" por lo que es sensible a mayúsculas y minúsculas. Por ejemplo, son diferentes:

```
var miVariable;
```

```
var mivariable;
```

- JavaScript es un lenguaje de tipado dinámico, por lo que cualquier declaración de variables se hace con la palabra clave "var" ó "let" sin especificar el tipo de contenido de la variable. Por lo que es posible que una variable sea de un tipo y, posteriormente, de otro conforme avance el programa, aunque es mejor no hacerlo así. En JavaScript hay tres tipos de datos principales:

Tipos de datos	Descripción	Ejemplos
Numérico	Para almacenar un número natural,	<pre>let numero = 1; // Este es un número entero var numero2 = 2.35 e6; //Este es un número en notación científica</pre>

Tipos de datos	Descripción	Ejemplos
	entero, en notación científica, etc.	<pre>var MiNumero = 0x457; //Este último está en notación hexadecimal</pre>
Cadenas de caracteres	Este tipo de datos representa texto.	<pre>var miString1 = "Este es mi primer texto"; //Con comillas dobles let miString2 = 'Este es mi segundo mensaje'; //Con comillas simples var miString3 = 'Este \'contiene\' comillas'; //Almacena comillas let miVariable = '1'; //No es un número sino una cadena de caracteres</pre>
Booleano	Permite almacenar dos estados: verdadero ó falso.	<pre>var miBool1 = true; // A la variable se le asigna "verdadero" let miBool2 = false; // A esta otra se le asigna "falso"</pre>

Declaración con "var" ó "let"

En JavaScript, "var" y "let" son ambos utilizados para declarar variables, pero difieren en su alcance y comportamiento de "elevación" (hoisting).

En general, "let" es preferido en JavaScript moderno debido a su comportamiento más predecible, su alcance de bloque y su prevención de re-declaraciones accidentales.

Alcance (scope)

- "var" tiene un alcance de función. Esto significa que una variable declarada con "var" dentro de una función está disponible en toda la función en la que se declara.
- "let" tiene un alcance de bloque. Esto significa que una variable declarada con "let" dentro de un bloque (como un bucle "for", "if", ó cualquier bloque de código delimitado por "{}") solo está disponible dentro de ese bloque y no fuera de él.

Hoisting

- Las variables declaradas con "var" son "elevadas" (hoisted) al inicio de su contexto de ejecución (ya sea el contexto global ó el de la función en la que se declaran). Esto significa que puede accederse a una variable "var" antes de que se declare en el código. Sin embargo, su valor será "undefined" hasta que se le asigne uno.
- Las variables declaradas con "let" no son hoisted. Si se intenta acceder a una variable "let" antes de que se declare en el código, se obtiene un error de referencia ("ReferenceError").

Re-declaración

- En el caso de "var", se puede re-declarar una variable dentro del mismo ámbito sin obtener un error. Sin embargo, no es una práctica recomendada ya que puede llevar a errores difíciles de detectar.
- Con "let", si se intenta re-declarar una variable en el mismo ámbito, se obtiene un error de sintaxis ("SyntaxError").

Comprobación del tipo de las variables

Para comprobar la existencia de una variable o comprobar su tipo es útil la instrucción "typeof".

Por ejemplo:

```
var miNumero = 1;  
alert (typeof miNumero ); //Se debería mostrar "number"  
  
var miString = "Este es mi texto";  
alert (typeof miString); //Se debería mostrar "string"  
  
var miBooleana = false;  
alert (typeof miBooleana); //Se debería mostrar "boolean"  
  
alert (typeof nada); //Se debería mostrar "undefined" porque la variable no ha sido declarada ó  
no contiene ningún valor
```

Operadores

Ahora se puede comenzar a operar con las variables.

Aritméticos

Operadores	Ejemplos
<ul style="list-style-type: none">• + : símbolo para sumar• - : símbolo para restar• * : símbolo para multiplicar• / : símbolo para dividir	<pre>var Resultado = 1 + 2; alert (Resultado); //Se muestra "3" var Num1=5, Num2 = 2, Resultado; Resultado = Num1 * Num2; alert (Resultado); //Se muestra "10" var Divisor = 2, Resultado1, Resultado2,</pre>

Operadores	Ejemplos
<ul style="list-style-type: none"> • %: símbolo para el resto de una división 	<pre>Resultado3; Resultado1 = (15 + 9) / 3 - 2; Resultado2 = Resultado1 / Divisor; Resultado3 = Resultado1 % Divisor; alert (Resultado1); //Se muestra 6 alert (Resultado2); //Se muestra 3 alert (Resultado3); //Se muestra 0</pre> <pre>var Numero = 2; Numero += 1; alert (Numero); //Se muestra 3</pre>

Lógicos

Se llaman así porque trabajan con valores booleanos.

