-

UD 01.
NAVEGADORES Y ENTORNO DE TRABAJO

Nombre modulo
CFGS DAW

Nombre autor
mail@ceedcv.es

2019/2020

Versión:210924.1233

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

ÍNDICE DE CONTENIDO

UD01. NAVEGADORES Y ENTORNO DE TRABAJO

1. Navegadores

1.1 Introducción

Para realizar cualquier desarrollo web, es imprescindible comprobar que el resultado que queremos es el adecuado con la mayor cantidad de navegadores posibles, especialmente aquellos más usados.

Además de procesar etiquetas HTML, los navegadores suelen interpretar lenguajes de script, siendo Javascript uno de los más populares.

1.2 Definición de navegador

Un navegador web es un software que permite el acceso a Internet, interpretando la información de distintos tipos de archivos y sitios web para que estos puedan ser visualizados.

La funcionalidad básica de un navegador web es permitir la visualización de documentos de texto, posiblemente con recursos multimedia incrustados. Además, permite visitar páginas web y hacer actividades en ella, es decir, podemos enlazar un sitio con otro, imprimir, enviar y recibir correo, entre otras funcionalidades más.

Los documentos que se muestran en un navegador pueden estar ubicados en la computadora en donde está el usuario, pero también pueden estar en cualquier otro dispositivo que esté conectado en la computadora del usuario o a través de Internet, y que tenga los recursos necesarios para la transmisión de los documentos (un software servidor web).

Tales documentos, comúnmente denominados páginas web, poseen hipervínculos que enlazan una porción de texto o una imagen a otro documento, normalmente relacionado con el texto o la imagen.

El seguimiento de enlaces de una página a otra, ubicada en cualquier computadora conectada a Internet, se llama navegación, de donde se origina el nombre navegador.

Para acceder a estos recursos, se utiliza un identificador único llamado URL (Uniform Resource Locator).

El formato general de una URL es “protocolo://máquina/directorio/archivo”.

- Si no se especifica el directorio, toma como directorio el raíz.
- Si no se especifica el fichero, toma alguno de los nombres por defecto. Usualmente estos nombres por defecto son similares a “index.html” o “index.php”.

> 📣 Un ejemplo muy tipico es https://www.google.es donde se accede al recurso www.google.es usando el protocolo https.

1.3 Funcionamiento de los navegadores

La comunicación entre el servidor web y el navegador se realiza mediante el protocolo HTTP, aunque la mayoría de los navegadores soportan otros protocolos como FTP y HTTPS (una versión cifrada de HTTP basada en Secure Socket Layer o Capa de Conexión Segura (SSL)).

La función principal del navegador es obtener documentos HTML e interpretarlos para mostrarlos en pantalla. En la actualidad, no solamente descargan este tipo de documentos sino que muestran con el documento sus imágenes, sonidos e incluso vídeos streaming en diferentes formatos y protocolos. Además, permiten almacenar la información en el disco o crear marcadores (bookmarks) de las páginas más visitadas.

Algunos de los navegadores web más populares se incluyen en lo que se denomina una Suite. Estas Suite disponen de varios programas integrados para leer noticias de Usenet y correo electrónico mediante los protocolos NNTP, IMAP y POP.

Los primeros navegadores web sólo soportaban una versión muy simple de HTML. El rápido desarrollo de los navegadores web propietarios condujo al desarrollo de dialectos no estándares de HTML y a problemas de interoperabilidad en la web. Los más modernos (como Google Chrome, Mozilla, Netscape, Opera e Internet Explorer / Microsoft Edge) soportan los estándares HTML y XHTML (comenzando con HTML 4.01, los cuales deberían visualizarse de la misma manera en todos ellos).

Los estándares web son un conjunto de recomendaciones dadas por el World Wide Web consortium (W3C) y otras organizaciones internacionales acerca de cómo crear e interpretar documentos basados en la web. Su objetivo es crear una web que trabaje mejor para todos, con sitios accesibles a más personas y que funcionen en cualquier dispositivo de acceso a Internet.

Se puede comprobar de manera online si un documento Web cumple el estándar definido por W3C mediante https://validator.w3.org/

Actualmente la mayoría de navegadores aceptan páginas no estándar, pero cuanto más estándar sea la aplicación web desarrollada, mayor probabilidad que funcione correctamente en todos los navegadores.

Es una práctica imprescindible el comprobar que cualquier desarrollo Web funcione correctamente

en los principales navegadores.

2. Principales Navegadores

- **Microsoft Edge** (Antiguo Internet Explorer)
  - URL Oficial: https://www.microsoft.com/es-es/windows/microsoft-edge
  - Sustituye al antiguo “Internet Explorer”.
  - Microsoft Edge está diseñado para ser un navegador web ligero con un motor de renderizado de código abierto construido en torno a los estándares web.
- **Mozilla Firefox**
  - URL Oficial: https://www.mozilla.org/es-ES/firefox/new/
  - Mozilla Firefox es un navegador web libre y de código abierto desarrollado por la Fundación Mozilla. Usa el motor Gecko para renderizar páginas webs, el cual implementa actuales y futuros estándares web.
    - Posee una versión para desarrolladores: “Firefox Developer Edition” https://www.mozilla.org/es-ES/firefox/developer/
- **Google Chrome**
  - URL Oficial: https://www.google.com/chrome/
  - Google Chrome es un navegador web desarrollado por Google y compilado con baseen varios componentes e infraestructuras de desarrollo de aplicaciones (frameworks) de código abierto, como el motor de renderizado Blink (fork de WebKit). Está disponible gratuitamente bajo condiciones específicas del software privativo.
- Safari
  - URL oficial:http://www.apple.com/es/safari/
  - Safari es un navegador web de código cerrado desarrollado por Apple Inc. Está disponible para Mac OS, iOS (el sistema usado por el iPhone, el iPod touch y iPad) y Windows (sin soporte desde el 2012).
- Opera
  - URL oficial: http://www.opera.com/es
  - Opera es un navegador web creado por la empresa noruega Opera Software. Usa el motor de renderizado Blink. Tiene versiones para escritorio, teléfono y tablet.

2.1 ¿Qué navegador se recomienda?

Por cuota de mercado y sencillez de uso recomiendo utilizarMozilla Firefox oGoogle Chrome.

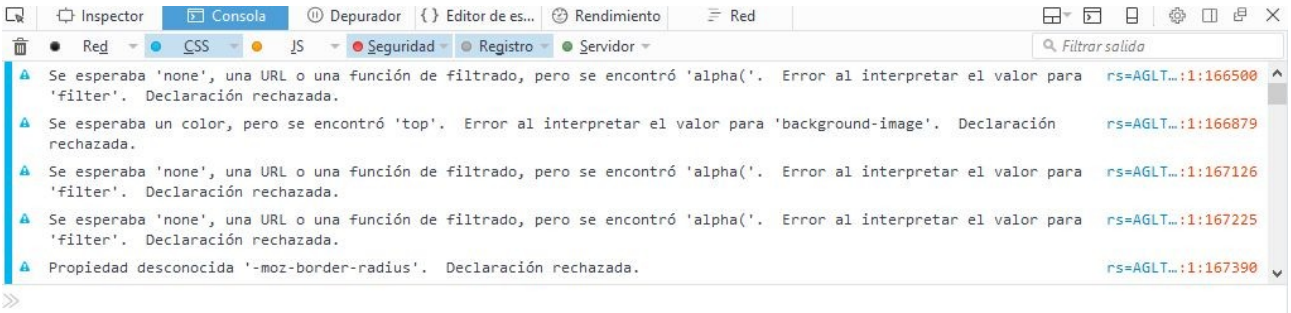
El motivo de usar estos es la gran cantidad de herramientas para depuración que poseen incluso ensu versión estándar. Para la mayoría de acciones con este será suficiente. En el caso de Firefox está disponible una versión que amplía las herramientas de desarrollo “Firefox Developer Edition”.

3. Herramientas útiles para el desarrollo: la consola web

La mayoría de los navegadores incorporan de manera nativa herramientas para facilitar el desarrollo, entra la que destacamos la "Consola Web". Asimismo, también mediante ampliaciones (extensiones, plugins, etc.) se amplían características para facilitar el desarrollo y la depuración de código.

3.1 Consola web

> 📣 En Firefox y Google Chrome, se puede acceder a la consola web pulsando la tecla F12.



Esta consola incluye varias pestañas, que van variando según la versión del navegador. Las pestañas más importantes son:

- Red: registro de Peticiones HTTP.
- CSS: análisis y errores CSS.
- JS: análisis y errores Javascript
- Seguridad: registra advertencias o fallos de seguridad.
- Registro: registra mensajes enviados al objeto "window.console"
- Servidor: registrar mensajes recibidos del servidor Web.

El resultado de las peticiones HTTP se muestra de color negro, CSS de color azul, JavaScript amarillo y los errores o advertencias de seguridad de color rojo, registro objeto "window.console" en gris y Servidor en verde.

> 📣 Mas información del uso de la "Web console" de Mozilla Firefox en la web https://developer.mozilla.org/en-US/docs/Tools/Web_Console

4. Entorno de desarrollo

Existen diversos entornos de desarrollo, desde los más sencillos (Brackets, Notepad++, Sublime, Visual Studio Code, etc...) a interfaces más complejas (Aptana, Eclipse, etc...)

4.1 ¿Qué entorno de desarrollo se recomienda?

