Introducción a los objetos en Javascript

En esta lección vamos a hacer una pequeña introducción a todos los tipos de datos que nos podemos encontrar en este lenguaje de programación. Y por ello, es importante diferenciar los **datos primitivos** de los **objetos en Javascript**.

Los **datos primitivos** hasta la versión ES5 eran cinco:

- Cadena.
- Número.
- Booleano.
- Nulo o *null*.
- No definido o *undefined*.

En la versión ES6 se incorpora un nuevo dato primitivo: symbol (más info)

Pero lo que nos incumbe en esta lección son los objetos, el otro tipo de datos que debemos sumar a los seis anteriores.

Podemos hablar de dos tipos de objetos en Javascript:

- Los **objetos nativos**, que no dependen del navegador.
- Los **objetos de alto nivel**, que dependen del navegador.

Además, no podemos dejar de hablar de los **objetos definidos por el usuario.**

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

```
            <meta charset="UTF-8">
            <title>Objetos. Introducción</title>
      </head>

      <body>
            <script>
            //DATOS PRIMITIVOS:
            var cadena = "Ada Lovelace";
            var pi = 3.14;
            var bool = true; //false
            var nodefinida = undefined;
            var nula = null;

            //OBJETOS NATIVOS: no dependen del navegador.
            /* - String
               - Number
               - Boolean
               - Data
               - Math
               - RegExp (Expresiones regulares)
               - Array
               - Function
               - Object */

            //OBJETOS DE ALTO NIVEL: dependen del navegador.
            /* - Window
               - Screen
               - Navigator
               - Location
               - History
               - Document */

            //OBJETO definido por el usuario:
            var programadora = {nombre: "Ada", apellido: "Lovelace",
anho: 1815};
            /* - Propiedades: se definen en modo nombre:valor. Ej.
nombre, apellido y anho.
               - Métodos: acciones que se pueden ejecutar sobre un
objeto. */
            </script>
      </body>
</html>
```

Objetos. Constructores integrados

Un **constructor integrado** nos permite definir un objeto del tipo que sea. En este caso, vamos a ver cómo definir los siguientes tipos de **objetos nativos**:

- ***String***
- ***Number***
- ***Boolean***
- ***Array***
- ***Function***
- ***Date***
- ***Math***
- ***Object***

En general, todos los **objetos nativos** utilizan la siguiente sintaxis para definirse:

```
new <TipoObjetoNativo>([value]);
```

Hay una pequeña excepción el objeto ***Math*** que es un objeto global y no se puede instanciar con new. simplemente utilizaremos sus métodos y propiedades

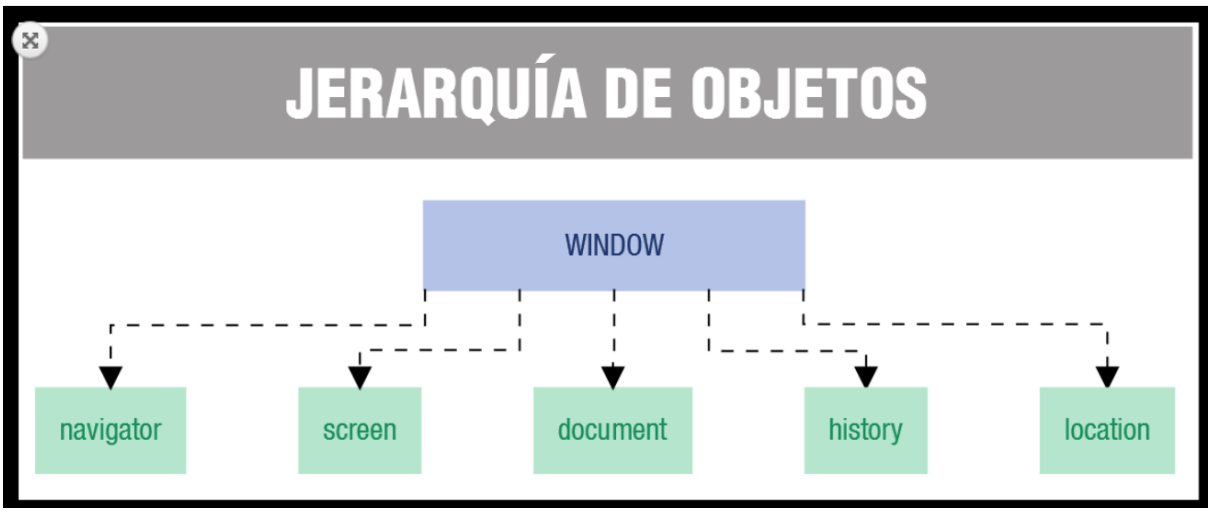
Aunque es una manera totalmente válida de crear objetos, lo cierto es que **no es recomendable** utilizar la palabra **new** para definir números, cadenas, booleanos… Reduce la velocidad de ejecución, complica el código y puede producir resultados inesperados en las comparaciones de objetos