Operadores	Ejemplos
<ul style="list-style-type: none"> • &&: operador Y (AND). El resultado es verdadero cuando todos los argumentos son verdaderos y falso, en caso contrario. 	<pre>var Resultado = true && true; alert (Resultado); //Se muestra "true" Resultado = true && false; alert (Resultado); //Se muestra "false" Resultado = false && false; alert (Resultado); //Se muestra "false" Resultado = 1>2 && 2>1; alert (Resultado); //Se muestra "false"</pre>
<ul style="list-style-type: none"> • : operador O (OR). El resultado es verdadero cuando alguno de los 	<pre>Resultado = true verdadero; alert (Resultado); //Se muestra "true" Resultado = true false; alert (Resultado); //Se muestra "true" Resultado = false false; alert (Resultado); //Se muestra "false"</pre>

Operadores	Ejemplos
argumentos es verdadero y falso, en caso contrario.	
<ul style="list-style-type: none"> • !: operador NO (NOT). El resultado es verdadero cuando el argumento es falso y falso, en caso contrario. 	<pre>Resultado = false; Resultado = ! Resultado; alert (Resultado); //Se muestra "true" Resultado = ! Resultado; alert (Resultado); //Se muestra "false"</pre>

Comparación

Permiten realizar comparaciones entre diferentes valores.

Operadores	Ejemplos
<ul style="list-style-type: none"> • == : Igual a • != : Diferente a • ==== : Contenido y tipo igual a • !== : Contenido o tipo diferente de • > : Mayor que • >= : Mayor o igual que • < : Menor que • <= : Menor o igual que 	<pre>var Num1 = 2, Num2 = 2, Num3 = 4, miString = '4', Resultado; Resultado = Num1 == Num2; alert (Resultado); //Se muestra "true" porque el valor y el tipo de Num1 es igual a Num2 Resultado = Num1== Num3; alert (Resultado); //Se muestra "false" porque el valor de Num1 es diferente de Num3 Resultado = Num1 < Num3; alert (Resultado); //Se muestra "true" porque la condición de comparación se cumple Resultado = Num3 == miString ; alert (Resultado); //Se muestra "true" porque se comparan valores Resultado = Num3 === miString ; alert (Resultado); //Se muestra "false" porque este</pre>

Operadores	Ejemplos
	operador también compara tipos de variables además de sus valores

Concatenación

—

La concatenación es unir una cadena de texto con otra. Por ejemplo:

```
var miString1= "Hola", miString2= "BIRT", miResultado;
miResultado = miString1 + " " + miString2;
alert (miResultado); //Se debería mostrar "Hola BIRT"
```

Interacción con el usuario

—

Para interactuar con el usuario podríamos utilizar la función prompt (). Por ejemplo:

```
var NombreUsuario = prompt('Introduce tu nombre:');
alert(NombreUsuario); //Se debería mostrar el nombre introducido
```

Conversiones

—

Necesidad	Solución
Para hacer un script que sume dos números proporcionados por el usuario:	Hay que hacer una conversión, convirtiendo en este caso a números con la función parseInt(). var Numero1, Numero2, Resultado; Numero1 = prompt ('Introduce el primero:');

Necesidad	Solución
<pre>var Numero1, Numero2, Resultado; Numero1 = prompt ('Introduce el primero: '); Numero2 = prompt ('Introduce el segundo: '); Resultado = Numero1 + Numero2; alert (Resultado);</pre> <p>Pero este código no funciona como se espera.. ¿Por qué? Porque todo lo que se escribe en el campo de texto prompt () se recupera como una cadena de caracteres.</p>	<pre>Numero2 = prompt ('Introduce el segundo: '); Resultado = parseInt(Numero1) + parseInt(Numero2); alert (Resultado);</pre>
<p>Conversión de un número en una cadena</p> <p>Por ejemplo:</p> <pre>var miString, Numero1 = 3, Numero2 = 2; miString= Numero1+ " " + Numero2; alert (miString) //Se debería mostrar "3 2"</pre>	<p>En este caso, al añadir una cadena vacía entre los dos números, la conversión ha sido automática.</p>

Instrucciones condicionales

A continuación, se muestran las diferentes instrucciones que se pueden ejecutar en JavaScript para llevar a cabo lo que se quiere realizar con el script.

Instrucción IF

Con este tipo de estructuras se ejecutan instrucciones cuando se cumple alguna condición.

Sintaxis	Ejemplos
<pre>if (condición) { instrucción_1; ... instrucción_n; }</pre>	<pre>if (true) { alert ("Este mensaje se muestra siempre."); } if (false) { alert ("Este mensaje, en cambio, nunca se mostrará."); } if (1<2 && 2>=1) { //La condición siempre es "true" alert ('La condición se cumple.'); } if (1>2 2<=1) { //La condición siempre es "false" alert ("La condición no se cumple."); //Esta instrucción nunca se ejecuta } if (confirm('¿Quieres ejecutar el código JavaScript que sigue?')) { alert ('El código se ha ejecutado.') } //Con esto se añade un poco de interacción con el usuario. Se pasa //un argumento que es una cadena que se mostrará en pantalla y la //respuesta del usuario es una condición booleana</pre>

Instrucción IF-ELSE

Se ejecuta un código en caso de que se cumpla la condición y otro código en caso contrario.

Sintaxis	Ejemplos
<pre>if (condición) { //Esto se ejecuta cuando se cumple la condición instrucción_1; ... instrucción_n; } else { //Esto se ejecuta cuando no se cumple instrucción_n+1; ... instrucción_m; }</pre>	<pre>if (confirm("¿Aceptas las condiciones?")) { alert ("Has entrado a la zona A."); } else { alert ("Has entrado a la zona B."); }</pre>

Instrucción SWITCH

Si la condición a evaluar tiene más de 2 posibilidades, quizás es más conveniente utilizar una instrucción de este tipo.