Recomendamos el uso de Visual Studio Code. Es muy potente y posee un gran ecosistema de plugins para ampliar su funcionalidad https://code.visualstudio.com/

Aquí algunos manuales libres de uso de Visual Studio Code en castellano:

http://www.mclibre.org/consultar/informatica/lecciones/vsc-instalacion.html

http://www.mclibre.org/consultar/informatica/lecciones/vsc-personalizacion.html

4.2 Control de versiones usando Git + Visual Studio Code

Durante el curso, se utilizarán repositorios Git tanto para la entrega de prácticas como para facilitaros el disponer de un repositorio con control de versiones.

Para instalar Git:

- Ubuntu:
  - `sudo apt-get update`
  - `sudo apt-get install git`
- Windows: https://git-for-windows.github.io/

Para facilitar la tarea del uso de Git es recomendable instalar alguna extensión o entorno que os facilite su uso. Para usar Git en Visual Studio Code recomendamos:

- https://code.visuastudio.com/docs/editor/versioncontrol
- http://www.mclibrle.org/consultar/informatica/lecciones/vsc-git-repositorio.html

Aquí un ejemplo del uso de Git dentro de Visual Studio Code:

https://code.visualstudio.com/docs/introvideos/versioncontrol

5. Material adicional

[1] Curso de Git en Udacity

https://www.udacity.com/course/how-to-use-git-and-github--ud775

[2] Uso de la “Consola Web”

https://developer.mozilla.org/en-US/docs/Tools/Web_Console

Desarrollo Web en Entorno Cliente

UD 01. Sintaxis Javascript ES6

Actualizado Octubre 2020

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

UD01. Sintaxis Javascript ES6

1. Introducción

Javascript es un lenguaje de programación que inicialmente nació como un lenguaje que permitia ejecutar código en nuestro navegador (cliente), ampliando la funcionalidad de nuestros sitios web.

Una de las versiones más extendidas de Javascript moderno, es la llamada por por muchos **Javascript ES6** (ECMAScript 6), también llamado ECMAScript 2015 o incluso por algunos llamado directamente Javascript 6. Al crear esta pequeña guía nos hemos basado en esta versión. Si ya sabes Javascript pero quieres conocer novedades de Javascript ES6, te recomendamos este curso https://didacticode.com/curso/curso-javascript-es6/

Actualmente esa es una de sus principales funciones, pero dada su popularidad el lenguaje ha sido portado a otros ámbitos, entre los que destaca el popular NodeJS https://nodejs.org/ que permite la ejecución de Javascript como como lenguaje escritorio y lenguaje servidor.

Aunque en este módulo nos centramos en la ejecución de Javascript en el navegador, lo aprendido puede ser utilizado para otras implementaciones de Javascript.

2. Hola mundo y comentarios

Para añadir JavaScript se usa la etiqueta SCRIPT.

Este puede estar en cualquier lugar de la página. El código se ejecuta en el lugar donde se encuentra de forma secuencial a como el navegador lo va encontrando.

```
<SCRIPT LANGUAGE="JavaScript">
Aquí va el código
// Esto es un comentario en Javascript de una línea
</SCRIPT>
```

En Javascript los comentarios se pueden hacer con // para una línea y con /* */ para varias líneas.

Asimismo desde Javascript es posible escribir mensajes a la consola de desarrollo mediante la orden “console.log(texto)”.

Este ejemplo podría ser un pequeño hola mundo que al ejecutarse se mostrará en la consola de desarrollo. Ejemplo con “console.log” y comentario multilínea.

```
<SCRIPT LANGUAGE="JavaScript">
console.log ("Hola mundo");
/* Esto es un comentario en
Javascript multilínea */
</SCRIPT>
```

Otra vía para mostrar información al usuario desde una ventana, es el comando “alert(texto)”.

```
<SCRIPT LANGUAGE="JavaScript">
alert("Hola mundo");
</SCRIPT>
```

Hay una forma mucho más práctica y ordenada de usar código Javascript. Se pueden incluir uno o varios ficheros con código Javascript en nuestro documento HTML. Se puede incluir tantos como se desee. Esta es la forma más adecuada para trabajar con código Javascript.

Un ejemplo de inclusión de ficheros.

```
<script type="text/javascript" src="rutaDelArchivo1.js"/>
<script type="text/javascript" src="rutaDelArchivo2.js"/>
<script type="text/javascript" src="rutaDelArchivo3.js"/>
```

3. Variables, constantes y arrays

En este punto hablaremos de variables, constantes y arrays. Antes de empezar, comentar que es recomendable mantener un estilo de nombramiento de variables. Aquí un enlace sobre formas de nombrar variables:

https://medium.com/@alonsus91/convenci%C3%B3n-de-nombres-desde-el-camelcase-hasta-el-kebab-case-787e56d6d023

Durante el curso, usaremos el estilo “CamelCase” https://es.wikipedia.org/wiki/Camel_case

3.1 Ámbito de variables y constantes

Antes de comenzar, vamos a hablar del ámbito de variables (también llamado “scope” en inglés). El ámbito de variables en un lenguaje de programación indica en qué lugares del programa puede ser accedida una variable/constante. Al comentar cada uno de los tipos, definiremos en qué ámbito existen.

3.2 Variables

Las variables son elementos del lenguaje que permiten almacenar distintos valores en cada momento. Se puede almacenar un valor en una variable y consultar este valor posteriormente. También podemos modificar su contenido siempre que queramos.

Para declarar las variables en JavaScript podemos utilizar **let o var,** según el ámbito donde queramos que sea accesible.

- **let**: let permite declarar una variable que sea accesible únicamente dentro del bloque donde se ha declarado (llamamos bloque al espacio delimitado por { }).
- **var**: var permite declarar una variable que sea accesible por todos los lugares de la función donde ha sido declarada. Si una variable con var se declara fuera de cualquier función, el ámbito de esta son todas las funciones del código.

- **Variables sin declarar** : Javascript nos permite usar variables no declaradas. Si hacemos esto, será equivalente a declararlas con **var** fuera del código, es decir, serán variables accesibles por cualquier función.

```
function ejemplo(){
  ejemplo=3; // Equivale a declararla fuera de la funcion como var
  if (ejemplo === 3){
   var variable1 = 1;
   let variable2 = 2;
  }
  console.log(variable1);  // variable1 existe en este lugar
  console.log(variable2); // variable2 no existe en este lugar

}
```

> **Importante:** En general, recomendamos usar **let.** Deberías tener una buena razón para usar **var**, ya que su uso es peligroso ya que podría modificarse una variable desde un lugar que no controles por accidente. Estos problemas no ocurren únicamente en Javascript, sino en otros lenguajes y pueden hacer que tu programa sea complicado de depurar.

3.3 Tipos de datos

Los principales tipos de datos que puede contener variables en Javascript son:

- **Numéricos (tipo "number")** : puede contener cualquier tipo de número real (0.3, 1.7, 2.9) o entero (5, 3, -1).
- **Enteros grandes (tipo "bigint"):** pueden contener enteros con valores superiores a 2^53 -1. Se pueden nomenclar escribiendo una letra "n" al final del entero. No pueden utilizarse con la mayoría de operadores matemáticos de Javascript.
- **Booleanos (tipo "boolean")** : puede contener uno de los siguientes valores: true, false, 1 y 0.
- **Cadenas (tipo "string")** : cualquier combinación de caracteres (letras, números, signos especiales y espacios). Las cadenas se delimitan mediante comillas dobles o simples ("Lolo","lalO"). Para concatenar cadenas puede usarse el operador "+".

El tipo de una variable puede comprobarse usando la estructura **typeof** variable==="tipo".

```
let edad=23, nueva_edad, incremento=4;
const nombre="Rosa García";
console.log(typeof incremento===”number”)
nueva_edad=edad+incremento;
console.log(nombre+ " tras "+incremento +" años tendrá "+ nueva_edad);
```

Asimismo existen otros tipos que Javascript considera primitivos: "undefined", "null", "symbol", "object" y "function".

Todos los valores que NO son de un tipo básico son considerados objetos: arrays, funciones, valores compuestos, etc. Esta distinción es muy importante porque los valores primitivos y los valores objetos se comportan de distinta forma cuando son asignados y cuando son pasados como parámetro a una función.

Más información en https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures

3.4 Coerción

Javascript es un lenguaje de tipado blando (es decir, al declarar una variable no se le asigna un tipo), aunque internamente Javascript si maneje tipos de datos.

En determinados momentos, resulta necesario convertir un valor de un tipo a otro. Esto en JS se llama "coerción", y puede ocurrir de forma implícita o podemos forzarlo de forma explícita.

Por ejemplo

```
let numero = 5;
console.log(numero);
```

En este código, ocurre una coerción **implícita** de número (que es de tipo number) a un tipo string, de modo que puede ser impreso por consola. Podríamos realizar la conversión de forma **explícita** de la siguiente forma:

```
console.log(numero.toString());
```

Las coerciones implícitas ocurren muy a menudo en JS, aunque muchas veces no seamos conscientes de ello. Resulta muy importante entender cómo funcionan para poder deducir cuál será el resultado de una comparación.

```
let a = "2", b = 5;
console.log( typeof a + " " + typeof b); // string number
console.log( a + b ); // nos muestra 15
```

En los lenguajes de tipado duro (por ejemplo, Java) se nos prohíbe realizar operaciones entre distintos tipos de datos. Sin embargo, Javascript, no hace eso, ya que permite operar entre distintos tipos, siguiendo una serie de reglas:

- Javascript tiene el operador === y !=== para realizar comparaciones estrictas, pero no posee esos operadores para desigualdades (>, <, >=, <=).
- Si es posible, JS prefiere hacer coerciones a tipo number por encima de otros tipos básicos. Por ejemplo, la expresión ("15" < 100) se resolverá como true porque JS cambiará "15", de tipo string, por 15 de tipo number.
    - Ten en cuenta que si conviertes "15" a string, al compararlo con "100" la expresión

se resolvería como false.