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
```

```
        <title>Objetos. Constructores integrados</title>
    </head>

    <body>
        <script>
            //1. String:
            var x1 = new String(); //No utilizar
            var y1 = "Ada Lovelace"; //Utilizar

            //2. Number:
            var x2 = new Number(); //No utilizar
            var y2 = 3.14; //Utilizar

            //3. Boolean:
            var x3 = new Boolean(); //No utilizar
            var y3 = true; //Utilizar

            //4. Array:
            var x4 = new Array(); //No utilizar
            var y4 = []; //Utilizar

            //5. RegExp:
            var x5 = new RegExp(); //No utilizar
            var y5 = //; //Utilizar

            //6. Function:
            var x6 = new Function(); //No utilizar
            var y6 = function (){}; //Utilizar

            //7. Date:
            var x7 = new Date();

            //8. Math: no se puede declarar con "new"
porque es un objeto global.

            //9. Object:
            var x8 = {};
```

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

Objetos del navegador (BOM). Window I



En esta lección empezamos a ver los **objetos del navegador**. Concretamente empezamos por uno de los más importantes: **window**. El **objeto window** representa la ventana que tiene un documento DOM.

El objeto ***window*** tiene una serie de propiedades:

- ***name***: representa el nombre de la ventana.
- ***outerWidth*** y ***outerHeight***: son el ancho y alto de la ventana incluyendo la barra de herramientas y la de scroll.
- ***innerWidth*** e ***innerHeight***: similar a los anteriores pero sin incluir la barra de herramientas ni la de scroll.
- ***pageXOffset*** y ***pageYOffset***: nos indica dónde se encuentra situado el scroll horizontal y vertical, respectivamente.
- ***screenX*** y ***screenY***: permite conocer la distancia de la ventana desde la izquierda y desde arriba respectivamente.

También hay una serie de **propiedades** con iframes y con otras ventanas que nos van a resultar muy útiles para poder conocer el estado de la ventana, quién la creó, etc. Son:

- ***frames***: devuelve todos los iframe o frames de la ventana.
- ***frameElement***: devuelve el frame en el que está insertada la ventana.
- ***length***: devuelve el número de frames que tiene la ventana.
- ***closed***: devuelve un booleano que indica si la ventana está cerrada.
- ***opener***: devuelve la referencia de la ventana que creó la ventana actual.
- ***parent***: devuelve la ventana padre de la actual.
- ***self***: devuelve la ventana actual

Además, dentro de window se encuentran otros objetos del navegador que veremos más adelante:

- window.document.
- window.navigator.
- window.screen.
- window.history.
- window.location.

Objetos del navegador (BOM). Window II

Sin duda, es muy importante aprender a trabajar con los **métodos del objeto window**:

- ***open***: permite abrir una nueva ventana, pudiendo indicar qué url se va a cargar en ella y qué características va a tener.
- ***close***: en este caso permite cerrar la ventana que ha llamado al método.
- ***resizeBy*** y ***resizeTo***: redimensiona la ventana a un número de píxeles en relación con el tamaño que tiene actualmente, o al número de píxeles indicamos, respectivamente.

- ***moveBy*** y ***moveTo***: al igual que los anteriores, mueve la ventana un número de píxeles respecto a la posición actual, o a una posición indicada, respectivamente.
- ***scrollBy*** y ***scrollTo***: mueve el scroll de la ventana un número de píxeles con respecto al scroll actual o a un número indicado, respectivamente.
- ***focus***: pone el foco en la ventana que llama al método.
- ***print***: imprime la ventana en la que nos encontramos o bien la que llama al método.
- ***stop***: detiene la carga de la página.