Por ejemplo, si tenemos un coche de 3 puertas, y se ejecutan diferentes acciones dependiendo de la puerta que ha sido abierta, con las instrucciones IF-ELSE sería muy largo y confuso.

Sintaxis	Ejemplos
<pre>switch (condición) { case x: // instrucciones_x break; case y: // instrucciones_y break; ... default: //</pre>	<pre>var puerta = (prompt("Elige la puerta a abrir [1-3]:")); switch (puerta) { case "1": alert ("Has abierto la puerta 1."); break; case "2": alert ("Has abierto la puerta 2."); break; case "3": alert ("Has abierto la puerta 3.");</pre>

Sintaxis	Ejemplos
instrucciones_SiNingunaSeCump le }	break; default: alert ("Solamente existen 3 puertas."); }

Una cosa que hay que tener en cuenta es que hay que utilizar la instrucción "break" para no seguir ejecutando el resto de instrucciones del switch.

Instrucciones iterativas

A veces, es necesario repetir la ejecución de algunas instrucciones. Es decir, un bucle es una estructura que repite una serie de instrucciones en función de una condición. Cada vez que el contenido del bucle se repite se dice que se hace una iteración. Para el control del número de repeticiones, se utilizan condiciones lógicas ó a veces se utiliza un contador que habrá que ir incrementando ó decrementando en cada repetición. Para el caso de los contadores, a tener en cuenta lo siguiente:

```

let Numero = 0;
var Resultado1, Resultado2;
Numero = Numero + 1;
alert (Numero); //Se muestra "1"
Numero = Numero -1;
alert (Numero); //Se muestra "0"
Numero++;
alert (Numero); //Se muestra "1"
Numero--;
alert (Numero); //Se muestra "0"
Resultado1 = ++Numero;
Resultado2 = Numero++;
alert (Numero); //Se muestra "2"
  
```

```
alert (Resultado1); //Se muestra "1"  
alert (Resultado2); //Se muestra "1"
```

Bucle WHILE

Se ejecuta el bucle mientras se cumple la condición. Una vez que la condición evaluada sea falsa, finaliza la ejecución del bucle.

Sintaxis	Ejemplos
while (condición) { instrucción_1; ... instrucción_n; }	let Numero = 1; while (Numero <=5) { alert (Numero); //Se muestra "1","2","3","4","5" Numero++; }

Bucle DO WHILE

Similar al anterior pero el cuerpo del bucle se ejecuta al menos una vez, ya que la condición se evalúa después del cuerpo del bucle.

Sintaxis	Ejemplos
do { instrucción_1; ... instrucción_n; } while (condición);	let Numero = 1; do{ alert(Numero); //Se muestra "1","2","3","4","5" Numero++; } while (Numero<=5);

Bucle FOR

Aquí encontramos tres bloques: inicio, condición e incremento. Los tres bloques que forman el bucle for no se ejecutan a la vez:

- inicio: se ejecuta justo antes de que comience el bucle
- condición: se ejecuta antes de cada repetición del bucle
- incremento: se ejecuta después de cada iteración

El bucle for es útil para contar y para repetir el cuerpo del bucle un número determinado de veces.

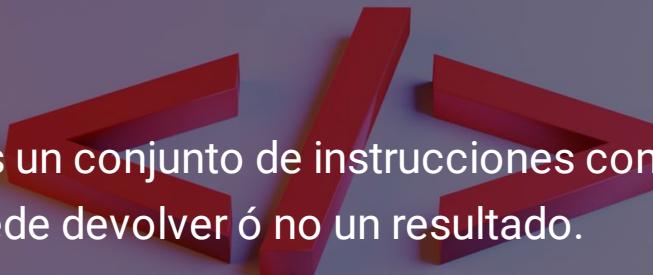
Sintaxis	Ejemplos
<pre>for (inicio;condicion;incremento) { instrucción_1; ... instrucción_n; }</pre>	<pre>for (var i = 1; i<=5; i++) { alert ('Contador: '+ i); }</pre>

Hay que tener la precaución de que una vez que se ejecuta el bucle for, la variable utilizada sigue existiendo, pudiendo interferir con otras instrucciones y producir efectos indeseados.

También existe la posibilidad de no utilizar los 3 bloques, como en el ejemplo:

```
var Registro = {nombre:"Fernando", apellido:"Arregi", edad:23};
var miString = "";
var i;
for (i in Registro) {
  miString += Registro[i] + " ";
}
alert(miString);
```

Funciones



Una función es un conjunto de instrucciones con un objetivo concreto y puede devolver ó no un resultado.

¿Qué es una función?