- A la hora de hacer coerción a boolean, los siguientes valores se convertirán en false: undefined, null, 0, “”, NaN. El resto de valores se convertirán en true.

Ejemplo coerción a number

```
// <, <=, >, <= también hacen coercion. No existe >== ni <==
let arr = [ "1", "10", "100", "1000" ];
for (let i = 0; i < arr.length && arr[i] < 500; i++) {
    console.log(i);
}//0,1,2
```

Ejemplo donde no se hace coerción

```
var x = "10";
var y = "9";
console.log(x < y);// true, los dos son String y los compara como cadena
```

Ejemplo de coerción con undefined

```
let altura; // variable no definida
console.log(altura ? true : false); //Al no estar definido, false
```

Interesante: al realizar comparaciones, si usas == o != para comparar los datos, Javascript realiza coerción. Si quieres que la comparación no convierta tipos y solo sea cierta si son del mismo tipo, debes usar === o !==. Esta es una buena práctica muy recomendada para que estas conversiones no nos jueguen malas pasadas.

Para más información https://www.etnassoft.com/2011/04/06/coercion-de-datos-en-javascript/

3.5 Arrays

Un array (también llamado vector o arreglo) es una variable que contiene diversos valores. Lo creamos simplemente con “[]” o con “new Array()” o inicializando los elementos. Podemos referenciar los elementos de un array indicando su posición.

Los arrays poseen una propiedad llamada “length” que podemos utilizar para conocer el número de elementos del array.

```
// Array definido 1 a 1
let miVector=[];// let miVector=new Array(); es equivalente
miVector[0]=22;
miVector[1]=1
2;
miVector[2]=3
3;
```

```
console.log(miVector1[])
;
console.log(otroArray2[]
);
console.log(miVector +" "+miVector.length );//  array y tamanyo
// Array bidimensional 5x4 declarado sin rellenar
// Para saber mas de map, mirar seccion de funciones flecha
let matrizBi = newArray(5).fill().map(x => newArray(4));
```

3.6 Clonando arrays (y objetos)

Los arrays en Javascript internamente almacenan referencias a donde está alojado cada objeto en memoria, por lo cual copiar un array no es tan simple como hacer miArray=miOtroArray.

En Javascript ES6, se puede hacer una copia sencilla de los valores de un array unidimensional así:

```
let miArray= { ...miOtroArray };
```

Esta copia solo funciona con arrays unidimensionales, ya que con multidimensionales, solo copiará las referencias de memoria de cada uno de estos.

Se pueden hacer copias completas con métodos como este, que se basa en convertir el array a JSON y volver a obtenerlo desde JSON:

```
let miArrayMultidimensional =
JSON.parse(JSON.stringify(miOtroArrayMultidimensional));
```

Más información en https://www.samanthaming.com/tidbits/70-3-ways-to-clone-objects/

3.7 Conversiones entre tipos

Javascript no define explícitamente el tipo de datos de sus variables. Según se almacenen, pueden ser cadenas (Entre Comillas), enteros (sin parte decimal) o decimales (con parte decimal).

Elementos como la función “prompt” para leer de teclado con una ventana emergente del navegador, leen los elementos siempre como cadena. Para estos casos y otros, merece la pena usar funciones de conversión de datos.

```
let num="100"; //Es una cadena
let num2="100.13"; //Es una cadena
let num3=11; // Es un entero
let n=parseInt(num);// Almacena un entero. Si hubiera habido parte
decimal la truncará
let n2=parseFloat(num); // Almacena un decimal
let n3=num3.toString(); // Almacena una cadena
```

3.8 Constantes

Las constantes son elementos que permiten almacenar un valor, pero ese valor una vez almacenado es invariable (permanece constante). Para declarar constantes se utiliza la instrucción "const". Suelen ser útiles para definir datos constantes, como el número PI, el número de Euler, etc…

Su ámbito es el mismo que el de **let**, es decir, solo son accesibles en el bloque que se han declarado.

```
const PI=3.1416;
console.log(PI);
PI=3; // Esto falla
```

Aunque resulta posible definir arrays y objetos usando **const**, no es recomendable hacerlo, ya que es posible que su uso no sea el que pensamos.

Por ejemplo, al declarar un array/objeto, realmente lo que ocurre es que la variable almacena la dirección de memoria del objeto/array. Si lo declaramos usando **const**, lo que haremos es que no pueda cambiarse esa dirección de memoria, pero nos permitirá cambiar sus valores.

```
const miArray=[1,2,3]
console.log(miArray0[]); // Muestra el valor
1 miArray[0]=4;
console.log(miArray0[]); // Muestra el valor
4 miArray=[];// Esto falla
```

3.9 Modo estricto "use strict"

Javascript ES6 incorpora el llamado "modo estricto". Si en algún lugar del código se indica la sentencia "use strict" indica que ese código se ejecutará en modo estricto:

- Escribir "use strict" fuera de cualquier función afecta a todo el código.
- Escribir "use strict" dentro de una función afecta a esa función.

Entre las principales características de "use strict" tenemos que **obliga a que todas las variables sean declaradas**.

"Use strict" es ignorado por versiones anteriores de Javascript, al tomarlo como una simple declaración de cadena.

Las principales carácteristicas de "use strict" son:

- No permite usar variables/objetos no declarados (los objetos son variables).
- No permite eliminar (usando delete) variables/objetos/funciones.
- No permite nombres duplicados de parámetros en funciones.
- No permite escribir en propiedades de objetos definidas como solo lectura.
- Evita que determinadas palabras reservadas sean usadas como variables (eval, arguments,

this, etc…)

Ejemplo 1:

```
"use strict";
pi=3.14;        // Da error
funcionPrueba();

function funcionPrueba() {
  piBIS=3.14;    // Da error
}
```

Ejemplo 2:

```
pi=3.14;        // Da error, use strict sólo se aplica a la función
funcionPrueba();

function funcionPrueba() { "use  strict"; piBIS=3.14;// Da error
}
```

4. Entrada y salida en navegadores

4.1 alert

El método alert() 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.

```
alert("Hola mundo");
```

4.2 console.log

El método console.log permite mostrar información en la consola de desarrollo. En versiones de Javascript de escritorio tipo NodeJS, permite mostrar texto “por pantalla”.

```
console.log("Otro hola mundo");
```

4.3 confirm

A través del método confirm() se activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar. Cuando el usuario pulsa el botón Aceptar, este método devuelve el valor true; Cancelar

devuelve el valor false. Con ayuda de este método el usuario puede decidir sobre preguntas concretas e influir de ese modo directamente en la página.

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

4.4 prompt

El método prompt() 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. Pulsando en Aceptar se obtiene el valor true y la cadena de caracteres introducida se guarda para su posterior procesamiento.

```
let provincia;
provincia=prompt("Introduzca la provincia ","Valencia");
alert("Usted ha introducido la siguiente información "+provincia);
console.log("Operación realizada con éxito");
```

5. Operadores

Combinando variables y valores, se pueden formular expresiones más complejas. Las expresiones son una parte clave en la creación de programas. Para formular expresiones utilizamos los llamados operadores. Pasamos a comentar los principales operadores de Javascript.

5.1 Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables. Algunos de ellos, además de asignar el valor, también incluyen operaciones (Ejemplo += asigna y suma).

Operador	Descripción
=	Asigna a la variable de la parte izquierda el valor de la parte derecha.
+=	Suma los operandos izquierdo y derecho y asigna el resultado al operando izquierdo.
-=	Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo.
*=	Multiplica ambos operandos y asigna el resultado al operando izquierdo.
/=	Divide ambos operandos y asigna el resultado al operando izquierdo.

```
let num1=3;
let num2=5;
num2+=num1;
num2-=num1;
num2*=num1;
num2/=num1;
num2%=num1;
```

5.2 Operadores aritméticos

Los operadores aritméticos se utilizan para realizar cálculos aritméticos.

Operador	Descripción
+	Suma.
-	Resta.
*	Multiplicación.
/	División.
%	Calcula el resto de una división entera.

Además de estos operadores, también existen operadores aritméticos unitarios: incremento (++), disminución (--) y la negación unitaria (-).

Los operadores de incremento y disminución pueden estar tanto delante como detrás de una variable, con distintos matices en su ejecución. Estos operadores aumentan o disminuyen en 1, respectivamente, el valor de una variable.