```
<!DOCTYPE html>
<html>
     <head>
          <meta charset="UTF-8">
          <title>Objetos del navegador. Window</title>
     </head>

     <body>
          <p id="ventana"></p>
          <script>
               /* CARACTERÍSTICAS:
               - Representa una ventana abierta en un
navegador.
               - Si una ventana tiene etiquetas de tipo
<iframe> el navegador crea un objeto window para el html
inicial y uno para cada <iframe>
               - Todos los objetos, funciones, variables...
son miembros del objeto window: las funciones son sus
métodos; las variables globales y el DOM sos propiedades.
               - No todos los métodos sirven para todos los
navegadores: hay que comprobar la compatibilidad.*/

               //PROPIEDADES BÁSICAS DE WINDOW
               window.name = "Mi ventana";
               var texto = "";
               //Nombre de la ventana
```

```
                texto += "<br/>Nombre: "+window.name;
                //Tamaño de la ventana con toolbar y
scrollbar
                texto += "<br/>Ancho externo:
"+window.outerWidth;
                texto += "<br/>Alto externo:
"+window.outerHeight;
                //Tamaño de la ventana sin toolbar ni
scrollbar
                texto += "<br/>Ancho interno:
"+window.innerWidth;
                texto += "<br/>Alto interno:
"+window.innerHeight;
                //Scroll horizontal y vertical
                texto += "<br/>Scroll horizontal:
"+window.pageXOffset;
                texto += "<br/>Scroll vertical:
"+window.pageYOffset;
                //Distancia de la esquina superior izquierda
                texto += "<br/>Distancia desde la izquierda:
"+window.screenX;
                texto += "<br/>Distancia desde arriba:
"+window.screenY;

                document.getElementById("ventana").innerHTML
= texto;

                /* PROPIEDADES CON IFRAMES:
                -frames: devuelve todos los elementos iframe
de la ventana.
                - frameElement: devuelve el frame en el que
la ventana está insertada.
                - length: devuelve el número de frames que
tiene la ventana. */

                /*PROPIEDADES CON OTRAS VENTANAS:
                - closed: devuelve un booleano si la ventana
```

```
está cerrada.
              - opener: devuelve una referencia con la
ventana que creó la ventana actual.
              - parent: devuelve la ventana padre de la
ventana actual.
              - self: devuelve la ventana actual. */

              /* OTROS OBJETOS DEL NAVEGADOR:
              window.document
              window.navigator
              window.screen
              window.history
              windon.location*/

              //MÉTODOS DE WINDOW MÁS IMPORTANTES
              var miVentana; //Crear fuera de las
funciones para poder acceder a ella

              //open(<URL>,<nombre>,<especificaciones>)
              function crearVentana(){

//miVentana=window.open("http://www.google.com");
                   miVentana =
window.open("","","width=500,height=200");
                   miVentana.document.write("<h1>Mi
ventana</h1>");
              }

              //close(): cierra una ventana en concreto
              function cerrarVentana(){
                   miVentana.close();
              }

              //resizeBy(<nºpix>,<nºpix>): redimensiona
una ventana un número de píxeles respecto a su tamaño
actual
              function redimensionarVentana(){
```

```
                    miVentana.resizeBy(10,10);
               }
               //resizeTo(<nºpix>,<nºpix>): redimensiona
una ventana al número de píxeles indicado

               //moveBy(<nºpix>,<nºpix>): mueve una ventana
un número de píxeles respecto a su posición actual
               function moverVentana(){
                    miVentana.moveBy(10,10);
               }
               //moveTo(<nºpix>,<nºpix>): mueve una ventana
a una posición en concreto

               //scrollBy(<nºpix>,<nºpix>): mueve las
barras de scroll un número de píxeles desde la posición
actual.
               //scrollTo(<nºpix>,<nºpix>): mueve las
barras de scroll un a una posición determinada