Una función se define de la siguiente manera:

- **function**
- **nombre**
- seguido por un **par de paréntesis** (uno de apertura y uno de cierre) donde se indican los argumentos o parámetros

Con return, además, la función devuelve un resultado. Un ejemplo de definición y uso de una función:

Por ejemplo:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      function mensaje() {
        alert("Kaixo BirtLH");
      }
    </script>
    <button type="button" onclick="mensaje()">Llamar a la función</button>
  </body>
</html>
```

JavaScript trabaja con funciones definidas por el usuario y con funciones predefinidas. Algunas de las funciones predefinidas son:

Nombre	Descripción	Ejemplo
alert()	Para mostrar un mensaje	Ejemplo anterior
confirm()	Con un cuadro de diálogo se pregunta al usuario sobre alguna cuestión cuyo resultado sea true (verdad) false (falso) y que se guarda en la variable dada.	var enviar=confirm("¿Enviar formulario?"); if (enviar==true) { ... } else { ... }
prompt()	Se piden datos al usuario donde el primer parámetro	var numero=prompt("Introduce un número","");

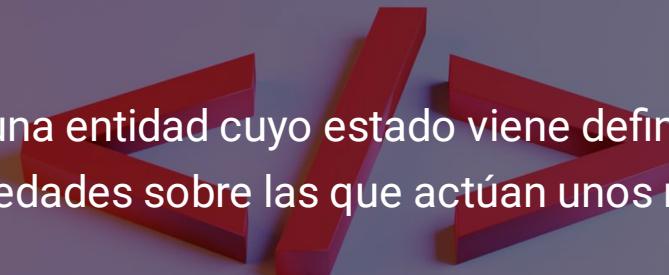
Nombre	Descripción	Ejemplo
	es el texto del cuadro de diálogo y el segundo, opcional, el valor por defecto.	<pre>if (numero=="7") { alert("Tendrás buena suerte"); } else { alert("Otra vez será!. No es el "+numero); }</pre>

Tratamiento de fecha y hora

JavaScript no dispone de un tipo de dato para fechas. Sin embargo, el objeto Date tiene sus métodos para trabajar con fechas y horas:

Método	Descripción
getDate()	Devuelve el día del mes en el rango 1-31.
getDay()	Devuelve el día de la semana en el rango 0-6. El domingo es 0.
getMonth()	Devuelve el mes en el rango 0-11.
getFullYear()	Devuelve el año en formato de 4 dígitos.
getHours()	Devuelve la hora en el rango 0-23.
getMinutes()	Devuelve los minutos en el rango 0-59.
getSeconds()	Devuelve los segundos en el rango 0-59.

Objetos de JavaScript



Un objeto es una entidad cuyo estado viene definido por una serie de propiedades sobre las que actúan unos métodos ó funciones

¿Qué es un objeto?

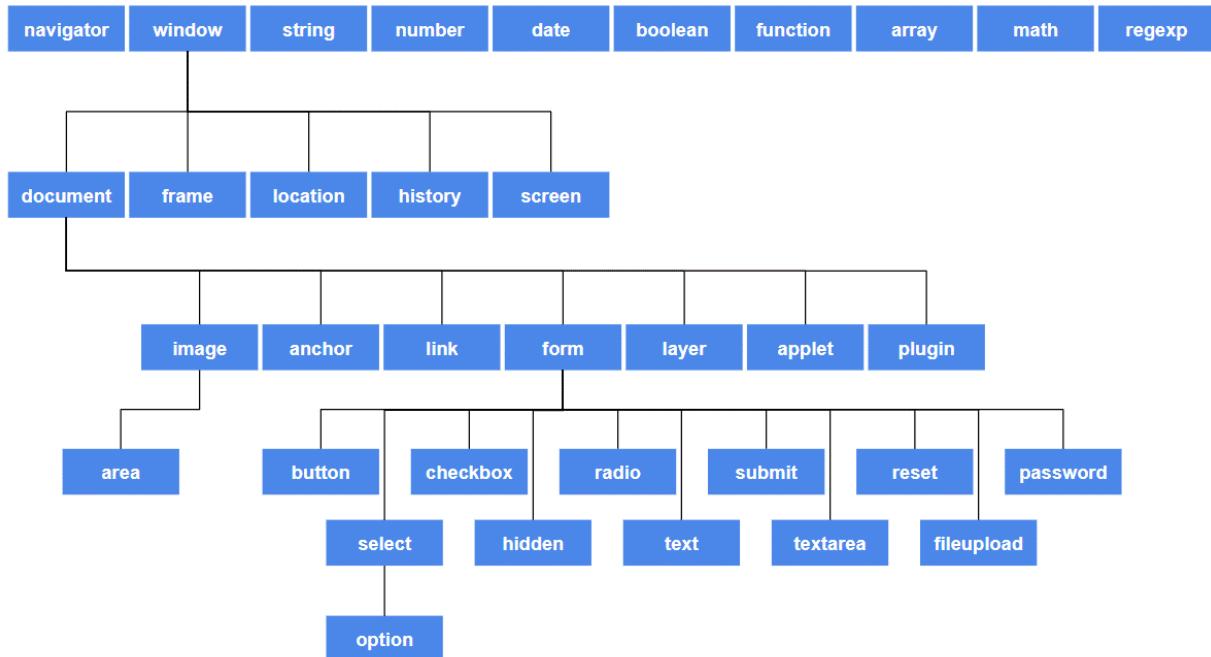
Un objeto tiene unas propiedades que lo definen y dispone de métodos ó funciones que modifican esas propiedades.

Para acceder a la propiedad de un objeto se escribe **NombreObjeto.propiedad** (también es posible hacerlo de la forma **NombreObjeto["propiedad"]**) y para acceder a un método de un objeto **NombreObjeto.método([parámetros])**.

Por ejemplo:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>JavaScript Propiedades y Métodos</h1>
    <script>
      let MiMensaje1 = "Kaixo BirtLH!";
      //muestra la longitud del string
      alert(MiMensaje1.length);
      //asigna una parte del string y se muestra
      let MiMensaje2 = MiMensaje1.substring(6,13);
      alert(MiMensaje2);
    </script>
  </body>
</html>
```

Jerarquía de objetos en JavaScript



DOM

El DOM (Document Object Model) es un estándar del W3C (World Wide Web Consortium) que permite el acceso y la modificación de los diferentes elementos de un documento HTML.