Operador	Descripción (Suponiendo x=5)
y = ++x	Primero el incremento y después la asignación. x=6, y=6.
y = x++	Primero la asignación y después el incremento. x=6, y=5.
y = --x	Primero el decremento y después la asignación. x=4, y=4.
y = x--	Primero la asignación y después el decremento x=4, y=5.
y =-x	Se asigna a la variable “y” el valor negativo de “x”, pero el valor de la variable “x” no varía. x=5, y= -5.

```
let num1=5, num2=8,resultado1, resultado2;
resultado1=((num1+num2)*200)/100;
resultado2=resultado1%3; resultado1=+
+num1;
resultado2=num2++;
resultado1=--num1;
resultado2=num2--;
resultado1=-resultado2;
```

5.3 Operadores de comparación

Operadores utilizados para comparar dos valores entre sí. Como valor de retorno se obtiene siempre un valor booleano: *true* o *false.*

Operador	Descripción
===	Compara dos elementos, incluyendo su tipo interno. Si son de distinto tipo, no realiza conversión y devuelve false ya que siempre los considera diferentes. **Uso recomendado.**
!==	Compara dos elementos, incluyendo su tipo interno. Si son de distinto tipo, no realiza conversión y devuelve true ya que siempre los considera diferentes. **Uso recomendado.**
==	Devuelve el valor *true* cuando los dos operandos son iguales. Si los elementos son de distintos tipos, realiza una conversión. **No está recomendado su uso.**
!=	Devuelve el valor *true* cuando los dos operandos son distintos. Si los elementos son de distintos tipos, realiza una conversión. **No está recomendado su uso.**
>	Devuelve el valor *true* cuando el operando de la izquierda es mayor que el de la derecha.
<	Devuelve el valor *true* cuando el operando de la derecha es menor que el de la izquierda.
>=	Devuelve el valor *true* cuando el operando de la izquierda es mayor o igual que el de la derecha.
<=	Devuelve el valor *true* cuando el operando de la derecha es menor o igual que el de la izquierda.

```
let a=4;b=5,c="5";
console.log("El resultado de la expresión 'a==c' es igual a "+(a==c));
console.log("El resultado de la expresión 'a===c' es igual a "+(a===c));
console.log("El resultado de la expresión 'a!=c' es igual a "+(a!=c));
console.log("El resultado de la expresión 'a!==c' es igual a "+(a!==c));
console.log("El resultado de la expresión 'a==b' es igual a "+(a==b));
console.log("El resultado de la expresión 'a!=b' es igual a "+(a!=b));
console.log("El resultado de la expresión 'a>b' es igual a "+(a>b));
console.log("El resultado de la expresión 'a<b' es igual a "+(a<b));
console.log("El resultado de la expresión 'a>=b' es igual a "+(a>=b));
console.log("El resultado de la expresión 'a<=b' es igual a "+(a<=b));
```

5.4 Operadores lógicos

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

Operador	Descripción
&&	Y "lógica". El valor de devolución es true cuando ambos operandos son verdaderos.
||	O "lógica". El valor de devolución es true cuando alguno de los operandos es verdadero o lo son los dos.
!	No "lógico". Si el valor es true, devuelve false y si el valor es false, devuelve true.

Se muestra el resultado de distintas operaciones realizadas con operadores lógicos. (En el ejemplo se usa directamente los valore true y false en lugar de variables).

```
console.log("El resultado de la expresión 'false&&false' es igual a
"+(false&&false));
console.log("El resultado de la expresión 'false&&true' es igual a
"+(false&&true));
console.log("El resultado de la expresión 'true&&false' es igual a
"+(true&&false));
console.log("El resultado de la expresión 'true&&true' es igual a
"+(true&&true));
console.log("El resultado de la expresión 'false||false' es igual a
"+(false||false));
console.log("El resultado de la expresión 'false||true' es igual a
"+(false||true));
```

```
console.log("El resultado de la expresión 'true||false' es igual a
"+(true||false));
console.log("El resultado de la expresión 'true||true' es igual a
"+(true||true));
console.log("El resultado de la expresión '!false' es igual a "+(!
false));
```

Para saber más de comparadores y expresiones, puedes consultar:
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

6. Estructuras de control

6.1 Instrucciones if/else

Para controlar el flujo de información en los programas JavaScript existen una serie de estructuras condicionales y bucles que permiten alterar el orden secuencial de ejecución. Estas son las instrucciones if y else.

La instrucción if es permite la ejecución de un bloque u otro de instrucciones en función de una condición.

Sintaxis

```
if (condición) {
// bloque de instrucciones que se ejecutan si la condición se cumple
}
else{
// bloque de instrucciones que se ejecutan si la condición no se cumple
}
```

Las llaves solo son obligatorias cuando haya varias instrucciones seguidas pertenecientes a la ramificación. Si no pones llaves, el if se aplicará únicamente a la siguiente instrucción.

Puede existir una instrucción if que no contenga la parte else. En este caso, se ejecutarán una serie de órdenes si se cumple la condición y si esto no es así, se continuaría con las órdenes que están a continuación del bloque if. Ejemplos de uso:

```
let diaSem;
diaSem=prompt("Introduce el día de la semana ", "");
if (diaSem === "domingo")
{
    console.log("Hoy es festivo");
```

```
}
else // Al no tener {}, es un "bloque de una instrucción"
    console.log("Hoy no es domingo, a descansar!!");
```

Otro ejemplo

```
let edadAna,edadLuis;
// Usamos parseInt para convertir a entero
edadAna=parseInt(prompt("Introduce la edad de Ana",""));
edadLuis=parseInt(prompt("Introduce la edad de Luis",""));

if (edadAna > edadLuis){
    console.log("Ana es mayor que Luis.");
    console.log(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
}
else{
    console.log("Ana es menor o de igual edad que Luis.");
    console.log(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
}
```

6.2 Ramificaciones anidadas

Para las condiciones ramificadas más complicadas, a menudo se utilizan las ramificaciones anidadas. En ellas se definen consultas if dentro de otras consultas if, por ejemplo:

```
let edadAna,edadLuis;
// Convertirmos a entero las cadenas
edadAna=parseInt(prompt("Introduce la edad de Ana",""));
edadLuis=parseInt(prompt("Introduce la edad de Luis",""));
if (edadAna > edadLuis){
    console.log("Ana es mayor que Luis.");
}
else{
    if (edadAna<edadLuis){
        console.log("Ana es menor que Luis.");
    }else{
        console.log("Ana tiene la misma edad que Luis.");
    }
}
console.log(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
```

7. Estructura repetitivas (Bucles)

7.1 Bucle for

Cuando la ejecución de un programa llega a un bucle for:

- Lo primero que hace es ejecutar la “Inicialización del índice”, que solo se ejecuta una vez.
- A continuación analiza la condición de prueba y si esta se cumple ejecuta las instrucciones del bucle.
- Cuando finaliza la ejecución de las instrucciones del bucle se realiza la modificación del índice, se retorna a la cabecera del bucle for y se realiza de nuevo la condición de prueba.
- Si la condición se cumple se ejecutan las instrucciones y si no se cumple la ejecución continúa en las líneas de código que siguen posteriores al bucle.

Sintaxis

```
for (Inicialización del índice; Condición de prueba; Modificación en el índice){
 // ...instrucciones...
}
```

Ejemplo: números pares del 2 al 30

```
for (i=2;i<=30;i+=2) {
    console.log(i);
}
console.log("Se han escrito los números pares del 2 al 30");
```

Ejemplo: Escribe las potencias de 2 hasta 3000

```
let aux=1;
for (i=2;i<=3000;i*=2) {
    console.log("2 elevado a "+aux+" es igual a "+i);
    aux++;
}
console.log("Se han escrito las potencias de 2 menores de 3000");
```

7.2 Bucle while

Con el bucle while se pueden ejecutar un grupo de instrucciones mientras se cumpla una condición.

- Si la condición nunca se cumple, entonces tampoco se ejecuta ninguna instrucción.
- Si la condición se cumple siempre, nos veremos inmersos en el problema de los bucles infinitos, que pueden llegar a colapsar el navegador, o incluso el ordenador.
  - Por esa razón es muy importante que la condición deba dejar de cumplirse en algún momento para evitar bucles infinitos.

Sintaxis

```
while (condición){
//...instrucciones...
}
```

Ejemplo: Escribe los números pares de 0 a 30

```
let i=2;
while (i<=30) {
    console.log(i);
    i+=2;
}
console.log("Ya se han mostrado los números pares del 2 al 30");
```

Ejemplo: Pregunta una clave hasta que se corresponda con una dada.

```
let auxclave="";
while (auxclave!=="vivaYO"){
    auxclave=prompt("introduce la clave ","claveSecreta")
}
console.log("Has acertado la clave");
```

7.3 Bucle do-while

La diferencia del bucle do-while frente al bucle while reside en el momento en que se comprueba la condición: el bucle do-while no la comprueba hasta el final, es decir, después del cuerpo del bucle, lo que significa que el bucle do-while se recorrerá, una vez, como mínimo, aunque no se cumpla la condición.

Sintaxis

```
do {
// ...instrucciones...
} while(condición);
```

Ejemplo: Preguntar por una clave hasta que se introduzca la correcta

```
let auxclave;
do {
    auxclave=prompt("introduce la clave ","vivaYo")
} while (auxclave!=="EstaeslaclaveJEJEJE")
console.log("Has acertado la clave");
```

7.4 Instrucciones BREAK y CONTINUE

En los bucles for, while y do-while se pueden utilizar las instrucciones break y continue para modificar el comportamiento del bucle.

La instrucción "break" dentro de un bucle hace que este 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 final del bucle.

Ejemplo: Pregunta por la clave y permitir tres respuestas incorrectas

```
let auxclave=true;
let numveces=0;
//Mientras no introduzca la clave y no se pulse Cancelar
while (auxclave !== "anonimo" && auxclave){
    auxclave=prompt("Introduce la clave ","");
    numveces++;
    if (numveces === 3)
        break;
}
if (auxclave=="SuperClave"{
 console.log("La clave es correcta");
}else{
 console.log("La clave no es correcta correcta");
}
```

El efecto que tiene la instrucción "continue" en un bucle es el de hacer 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.