Cuando un navegador carga una página web, crea de forma automática una estructura jerárquica con todos los objetos que componen dicho documento. A cada elemento de esta estructura se llama nodo y desde JavaScript se puede acceder a los nodos para:

- Modificar los atributos y estilos de todos los elementos del documento HTML.
- Añadir o eliminar elementos y atributos existentes.
- Responder a los eventos que ocurren en la página.
- Crear nuevos eventos.

Los nodos tienen la siguiente estructura: Nodo = elemento (etiqueta) + atributos + eventos + texto.

Objeto window

El objeto window() representa una ventana abierta en el navegador. De esta forma se dice que es el contenedor principal de todo lo que se muestra en el navegador y es el primer objeto que se carga en memoria cuando se abre una ventana o una pestaña en el navegador.

A partir de él se pueden crear nuevos objetos window que serán nuevas ventanas que se pueden controlar desde la ventana anterior. También permite cerrar ventanas, mostrar mensajes, etcétera. E decir, para JavaScript, tanto las ventanas de navegador como las pestañas son objetos window.

Los documentos se cargan dentro de la sección del objeto window y desde él se controla todo lo relativo a la ventana: barras de desplazamiento, tamaño de la ventana, barra de herramientas, de estado,

Nombrar una ventana

Para asignar un nombre u obtener el nombre de una ventana se utiliza la propiedad name del objeto window. Por ejemplo:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Objetos de JavaScript</h1>
    <script>
      window.name="MiVentanaBirtLH";
      alert(window.name);
    </script>
  </body>
</html>
```

Abrir y cerrar una ventana

Para abrir y cerrar una ventana se utilizan los métodos open() y close() del objeto window. El usuario abre la ventana principal del navegador cuando accede a una URL y desde JavaScript no se puede crear esta ventana, pero sí si se pueden abrir ventanas dentro de ella.

Los parámetros de open():

- URL (opcional): dirección de la página web a cargar en la ventana creada. Si se indican comillas seguidas se abre una ventana en blanco.
- Nombre (opcional): el nombre de la ventana.
- Opciones (opcional): permite indicar las características de la nueva ventana: anchura y altura de la ventana, mostrar la barra de direcciones, la barra de menú, si permite redimensionar la ventana, ...

Utilizando close() solo se pueden cerrar las ventanas abiertas con open(). Por ejemplo:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Ventana Principal</title>
    <script>
      //la variable MiVentana contendrá una referencia al popup
      //será variable global para poder usarla desde las distintas funciones
      var MiVentana;
      function abrir(){
        //se guarda la referencia de la ventana para poder utilizarla luego
        MiVentana=window.open("", "miventana", "width=400,height=200,menubar=0");
      }
      function cerrar(){
        //la referencia de la ventana es el objeto window del popup
        MiVentana.close();
      }
    </script>
  </head>
  <body>
    <h1>Esta es la ventana principal</h1>
    <input type=button value="Abrir ventana" onclick="abrir()">
    <input type=button value="Cerrar ventana" onclick="cerrar()">
  </body>
</html>

```

Objeto document

Existe un objeto document para cada página cargada en una ventana. Dicho objeto contiene todos los elementos visibles de la página.

El objeto document enlaza directamente con window y contiene toda la jerarquía de objetos dentro de la página HTML.

Por ejemplo, en Notepad++ se muestra de manera muy esquemática, que sirve para hacerse una idea, el árbol DOM correspondiente a una página HTML, recordando que el elemento html es sucesor de document.

The screenshot shows a browser window titled "EJEMPLO DE DOM.html". On the left, the code editor displays the following HTML structure:

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="utf-8"/>
5     <title>Ejemplo de DOM</title>
6   </head>
7   <body>
8     <h1>Título de nivel 1</h1>
9     <p>párrafo</p>
10    </body>
11 </html>
```

On the right, the "XML Treeview" panel shows the corresponding DOM tree structure:

```
html
  head
    meta
    title
  body
    h1
    p
```

En un navegador Firefox ó Chrome, también se puede visualizar el DOM de una página en las Herramientas para Desarrolladores.

El DOM define 12 tipos de nodos, pero en las páginas HTML generalmente se utilizan:

- **document**: es la raíz de la estructura en árbol.
- **element**: nodo sucesor del raíz, identifica la etiqueta <html> del que a su vez dependen otros nodos element que contienen etiquetas HTML.
- **attr**: identifica los atributos de las etiquetas de la forma 'atributo=valor'.
- **text**: identifica el texto encerrado por una etiqueta.
- **comment**: identifica los comentarios incluidos en la página HTML.

Así, el árbol DOM del código anterior quedaría como la imagen siguiente:

```

document
  #element:HTML
    #attr: lang="es"
    #element:HEAD
      #element:META
        #attr: charset="utf-8"
      #element:TITLE
        #text: Ejemplo de DOM
    #element:BODY
      #element:H1
        #text: Título de nivel 1
      #element:P
        #text: párrafo

```

En la siguiente tabla se muestran las propiedades y métodos más usuales para acceder y modificar elementos HTML.

Acceso a los nodos	Descripción
getElementById(id)	Encuentra un elemento por su "id".
getElementsByName(nombre)	Encuentra un elemento por su nombre.
Modificar nodos	Descripción
<ul style="list-style-type: none"> • createElement(elemento) • remove() • removeChild(elemento) 	<ul style="list-style-type: none"> • Crea un elemento HTML. • Elimina un elemento HTML accedido previamente.

- | | |
|--|--|
| <ul style="list-style-type: none">• appendChild(elemento)• replaceChild(nuevo, viejo)• write(texto) | <ul style="list-style-type: none">• Elimina un elemento de una lista.• Añade un elemento a una lista.• Sustituye un elemento por otro.• Escribe un texto en la salida HTML. |
|--|--|

Por ejemplo, para eliminar un elemento, el código podría ser el siguiente:

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JavaScript HTML DOM</h2>
    <h3>Eliminar un elemento.</h3>
    <p id="p1">Este es el párrafo1.</p>
    <p id="p2">Este es el párrafo2.</p>
    <input type="button" onclick="MiFuncion()" value="Eliminar"/>
    <script>
      function MiFuncion() {
        document.getElementById("p1").remove();
      }
    </script>
  </body>
</html>
```

Eventos de JavaScript



Un evento es una acción del usuario o del navegador que sucede en el sistema.

Un evento sucede, por ejemplo, cuando el usuario hace clic sobre un botón en una página web ó el cursor se posiciona encima de una imagen.

Cuando se producen, es posible responder con alguna acción por medio de las funciones asociadas ("handlers"-manejadores) a dicho evento. Es decir, el **manejador** es el código que responde al evento.

La gestión de eventos de JavaScript consiste en asociar el 'manejador de evento' a un elemento HTML y esto se puede hacer de tres formas:

Evento	Descripción
onclick	El usuario hace clic en un elemento HTML.
onmouseover	El usuario mueve el mouse sobre un elemento HTML.
onmouseout	El usuario aleja el mouse de un elemento HTML.
onkeydown	El usuario presiona una tecla del teclado.
onload	El navegador ha terminado de cargar la página.

La gestión de eventos de JavaScript consiste en asociar el 'manejador de evento' a un elemento HTML y esto se puede hacer de tres formas:

Desde HTML

En este caso, se usan los **atributos de eventos de elementos** HTML5. Por ejemplo:

```
<input type="button" onclick="alert ('Kaixo BirtLH!'); " value="mensaje"/>
```

En este ejemplo, al elemento HTML <input> se le ha asignado un atributo de evento onclick. El valor asignado a este atributo es el código JavaScript que ejecutará la acción asociada, mostrando el texto 'Kaixo BirtLH!'.

Este mecanismo es sencillo, pero se puede complicar cuando hay varias acciones a ejecutar. En este caso es mejor utilizar las llamadas a funciones que recopilan todas las acciones, así:

```
<input type="button" onclick="saludar()" value="mensaje"/>
<script>
    function saludar() { alert("Kaixo BirtLH!"); };
</script>
```

Desde JavaScript

Consiste en asignar un manejador ("**handler**") al elemento HTML desde JavaScript sin necesidad de añadir atributos de eventos en HTML. De esta forma, los códigos HTML y JavaScript están separados y mejor organizado. Por ejemplo:

```
function MiFuncion () { Código };

elementoHTML.onclick=MiFuncion;
```

La asignación del manejador debe hacerse al final del documento para estar seguros de que todos los elementos implicados están ya cargados.

Con addEventListener()

El método **addEventListener()** (literalmente "añadir escuchador de evento") modifica el DOM al cargar el documento y permite añadir eventos a los nodos de un documento web. El elemento puede ser cualquier elemento HTML5 existente en el documento web.

Este método tiene tres argumentos:

- nombre del evento
- función a ejecutarse
- un valor booleano (true ó false) que define el orden del flujo de eventos cuando hay varios eventos posibles

Por ejemplo:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>El objeto documento</h1>
    <h2>Ejemplo del método addEventListener()</h2>
    <p>Haz Click en cualquier sitio del documento.</p>
    <h1 id="demo"></h1>
```

```

<script>
    document.addEventListener("click", MiFuncion);
    function MiFuncion() {
        document.getElementById("demo").innerHTML = "Kaixo BirtLH!";
    }
</script>
</body>
</html>

```

En este ejemplo, el código HTML define un encabezado con un ID "demo". Luego, el código JavaScript obtiene una referencia a este encabezado utilizando getElementById y agrega un event listener para el evento "click" utilizando addEventListener. Cuando se hace clic, se ejecuta la función que muestra un mensaje de bienvenida.

Otro ejemplo:

```

<!DOCTYPE html>
<html lang="es">
<head>
    <title>Ejemplo de EventListener</title>
    <style>
        .cambiarColor {
            padding: 20px;
            border: 2px solid #333;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <h1>Pasa el mouse sobre el cuadro para cambiar el color de fondo!</h1>
    <div class="cambiarColor" id="miCuadro">Pasa el mouse aquí</div>
    <script>
        // Obtener una referencia al elemento
        var cuadro = document.getElementById("miCuadro");

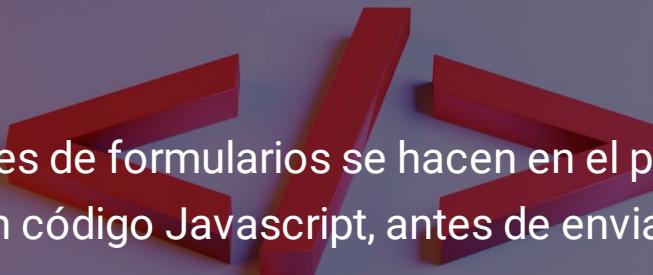
        // Agregar un event listener para el evento "mouseover"
        cuadro.addEventListener("mouseover", function() {
            // Acción a realizar cuando el mouse pasa sobre el cuadro
            document.body.style.backgroundColor = "lightblue";
            cuadro.style.backgroundColor = "lightgreen";
        });

        // Agregar un event listener para el evento "mouseout"
        cuadro.addEventListener("mouseout", function() {
            // Acción a realizar cuando el mouse sale del cuadro
            document.body.style.backgroundColor = "lightgreen";
            cuadro.style.backgroundColor = "lightblue";
        });
    </script>

```

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

Validación de formularios



Las validaciones de formularios se hacen en el propio navegador, con código Javascript, antes de enviarlo.

Introducción

Validar un formulario implica verificar que todos los datos ingresados sean correctos. Por ejemplo, si el formulario requiere un correo electrónico o un DNI, se debe asegurar que estos campos estén correctamente escritos antes de continuar.

Esta validación puede realizarse en el lado del servidor, utilizando programas como PHP, ó en el lado del cliente, mediante scripts de JavaScript. La validación en el lado del servidor es más segura pero más compleja, mientras que en el lado del cliente es más simple y rápido, ya que los

errores se notifican instantáneamente al usuario. Sin embargo, no se debe depender únicamente de JavaScript, ya que si el usuario desactiva su ejecución, la validación no se realiza. Es importante validar los campos en tiempo real a medida que se completan o al presionar el botón de envío, verificando la existencia de un valor, la corrección de datos numéricos y el cumplimiento de patrones específicos como fechas o DNI's. La validación suele implementarse mediante una función que se incorpora al formulario, especificando la acción, el método y la función de validación.

Es importante mencionar que existen muchas formas de hacer este proceso.

Proceso

Si en el formulario hay algún campo no llenado ó con información errónea, el formulario muestra el campo que está incorrecto y solicitará al usuario que lo cambie. Si todos los datos del formulario son correctos se envía el formulario.

Básicamente, el proceso es siempre el mismo:

1. Acceder a un campo del formulario.
2. Comprobar que los valores introducidos son los esperados.
3. Si hay más campos, volver al paso 1.
4. Se realiza el envío del formulario.

Varias alternativas:

Validación definiendo una función para el evento onsubmit

En este primer ejemplo, se define una función que se ejecutará al enviar el formulario con el evento onsubmit del formulario con Javascript, ya que el momento de enviarlo es el adecuado para validar los datos. Realizaremos la validación y luego, si ha ido todo bien, lo enviaremos.

```
<form action="" method="" id="" name="" onsubmit="return validar()">
/* campos del formulario */
```

```

</form>

Donde la función de validación podría ser, por ejemplo:
function validar() {
    if (primera comprobación de un campo del formulario) {
        // Si no se cumple...
        alert('[ERROR] El campo del formulario debe tener un valor de...');

        return false;
    } else if (Segunda comprobación de otro campo del formulario) {
        // Si no se cumple...
        alert('[ERROR] El campo del formulario debe tener un valor de...');

        return false;
    } else if (Tercera comprobación o validación en algún campo del formulario) {
        // Si no se cumple...
        alert('[ERROR] El campo del formulario debe tener un valor de...');

        return false;
    }

    //etc
    // La validación del formulario es correcta y devolvemos el valor true
    return true;
}

```

Validación capturando el evento submit del formulario

En este primer ejemplo, vamos a capturar el evento submit del formulario con Javascript, ya que el momento de enviarlo es el adecuado para validar los datos. Realizaremos la validación y luego, si ha ido todo bien, lo enviaremos.

Para ello, el formulario tiene un campo "id", en la etiqueta FORM, que servirá para acceder a él mediante Javascript. Todos los campos que necesitamos validar, también tienen su "id".

```

<form action="mailto:info@info.com" method="post" id="formulario">
    <p>Usuario: <input type="text" name="usuario" id="usuario"></p>
    <p>Clave: <input type="password" name="clave" id="clave"></p>
    <input type="submit" value="ENVIAR">
    <input type="reset" value="Limpiar">
</form>

```

- En este caso, definiremos un evento "submit" para capturar el instante en el que el usuario enviará el formulario.
- Una vez capturado el evento submit del formulario con Javascript, vamos a bloquear el envío, evitando que se puedan enviar datos incorrectos.
- Validaremos todos los campos del formulario.
- Si todos los campos se validan correctamente, entonces realizaremos el envío nosotros mediante Javascript.

Por tanto, hay una parte en la que se definen los eventos de Javascript necesarios para enganchar nuestros manejadores para procesar el envío del formulario y otra parte en la que realizamos la validación propiamente dicha.