Ejemplo: Presenta todos los números pares del 0 al 50 excepto los que sean múltiplos de 3

```
let i;
for (i=2;i<=50;i+=2){
    if ((i%3)===0)
```

```
            continue;
        console.log(i);
}
```

8. Funciones

Una función es un conjunto de instrucciones que se agrupan bajo un nombre de función. Se ejecuta solo cuando es llamada por su nombre en el código del programa. La llamada provoca la ejecución de las órdenes que contiene.

Las funciones son muy importantes por diversos motivos:

- 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.

Una función consta de las siguientes partes básicas:

- Un nombre de función.
- Los parámetros pasados a la función separados por comas y entre paréntesis.
- Las llaves de inicio y final de la función.
- Desde Javascript ES6, se pueden definir valores por defecto para los parámetros.

Sintaxis de la definición de una función

```
function nombrefuncion (parámetro1, parámetro2=valorPorDefecto...){
// instrucciones
//si la función devuelve algún valor añadimos:
return valor;
}
```

Sintaxis de la llamada a una función

```
// La función se ejecuta siempre que se ejecute la sentencia.
valorRetornado=nombrefuncion (parám1, parám2...);
```

Es importante entender la **diferencia entre definir una función y llamarla:**

- Definir una función es simplemente especificar su nombre y definir qué acciones realizará en el momento en que sea invocada, mediante la palabra reservada function.
- Para llamar a una función es necesario especificar su nombre e introducir los parámetros que queremos que utilice. Esta llamada se puede efectuar en una línea de órdenes o bien a la derecha de una sentencia de asignación en el caso de que la función devuelva algún valor debido al uso de la instrucción return.

La definición de una función se puede realizar en cualquier lugar del programa, pero se recomienda hacerlo al principio del código o en un fichero "js" que contenga funciones.

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.

Ejemplo: funciones que devuelve la suma de dos valores que se pasan por parámetros y que escriben el nombre del profesor.

```
// Definiciones de las funciones
function suma (a,b){
    // Esta función devuelve un valor
    return a+b;
}

// Esta función muestra un texto, pero no devuelve un valor
function profe (){
    console.log("El profesor es muy bueno");
    // OJO: esto es un ejemplo, pero rara vez se realiza en una función
real
}
// Código que se ejecuta
let op1=5, op2=25;
let resultado;
// Llamada a función
resultado=suma(op1,op2);
// llamada a la función
console.log (op1+"+"+op2+"="+resultado);
// Llamada a función
profe();
```

> **Atención:** recordad que dentro de las funciones **rara vez se utilizan funciones de entrada/salida**. El 99.9% de las veces simplemente procesan la entrada por parámetros y devuelven un valor.

Desde Javascript ES6, las funciones soportan los llamados parámetros REST.

Los parámetros REST son un conjunto de parámetros que se almacenan como array en un "parámetro final" nomenclado con **...nombreParametro**. Esto nos permite manejar la función sin tener que controlar el número de parámetros con los que esta es llamada.

Importante: sólo el último parámetro puede ser REST.

Ejemplo:

```
function pruebaParREST(a, b, ...masParametros) {
  console.log("a: "+a+" b: "+ b + " otros: " + masParametros);
}
pruebaParREST("param1", "param1", "param3", "param4", "param5");
```

Para saber más de los parámetros REST
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/parametros_rest

9. Funciones flecha (arrow functions)

Una función flecha (arrow function) es una alternativa compacta al uso de funciones tradicionales.

- Este tipo de funciones tienen sus limitaciones y deben ser utilizadas solo en algunos contextos donde sean útiles.
- No son adecuadas para ser utilizadas como métodos.

Soporta varias formas de sintaxis, a continuación indicamos las más tipicas:

- (parametro1, parametro2, ...) => {sentencias}
- () => {sentencias}
- parámetro => sentencia

A veces, son utilizados con la **función "map" de los arrays** . Esta función crea un nuevo array formado por la aplicación de una función a cada uno de sus elementos:
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/map
También es muy útil con la **función "reduce" de los arrays** . Esta función ejecuta una "función reductora" sobre cada elemento del array, devolviendo un único valor.
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/reduce

Ejemplo:

```
let nombres = ['Pedro',  'Juan',  'Elena'];
console.log(nombres.map(nom =>
nom.length));
// Muestra el array con los valores [5, 4, 5]
let sumaNombres= nombres.reduce((acumulador, elemento) => {
  return acumulador + elemento.length;
}, 0);
// Muestra la suma de la longitud de los nombre
```

Para saber más: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Arrow_functions

10. Clases en Javascript

Javascript ES6 permite una mejor definición de clases que en anteriores versiones de Javascript.

Mediante la palabra reservada “class”, permite definir clases, métodos, atributos, etc…

Recordad que los objetos en Javascript se guardan como referencias de memoria. En apartados anteriores hablamos de como clonar arrays y objetos.**R**

Ejemplo:

```
class Punto {
  // Constructor de la clase
  constructor ( pX , pY ){
    this.pX = pX;
    this.pY = pY;
  }
  // Método estático para calcular distancia entre dos puntos
  static distancia ( a , b) {
    const dx = a.pX - b.pX;
    const dy = a.pY - b.pY;

    return Math.sqrt ( dx * dx + dy * dy );
  }
 // Método indicado para ser usado como getter
  get coordX() {
     return this.pX;
   }
  // Método normal
  devuelveXporY () {
    return this.pX * this.pY;
  }
}
let p1 = new Punto(5, 5);
let p2 = new Punto(10, 10);
//Llamada método estático
console.log (Punto.distancia(p1, p2));
//Llamada método normal
console.log (p1.devuelveXporY());
// Al ser un getter, puede usarse como una propiedad
console.log (p1.coordX);
```

Para saber más: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes

11. Más cosas interesantes de Javascript ES6

Este documento pretende ser una simple guía y no podemos incluir todas las novedades de Javascript ES6, pero desde aquí enlazamos las más interesantes por si queréis ampliar:

- Iteradores y generadores
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Iterators_and_Genera tors
- Operadores de propagación (spread)
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Spre ad_operator
- Plantillas de cadenas de texto (template strings)
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/template_strings
- For … of
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/for...o f
- Tail Call Optimization (hace más seguras funciones recursivas)
  - https://hackernoon.com/es6-tail-call-optimization-43f545d2f68b

Aquí tenéis otras referencias a nuevas características de Javascript ES6

https://github.com/lukehoban/es6features

12. Introducción a la manipulación del DOM

Con el fin de facilitar la realización de pequeños ejercicios, incluimos unos primeros conceptos de manipulación del DOM (Document Object Model). Conociendo la id de algún objeto HTML existente en la página podemos cambiar sus propiedades.

Cuando queremos cambiar una propiedad de un objeto presente en el documento, utilizamos su ID (nombre único que define a un elemento) y el método “document.getElementById()” para acceder al elemento en sí.

Ejemplo:

Para cambiar una imagen con id 'matrix', accedemos al elemento y modificamos la propiedad src :

```
document.getElementById('matrix').src = "mt05.jpg";
```

Esto se puede usar para cualquier propiedad existente en cualquier objeto HTML.

Una función Javascript genérica para cambiar una imagen sería:

```
function cambiarImagen (id, rutaImagen) {
    document.getElementById(id).src=rutaImagen;
}
```

13. Material adicional

[1] Curso de Git en Udacity
https://www.udacity.com/course/how-to-use-git-and-github--ud775
[2] Uso de la "Consola Web"
https://developer.mozilla.org/en-US/docs/Tools/Web_Console

14. Bibliografía

[1] Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

[2] Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

15. Autores (en orden alfabético)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- Folgado Galache, Laura
- García Barea, Sergi

Desarrollo Web en Entorno Cliente

UD 01. Objetos predefinidos

Actualizado Octubre 2020

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

UD01. Fundamentos del Lenguaje - OBJETOS PREDEFINIDOS

1 INTRODUCCIÓN

Entre otras posibilidades, Javascript es un lenguaje que de forma nativa posee gran cantidad de funciones y objetos predefinidos. Estas funciones y objetos nos pueden ser útiles para realizar un código más eficiente, claro y ahorrarnos tiempo evitando "reinventar la rueda".

A continuación describiremos algunos de los objetos y funciones predefinidas más importantes.

2 FUNCIONES PREDEFINIDAS

2.1 Tratamiento numérico

Hay dos funciones predefinidas (**parseInt y parseFloat**) que ya comentamos en la unidad anterior. En este punto, además de recordarlas, ampliamos con la función **isNaN** que nos ayudará a distinguir si una cadena es un número o no.

- **parseInt(cadena)**: convierte una cadena de texto entero a entero .
- **parseFloat(cadena)**: convierte una cadena de texto en 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).

Ejemplo:

```
let numero;
do{
     numero=prompt("Esto se repetirá hasta que metas un número");
}while(isNaN(numero));
```

2.2 Función eval

Eval es una función que recibe una cadena y la interpreta como código Javascript. Dado que Javascript admite expresiones numéricas, se puede usar eval para calcular el resultado de expresiones numéricas.

Eval es una función muy útil ya que podemos construir código dinámicamente mediante una cadena. Pero... al ser una función muy poderosa, también es muy peligrosa. Debe utilizarse únicamente en situaciones que lo requiera, ya que una cadena interpretada por eval que esté formada maliciosamente puede causar un agujero de seguridad.