```
document.addEventListener("DOMContentLoaded", function() {
  document.getElementById("formulario").addEventListener("submit", validacion);
});

function validacion(evento) {
  evento.preventDefault();
  var usuario = document.getElementById("usuario").value;
  if(usuario.length == 0) {
    alert("Usuario está vacío");
    return;
  }
  var clave = document.getElementById('clave').value;
  if (clave.length < 8) {
    alert("Clave no válida");
    return;
  }
  this.submit();
}
```

Comenzamos definiendo un manejador para el evento "DOMContentLoaded". Este manejador nos asegura que el resto del código a ejecutar lo haga cuando verdaderamente el navegador esté listo.

Luego definimos un manejador para el evento "submit" del formulario. Como manejador para este segundo evento indicamos la función "validacion()". El acceso al DOM de los elementos de la página, se hace con "document.getElementById()" y enviando su identificador (atributo "id") como parámetro.

La función validacion() recibe como parámetro el objeto evento. Esto es algo que ocurre con todos los manejadores de eventos JavaScript. Este objeto "evento" nos sirve para hacer diversas cosas, como por ejemplo detener el comportamiento por defecto de un evento. Como se trata del evento submit, el comportamiento predeterminado es justamente el envío del formulario, que hemos detenido con la instrucción "evento.preventDefault()".

A continuación vamos accediendo y verificando cada elemento del formulario. En el caso que no lo sean, mostramos un mensaje en una caja de alerta y luego salimos de la función con "return". Ese return permite que, una vez que se ha detectado un error, no se siga ejecutando el código de la función.

Por último, si todo ha ido bien, enviamos el formulario utilizando el método submit() del formulario. Como estamos dentro de un manejador de evento definido por el formulario, la variable "this" corresponde con el propio formulario. Por ello, el envío lo podemos realizar con un simple "this.submit()".

El código completo sería:

```
<!DOCTYPE html>
<html>
<body>
<h1>Validación de formularios-1</h1>
<h2>Formulario de BirthLH</h2>
<script>
document.addEventListener("DOMContentLoaded", function() {
```

```

document.getElementById("formulario").addEventListener("submit",validacion);
}

function validacion(evento) {
  evento.preventDefault();
  var usuario = document.getElementById("usuario").value;
  if(usuario.length == 0) {
    alert("Usuario está vacío");
    return;
  }
  var clave = document.getElementById("clave").value;
  if (clave.length < 8) {
    alert("Clave no válida");
    return;
  }
  alert("Eskerrik asko");
  this.submit();
}
</script>
<form action="mailto:info@info.com" method="post" id="formulario">
<p>Usuario: <input type="text" name="usuario" id="usuario"></p>
<p>Clave: <input type="password" name="clave" id="clave"></p>
<input type="submit" value="ENVIAR">
<input type="reset" value="Limpiar">
</form>

</body>
</html>

```

Validación usando el DOM para acceder a los campos

El código completo sería el siguiente:

```

<!DOCTYPE html>
<html>
<body>
<h1>Validación de formularios-2</h1>
<h2>Formulario de BirtLH</h2>
<script>
function ValidaEnvia() {
//validar el nombre
if (document.formularioBirtLH.usuario.value.length==0) {

```

```

        alert("Usuario está vacío");
        document.formularioBirtLH.usuario.focus();
        return;
    }
    //validar la clave
    if (document.formularioBirtLH.clave.value.length<8) {
        alert("Clave no válida");
        document.formularioBirtLH.clave.focus();
        return;
    }
    //el formulario se envia
    alert("Eskerrik asko");
    document.formularioBirtLH.submit();
}
</script>
<form name="formularioBirtLH" action="mailto:info@info.com" method="post">
<p>Usuario: <input type="text" name="usuario"></p>
<p>Clave: <input type="password" name="clave"></p>
<input type="button" value="Enviar" onclick="ValidaEnvia()">
<input type="reset" value="Limpiar">
</form>
</body>
</html>

```

En este caso, se utiliza el nombre del formulario, "formularioBirtLH ", para referirse a él en JavaScript.

El botón de enviar, que en lugar de ser un submit corriente, es un botón que llama a una función, que se encarga de validar el formulario y enviarlo si todo fue correcto.

La función para validar el formulario se llama ValidaEnvia() que para cada campo del formulario, y utilizando el nombre de cada elemento del mismo, comprueba que el valor introducido es correcto. Si no es correcto, muestra un mensaje de alerta, pone el foco de la aplicación en el campo que ha dado el error y abandona la función retornando el valor 0.

Si todos los campos son correctos, la función continúa hasta el final, sin salirse, por no estar ningún campo incorrecto. Entonces ejecuta la sentencia última, que es el envío del formulario.

Autoevaluación

Pregunta

01/03

JavaScript:

- Es un lenguaje de programación
- Utiliza scripts
- Está orientado a objetos
- Todas las respuestas son incorrectas.

Pregunta

02/03

En JavaScript, los comentarios:

- Pueden comenzar con "://"
- Pueden comenzar con "/*"
- Pueden comenzar con "*/" y terminar con "/*"
- Se utilizan con el elemento "comment"
- Pueden comenzar con "/*" y terminar con "*/"

Pregunta

03/03

JavaScript no es un lenguaje "case sensitive":



Verdadero



Falso