> **Atención:** esta función se utilizaba para procesar JSON (estudiaremos JSON más adelante en el curso). Actualmente por seguridad se utilizan otras funciones para procesar JSON.

Ejemplo:

```
let x=3;
let y=2;
let a=eval("2+3");
let b=eval("x*y");
eval("alert('a vale '+a+' b vale '+b)");
```

3 OBJETO PREDEFINIDO STRING

Atención: aunque Javascript no exige la declaración de tipos de datos, internamente sus variables sí que tienen un tipo de datos concreto.

Cuando creamos una cadena, implícitamente esta cadena se convierte en un objeto String, con sus propiedades y métodos predefinidos.El objeto predefinido String es útil porque nos ayuda a realizar múltiples operaciones con cadenas.

Importante: los métodos de manipulación de cadenas de Javascript, no modifican al objeto actual, sino que devuelven el objeto resultante de aplicar la modificación.

Si queremos que la modificación se aplique sobre la misma cadena que estamos trabajando, haremos algo así:

```
cadena=cadena.metodoQueDevuelveUnaModificacion();
```

En este objeto, hay una propiedad muy importante llamada “length”. Esta propiedad nos indica cuantos elementos (caracteres) tiene la cadena.

También existen una serie de métodos muy útiles.

- **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 solicitada. Debemos tener en cuenta que las posiciones comienzan a contar desde cero.
- **indexOf(texto, [indice])**: devuelve la primera posición donde se encuentra el texto buscado, empezando a buscar desde la posición “indice”. Si “indice” no se indica, se toma por defecto el valor 0.
- **lastIndexOf (texto, [indice])**: como la anterior. Busca “hacia atrás” la primera ocurrencia del texto buscado. Indice indica desde qué punto se empieza a buscar “hacia atrás”. Si no se indica el valor de “indice”, se busca desde el final.
- **replace(texto1,texto2)**: busca ocurrencias de la cadena texto1 y las reemplaza por texto2.
- **split(caracter, [trozos])**: separa la cadena mediante un carácter separador. Trozos indica el máximo de separaciones. Si no se indica, se harán todas las separaciones posibles.
- **substring(inicio, [fin]):** devuelve la subcadena comprendida entre la posición inicio y la posición fin. Si fin no se indica, se toma como valor el final de la cadena.

Ejemplo de algunas de las funciones:

```
let cad="Sergi:Garcia:123456";
let tfo;
cad=cad.toUpperCase(
); alert(cad);
splitTodosCampos=cad.split(":
");
split1Campo=cad.split(":",1);
alert(splitTodosCampos
); alert(split1Campo);
tfo=splitTodosCampos[2];
//Cambio en el telefono los números 3 por 9s
tfo=tfo.replace("2","9
"); alert(tfo);
//Muestro el quinto número del teléfono
alert(tfo.charAt(4));
alert("Bienvenido al CEEDCV");
```

4 OBJETO PREDEFINIDO DATE

Date es un objeto predefinido que nos permite trabajar con fechas. Para crear un objeto date se admiten múltiples formatos. Mas información en http://www.w3schools.com/js/js_dates.asp

Los meses en Date se numeran del 0 al 11 (siendo el 0 Enero y el 11 Diciembre) mientras que los días si se numeran del 1 al 31.

Ejemplo:

```
 // Crea una fecha con la fecha y hora del sistema
let d=new Date();
// Crea una fecha basándose en la cadena de fecha especificada
d=new Date("October 13, 2014 11:13:00");
// Crea una fecha indicando año, mes, dia, horas, minutos,
segundos milisegundos
d=new Date(99,5,24,11,33,30,0);
// Crea una fecha indicando año, mes, día.
d=new Date(99,5,24);
```

Algunos de los métodos más importantes del objeto Date:

- **Métodos 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).
- Ejemplos: setMonth(mes), getMonth(); setDate(dia), getDate(), setHours(hora, minuto,segundo), getHours(), etc.
- **getDay()**: devuelve el día de la semana, (el día del mes es con getDate). Están numerados del 0 al 6, siendo el 0 domingo y el 6 sábado.

- **toDateString()**: convierte la fecha del objeto a una cadena en objeto fecha.
- **toGMTString()**: convierte la fecha del objeto en una cadena con formato de fecha GMT.
- **toUTCString()**: convierte la fecha del objeto en una cadena con formato de fecha UTC.

Ejemplo:

// Con fecha actual y con dia uno del mes, mostramos dia de la semana

```
let d=new Date();
alert(d.getDay());
d.setDate(1);
alert(d.getDay);
```

5 OBJETO PREDEFINIDO ARRAY

En la unidad anterior describimos que es un array en Javascript. Para el uso de arrays, Javascript utiliza un objeto que nos puede ser muy útil para realizar determinadas operaciones.

Aclarar que en el interior de un array se pueden guardar cualquier tipo de objetos (date, string, números enteros, decimales, otros Arrays, etc...).

Todo lo aplicado al uso de arrays que comentamos en el tema anterior, es aplicable al objeto Array (length para número de elementos, modificar/consultar array, etc...).

Importante: generalmente, los métodos del objeto predefinido Array modifican el contenido del propio array.

Algunos de los métodos más importantes del objeto Array:

- **join([separador]):** devuelve una cadena con todos los elementos del array, separados por el texto que se incluya en separador.
- **push(elemento1, elemento2,….):** añade al final los elementos 1 y 2 en el orden proporcionado.
- **pop():** devuelve y elimina el último elemento del array.
- **reverse()**: invierte el orden de los elementos de un array.
- **sort()**: ordena los elementos de un array alfabéticamente. Nota: al ser un orden alfabético, puede dar resultados incorrectos con números, por ejemplo 10 ir antes que 2.
- **slice(inicio, [final])**: devuelve los elementos de un array comprendidos entre inicio y final. Si no se indica final, se toma hasta el último elemento del array.

Ejemplo:

```
let x=[1,2,3];
let y=[2,3,10,2];
// Damos la vuelta a un vector
x.reverse();
alert(x);
// Ordenamos (recordamos que por defecto es alfabético y no numérico)
y.sort()
```

```
alert(y);
// Sacamos y eliminamos elemento
let
sacado=y.pop();
alert(sacado);
```

6 OBJETO PREDEFINIDO 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 más importantes que tiene disponibles son:
- **Redondeo**:
    - **floor(numero)**: redondea hacia abajo un número.
    - **ceil(numero)**: redondea hacia arriba un número.
    - **round(numero)**: redondea al entero mas cercano.
- **Operaciones matemáticas:**
    - **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.

Ejemplo:

```
// Obtenemos un entero entre 1 y 11
let x=parseInt( (Math.random()*10)+1 )
// Redondeamos y hacia abajo
let
y=Math.floor(11.5);
```

7 OBJETO PREDEFINIDO WINDOW

El objeto Window es un objeto que solo está presente al trabajar con Javascript en navegadores, no estando presente en entornos como NodeJS.

El motivo de que este objeto sólo exista en navegadores es que fuera de ellos carece de sentido, ya que posee propiedades y controla elementos relacionado con lo que ocurre en la “ventana” del navegador.

Los métodos que estudiamos en el tema anterior cómo alert, prompt, etc. forman parte del objeto Window. Para hacer llamada a estos métodos no hace falta nombrar explícitamente Window (el navegador ya se encarga de ello).

Algunos de los métodos más importantes no estudiados previamente son:

- **setTimeout(cadenaFuncion,tiempo):** este método ejecuta la llamada a la función proporcionada por la cadena (se puede construir una cadena que lleve parámetros) y la ejecuta pasados los milisegundos que hay en la variable tiempo. Devuelve un identificador del "setTimeout" que nos servirá para referenciar lo si deseamos cancelarlo. SetTimeout solo ejecuta la orden una vez.
- **setInterval(cadenaFunción, tiempo):** exactamente igual que setTimeout, con la salvedad de que no se ejecuta una vez, sino que se repite cíclicamente cada vez que pasa el tiempo proporcionado.
- **clearTimeout / clearInterval (id):** se le pasa el identificador del timeout/interval y lo anula.

Ejemplo:

```
// Creamos un intervalo que cada 15 segundos muestra mensaje hola
let idA=setInterval("alert('hola');",15000);
// Creamos un timeout que cuando pasen 3 segundos muestra mensaje adios
let idB=setTimeout("alert('adios');",3000);
// Creamos un timeout que cuando pasen 5 segundos muestra mensaje estonoseve
let idC=setTimeout("alert('estonoseve');",5000);
// Cancelamos el último timeout
clearTimeout(idC);
```

8 MATERIAL ADICIONAL

[1] Curso de Javascript en Udacity https://www.udacity.com/course/javascript-basics--ud804

9 BIBLIOGRAFÍA

1 Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

2 Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

10 AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- García Barea, Sergi

Desarrollo Web en Entorno Cliente

UD 02.
Manejo del navegador

Actualizado Octubre 2020

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

UD02. MANEJO DEL NAVEGADOR: DOCUMENT OBJECT MODEL (DOM)

1 INTRODUCCIÓN

El llamado DOM (Document Object Model) es un modelo que permite tratar un documento Web XHTML como si fuera XML, navegando por los nodos existentes que forman la página, pudiendo manipular sus atributos e incluso crear nuevos elementos.

Para más información general de DOM:

- https://es.wikipedia.org/wiki/Document_Object_Model
- http://www.w3schools.com/js/js_htmldom.asp

Usando Javascript para navegar en el DOM podemos acceder a todos los elementos XHTML de una página. Esto nos permite cambiar dinámicamente el aspecto de nuestras páginas Web.

En este tema vamos a estudiar las principales funciones de Javascript para modificar el DOM.

2 MODIFICANDO ATRIBUTOS

Cuando obtengamos algún elemento con las funciones que estudiaremos más adelante, podemos manipular los atributos de dicho elemento de la siguiente forma.

```
let elemento=document.getElementById("miElemento");
elemento.innerHTML="El html interno a cambiar de ese elemento";
```

Los atributos disponibles a modificar pueden depender de cada elemento.

3 FUNCIONES DE JAVASCRIPT PARA LOCALIZAR ELEMENTOS EN EL DOM

Las funciones aquí estudiadas normalmente se usan sobre el elemento “document”, ya que así se aplican a todo el documento.

Aun así, pueden usarse en cualquier nodo XHTML, entonces la búsqueda se realizaría no en todo en el documento, sino en el sub-árbol formado por el elemento en sí y sus hijos.

3.1 getElementById(identificador)

Esta función devuelve un elemento DOM del subárbol cuya identificador sea el indicado en la cadena “identificador”.

http://www.w3schools.com/jsref/met_document_getelementbyid.asp

Ejemplo:

```
let myDiv = document.getElementById("miDiv");
console.log("El html de miDiv es "+myDiv.innerHTML);
```

3.2 getElementsByTagName(etiqueta)

Esta función devuelve una array con todos los elementos DOM del subárbol cuya etiqueta XHTML sea la indicada en la cadena "etiqueta".

http://www.w3schools.com/jsref/met_document_getelementsbytagname.asp

Ejemplo:

```
let myDiv = document.getElementById("miDiv")
let losP = myDiv.getElementsByTagName("p");
let num = losP.length;
console.log("Hay " + num + " <p> elementos en el elemento miDiv");
console.log("En el primer P el HTML asociado es "+losP[0].innerHTML);
```

3.3 getElementsByName(nombre)

Esta función devuelve una array con todos los elementos DOM del subárbol cuya atributo name sea el indicado en la cadena "nombre".

http://www.w3schools.com/jsref/met_doc_getelementsbyname.asp

Ejemplo:

```
let elementos = document.getElementsByName("name");
let i;
// Todos los textbox que tengan de name alumnos, los marcamos
for (i = 0; i < elementos.length; i++) {
     if (elementos[i].type === "checkbox") {
           elementos[i].checked = true;
     }
}
```

4 FUNCIONES PARA CREAR/ELIMINAR NODOS

En esta parte veremos las funciones básicas para crear y eliminar nodos XHTML.

4.1 removeChild(nodo)

Esta función se aplica a un nodo padre. La función recibe un nodo hijo suyo y lo borra. Es útil usarlo con el atributo "parentnode", que devuelve el nodo padre del elemento que estamos manejando.

http://www.w3schools.com/jsref/met_node_removechild.asp

Ejemplo:

```
let parrafo=document.getElementById("miParrafo");
// Obtiene la referencia del padre, y al padre le aplica la función removeChild
parrafo.parentnode.removeChild(parrafo);
```

4.2 appendChild(nodo)

Esta función se aplica a un nodo padre. La función recibe un nodo y lo incluye como nodo hijo del padre. Se puede combinar con funciones como "createElement", que permiten crear elementos XHTML.

http://www.w3schools.com/jsref/met_node_appendchild.asp

Ejemplo:

```
// Creo un nodo de tipo LI
let nuevoNodo = document.createElement("LI");
// Al nodo LI le asocio un texto (también podría asociarse XHTML con innerHTML)
let nodoTexto = document.createTextNode("Agua");
nuevoNodo.appendChild(nodoTexto);
// A miLista, lista ya existente, le añado el elemento creado
document.getElementById("miLista").appendChild(nuevoNodo);
```

5 BIBLIOGRAFÍA

1 Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

2 Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

3 Referencia DOM
https://developer.mozilla.org/es/docs/Referencia_DOM_de_Gecko/Introducci%C3%B3n

6 AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- García Barea, Sergi

Desarrollo Web en Entorno Cliente

UD 02.
Manejo del navegador

Actualizado Octubre 2020

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

UD02. MANEJO DEL NAVEGADOR: EVENTOS

1 INTRODUCCIÓN

Los eventos son manejadores que nos proporciona el navegador para que cuando detecte que se produzca una acción (evento), se ejecute un código asociado a esa acción. En esta unidad trataremos los eventos existentes en Javascript y las distintas formas de manejarlos.

2 EVENTOS

2.1 Principales eventos

A continuación mostramos un listado de los principales eventos existentes en Javascript:

- **onfocus**: al obtener un foco.
- **onblur**: al salir del foco de un elemento.
- **onchange**: al hacer un cambio en un elemento.
- **onclick**: al hacer un click en el elemento.
- **ondblclick**: al hacer doble click en un elemento.
- **onkeydown**: al pulsar una tecla (sin soltarla).
- **onkeyup**: al soltar una tecla pulsada.
- **onkeypress**: al pulsar una tecla.
- **onload**: al cargarse una página.
- **onunload**: al descargarse una página (salir de ella).
- **onmousedown**: al hacer clic de ratón (sin soltarlo).
- **onmouseup**: al soltar el botón del ratón previamente pulsado.
- **onmouseover**: al entrar encima de un elemento con el ratón.
- **onmouseout**: al salir de encima de un elemento con el ratón.
- **onsubmit**: al enviar los datos de un formulario.
- **onreset**: al resetear los datos de un formulario.
- **onselect**: al seleccionar un texto.
- **onresize**: al modificar el tamaño de la página del navegador.

> **Importante:** El total de eventos disponibles está descrito en esta página http://www.w3schools.com/jsref/dom_obj_event.as

2.2 Asignando manejadores de eventos desde HTML

La forma más sencilla (aunque menos práctica para tener un código limpio y ordenado) de indicar que hay un evento asociado a un elemento HTML es indicandolo en el propio código.

Ejemplo 1:

```
<input type="button" value="Boton Hola mundo" onclick="alert('Hola mundo');alert('Adios mundo');" />
```

También en lugar de ejecutar una serie de instrucciones, es posible llamar a una función predefinida.

Ejemplo 2:

```
<input type="button" value="Botón miFuncion"
onclick="miFuncion('cadenaParam1');" />
```

2.3 Uso del objeto "this" en la gestión de eventos

Cuando ejecutamos código dentro de un evento, existe la posibilidad de utilizar un objeto llamado "this". Este objeto es una referencia al elemento DOM donde se ha producido el evento. Por ejemplo, si el evento se ha producido al hacer clic a una imagen con id="miImagen", el objeto "this" referencia a esa imagen, es decir, sería equivalente a referenciar el objeto usando document.getElementById("miImagen");

Ejemplo sin "this":

```
<div id="cont" style="width:150px; height:60px; border:thin solid
silver"
onmouseover="document.getElementById('cont').style.borderColor='black';"
onmouseout="document.getElementById('cont').style.borderColor='red';">
  Contenidos
</div>
```

Equivalente con "this":

```
<div id="cont" style="width:150px; height:60px; border:thin solid
silver" onmouseover=this.style.borderColor='black';"
onmouseout="this.style.borderColor='red';">
  Contenidos
</div>
```

2.4 Asignación de eventos desde código a objetos HTML

Es posible asignar y/o modificar mediante código el manejador de un evento predefinido en un documento HTML de una forma similar a esta.

Supongamos que tenemos un objeto dentro del DOM con id="miObjeto" que posee el evento "onclick" sin asignar y la función "mostrarMensaje".

Podemos referenciar al elemento HTML con "document.getElementById" y asignar la función como manejador del evento como vemos en este código

```
function mostrarMensaje(){
      alert("Hola");
}
document.getElementById("miObjeto").onclick=mostrarMensaje;
```

Atención: fijaros que pone "**mostrarMensaje**". Así asigna la función como manejadora del evento. Si ponemos "**mostrarMensaje()**" no funcionará, ya que ejecutará la función y asignará su resultado devuelto al manejador.

Realizar la asignación de eventos de esta forma nos permite facilitar la separación de código e interfaz, ya que se traslada la asignación del manejador del documento HTML al código JS.

Una operación típica dentro del manejo de eventos, es la de comenzar a ejecutar código cuando se haya acabado de cargar una página HTML. Esto ocurre porque si escribimos código Javascript que se ejecute al iniciar la página, pero no nos aseguramos que la página está completamente cargada, es posible que nos de errores (a menudo aleatorios) del estilo de "no encuentro el elemento X del DOM", ya que el código Javascript se ha ejecutado antes de la carga de dicho elemento.

Ejemplo de código que se ejecuta una vez se ha cargado el DOM:

```
// Función que se ejecuta cuando se ha cargado todo el DOM
function inicio(){
  //Código a ejecutar
}
// Asignamos la función inicio al manejador window.onload
// Esto garantiza que el código de "inicio" se ejecute con todo el DOM cargado
window.onload=inicio;
```

2.5 Obteniendo información del objeto event

Cuando se crea una función como manejador, al producirse el evento el navegador automáticamente manda como parámetro un objeto de tipo event.

```
function mostrarMensaje(evento){
      alert(evento.type);
}
document.getElementById("miObjeto").onclick=mostrarMensaje;
```

Este objeto posee cierta información útil del evento que se ha producido.

Sus atributos más destacados son:

- **type**: dice el tipo de evento que es ("click", "mouseover", etc...). Devuelve el nombre del evento tal cual, sin el "on". Es útil para hacer una función que maneje varios eventos.
- **keyCode**: en eventos de teclado, almacena el código de tecla de la tecla afectada por el evento.
- **clientX** / **clientY**: en eventos del ratón, devuelve las coordenadas X e Y donde se encontraba el ratón, tomando como referencia al navegador.
- **screenX** / **screenY**: en eventos del ratón, devuelve las coordenadas X e Y donde se

encontraba el ratón, tomando como referencia la pantalla del ordenador.

Ejemplo:

```
function mostrarMensaje(evento){
      if(evento.type==="keyup"){
            alert(evento.keyCode);
      }
      else if(evento.type==="click"){
            alert(evento.clientX+" "+evento.clientY);
      }
}
document.getElementById("miObjeto").onclick=mostrarMensaje;
document.onkeyup=mostrarMensaje;
```

2.6 Manejadores de eventos desde código

Podemos asignar manejadores de eventos usando **addEventListener(evento,manejador)**.

Ejemplo:

```
function mostrarMensaje(evento){
      if(evento.type==="keyup"){
            alert(evento.keyCode);
      }
      else if(evento.type==="click"){
            alert(evento.clientX+" "+evento.clientY);
      }
}
document.getElementById("miObjeto").addEventListener("click",mostrarMens
aje);
document.addEventListener("keyup",mostrarMensaje);
document.getElementById("miObjeto").addEventListener("dblclick",function
(){
 alert("Codigo metido directamente"); });
```

3 DRAG AND DROP (ARRASTRAR Y SOLTAR)

Además de los eventos típicos tratados, existe un proceso (en el que entran en juego varios eventos) que es útil para el desarrollo de aplicaciones Web, el "Drag and Drop" (Arrastrar y soltar).

En W3Schools podéis obtener teoría y ejemplos de esta técnica https://www.w3schools.com/html/html5_draganddrop.asp

Aquí un ejemplo completo:

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev) {
  ev.preventDefault();
}

function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

4 BIBLIOGRAFÍA

1 Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

2 Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

3 Referencia eventos Javascript http://www.w3schools.com/jsref/dom_obj_event.asp

5 AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- García Barea, Sergi

Desarrollo Web en Entorno Cliente

UD 02. Manejo del navegador

Actualizado Noviembre 2020

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

UD02. MANEJO DEL NAVEGADOR: FORMULARIOS

1. INTRODUCCIÓN

En esta unidad trataremos algunas de las operaciones más habituales en los formularios. En primer lugar veremos cómo acceder a elementos de formularios de distinto tipo. También veremos como validar envíos de formularios a mano, mediante el evento "onsubmit" y el uso sencillo de expresiones regulares.

> **Atención:** al programar una aplicación Web, el cliente debe validar la información introducida, pero ello no quita que el servidor también debe validarla, siendo esta validación incluso más importante que la validación en el cliente.

2. ACCESO A ELEMENTOS DE FORMULARIOS

2.1 text

Para acceder al valor de un input de tipo texto, simplemente debemos referenciar el atributo "value".

Ejemplo:

```
<input type="text" id="miTexto">
```

Javascript asociado:

```
let elemento=document.getElementById("miTexto");
alert(elemento.value);
```

2.2 Radio button

Los "Radio button" son elementos del formulario que ante varias entradas te permiten seleccionar sólo una de ellas. Se agrupan teniendo un "name" común.

Para acceder a ellos, se accede como un array, donde se tiene el atributo "value" y el atributo "checked" que es true si está seleccionado, false en caso contrario.

Ejemplo:

```
<input type="radio" id="preguntaSI" name="pregunta" value="si" />
<label for="preguntaSI">SI</label>
<input type="radio" id="preguntaNO" name="pregunta" value="no" />
<label for="preguntaNO">NO</label>
```

Javascript asociado:

```
let elementos=document.getElementsByName("pregunta");
for(let i=0;i<elementos.length;i++){
     if(elementos[i].checked===true)
          alert("Valor del elemento marcado "+elementos[i].value);
}
```

2.3 checkbox

Similar a los "Radio button", salvo que permite que haya más de un elemento marcado.

Ejemplo:

```
<input type="checkbox" id="preguntaAS" name="pregunta" value="asc" />
<label for="preguntaAS">Piso con ascensor</label>
<input type="checkbox" id="preguntaAM" name="pregunta" value="amb" />
<label for="preguntaAM">Piso amueblado</label>
```

Javascript asociado:

```
let elementos=document.getElementsByName("pregunta");
for(let i=0;i<elementos.length;i++){
     if(elementos[i].checked===true){
          alert("Valor del elemento marcado "+elementos[i].value);
     }
}
```

2.4 select

Elemento que muestra un desplegable y nos permite elegir una opción del mismo. Aquí destaca el atributo "options", que es un atributo que contiene un array con las opciones disponibles y el atributo "selectedIndex" que contiene (y se puede modificar) la posición del array "options" seleccionada actualmente (o la primera si se permite multiselección) o -1 si no está seleccionada ninguna opción (o queremos des-seleccionarlas).

Dentro de cada "options", "value" almacena el valor y "text" el texto mostrado.

Ejemplo:

```
<select id="aprobar" >
     <option value="10">Saco 10 en DWEC</option>
     <option value="9">Saco 9 en DWEC</option>
     <option value="8">Saco 8 en DWEC</option>
</select>
```

Javascript asociado:

```
let elemento=document.getElementById("aprobar");
for(let i=0;i<elemento.options.length;i++){
     alert("Valor de la opción "+elemento.options[i].value);
}
let sel=elemento.selectedIndex;
alert("El valor de la opción seleccionada es
```

```
"+elemento.options[sel].value+" y el texto asociado es
"+elemento.options[sel].text);
// Cambiamos el índice seleccionado
elemento.selectedIndex=0;
```

3. VALIDACIÓN DE FORMULARIOS: EVENTO ONSUBMIT

3.1 Validar un formulario

Si un manejador de un evento devuelve true (o no devuelve nada), se realiza el evento asociado. Si el manejador devuelve false, se cancela el evento.

Existe un evento asociado a un formulario completo llamado “onsubmit”.

Aprovechando esto, podemos a nuestro antojo permitir el envío de información al servidor mediante el formulario devolviendo true o **cancelarlo devolviendo false.**

La estructura típica es la siguiente:

```
<form onsubmit="return validar();">
```

Si la función validar devuelve true, se realiza el envío. Si devuelve false, se cancela.

En la función validar podemos hacer las validaciones que estimemos convenientes.

3.2 Deshabilitar enviar un formulario dos veces

A veces un usuario pulsa enviar un formulario más de una vez por error.

Si queremos evitar esto, podemos usar “this.disabled=true”;

Ejemplo:

```
<form onsubmit="this.disabled=true;">
```

3.3 Enviar un formulario desde código

En algunas aplicaciones por motivos estéticos o de funcionalidad es deseable que el “enviar un formulario” no se haga desde un botón “submit”, sino desde cualquier otro evento que permita la ejecución de código. Esto se puede hacer recogiendo el elemento del formulario y aplicando el método submit().

Ejemplo:

```
<form id="formulario">
```

Javascript asociado:

```
let elemento=document.getElementById("formulario");
elemento.submit();
```

4. VALIDACIÓN DE FORMULARIOS: EXPRESIONES REGULARES

4.1 Expresiones regulares con atributo pattern

Desde HTML5, los elementos de tipo input pueden definir un atributo llamado "pattern" donde definen una expresión regular.

Vemos aquí un ejemplo:

```
<form action="/paginaDestino.php">
  <label for="pwd">Password:</label>
  <input type="password" id="pwd" name="pwd"
  pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
  title="Debe contener al menos un número, una mayúscula y una
minúscula. Además debe contener 8 o más caracteres">
  <input type="submit">
</form>
```

En este ejemplo, si al enviarse el formulario no se cumple el patrón definido, se mostrará el contenido del atributo title.

Podéis ver más ejemplos en https://www.w3schools.com/tags/att_input_pattern.asp

En la web http://html5pattern.com/ podréis obtener una gran cantidad de expresiones regulares ya diseñadas para ser usadas con este atributo.

4.2 Funciones Javascript para el uso de expresiones regulares

Javascript posee dos formar de crear expresiones regulares:

- Literal con expresión regular. Ejemplo var re = /ab+c/;
- El objeto RegExp. Ejemplo var re = new RegExp("ab+c");

Entre distintos métodos, uno de los más usado es "test". El método "test" recibe una cadena y devuelve true si la cadena cumple esa expresión regular, false en caso contrario.

Ejemplo:

```
let re = new RegExp("ab+c");
let cadena=prompt("Dime una cadena");
if(re.test(cadena)){
     alert("La cadena cumple el patrón de una a, entre 1 e infinitas b
y al final una c");
}
```

Sobre el uso de expresiones regulares tenéis más información en

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

Podéis utilizarlas siempre que queráis para cualquier ejercicio.

5. BIBLIOTECAS PARA VALIDACIÓN FORMULARIOS JAVASCRIPT

Algunas de las bibliotecas más populares para la validación de formularios Javascript son:

- joi https://joi.dev/
- Validator.js https://github.com/validatorjs/validator.js
- Validate.js http://rickharrison.github.io/validate.js/
- jQuery Validation Plugin https://jqueryvalidation.org/documentation/

6. BIBLIOGRAFÍA

[1] Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

[2] Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

[3] Expresiones regulares
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

7. AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- García Barea, Sergi