               //focus(): pone el foco en la ventana
indicada
               function enfocar(){
                    miVentana.focus();
               }

               //print(): imprime la ventana indicada
               function imprimir(){
                    //print();
                    miVentana.print();
               }

               //stop(): detener la carga de la página

               </script>
               <button onclick="crearVentana()">Crear
ventana</button>
               <button onclick="cerrarVentana()">Cerrar
```

```
ventana</button>
                <button
onclick="redimensionarVentana()">Redimensionar
ventana</button>
                <button onclick="moverVentana()">Mover
ventana</button>
                <button onclick="enfocar()">Enfocar
ventana</button>
                <button onclick="imprimir()">Imprimir
ventana</button>

     </body>
</html>
```

Objetos del navegador (BOM). Window III. Cuadros de diálogo

Los **cuadros de diálogo**, que son tres métodos más del **objeto window** que utilizaremos muchísimo para interactuar con el usuario. Son tres:

- ***alert***: muestra un mensaje a un usuario, pero no recoge ningún valor.
- ***prompt***: muestra un mensaje a un usuario y, además, permite que el usuario introduzca un valor en su interior. Este valor quedará recogido en la variable a la que asignemos el *prompt*.
- ***confirm***: muestra un mensaje al usuario y dos botones. Si el usuario pulsa “Aceptar”, devuelve ***true*** y si pulsa “Cancelar” o cierra la ventana, devuelve ***false***. Tanto true como false se almacenarán en la variable a la que asignemos el *confirm*.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Objetos del navegador. Cuadros de diálogo</title>
    </head>

    <body>
        <script>
```

```
          //alert(<mensaje>): muestra un mensaje al usuario y no devuelve ningún
valor.
          alert("Hola, caracola");

          //prompt(<mensaje>[,<texto por defecto>]): muestra un mensaje al usuario y
un campo que puede contener o no texto por defecto. Lo que introduzca el usuario se
almacenará en una variable a la que se asigna.
          var respuesta = prompt("¿Cuál es tu nombre?", "Introduce un nombre");
          alert ("Hola, "+respuesta);

          //confirm(<mensaje>): muestra un mensaje al usuario y dos botones. Si el
usuario pulsa "Aceptar" devuelve true; si pulsa "Cancelar" o cierra la ventana
devuelve false.
          var respuesta2 = confirm("¿Te gusta javascript?");
          //if (respuesta2==true) alert ("¡Me alegro!");
          //else alert ("¡Pues es una pena!");
          alert((respuesta2) ? "¡Me alegro!" : "¡Pues es una pena!");

        </script>

    </body>
</html>
```

Objetos del navegador (BOM). Window IV. Instrucciones de tiempo

En esta ocasión trabajaremos con una serie de **métodos** muy interesantes que nos permitirán ejecutar código asociado a **instrucciones de tiempo.**

Concretamente hay tres **métodos**:

- ***setTimeout***: en el que indicamos la función que queremos ejecutar y el tiempo que tiene que transcurrir (en milisegundos) antes de que ésta se ejecute.
- ***clearTimeout***: si el método anterior lo asignamos a una variable, a éste podemos pasarle esa variable para detener su ejecución.
- ***setInterval***: con los mismos parámetros que setTimeout, en este caso repite una función cada vez que transcurre el intervalo de tiempo en milisegundos indicado.
- ***clearInterval***: si el método anterior lo asignamos a una variable, a éste podemos pasarle esa variable para detener su ejecución.

Si utilizamos setInterval o setTimeout para iniciar una función es necesario haberlos asociado a una variable si queremos detener su ejecución con clearTimeout

Objetos del navegador (BOM). Navigator

Este objeto, como os podréis imaginar, almacena **información sobre el navegador que está ejecutando nuestra aplicació**n. Esta información puede ser muy valiosa para saber ciertas cosas antes de ejecutar una u otra instrucción, como saber si están activadas las cookies, si estamos ejecutando la aplicación de manera online u offline, cuál es el idioma del navegador, etc.

Concretamente veremos las **propiedades** más importantes, que son:

- ***codeName***: devuelve el nombre código interno del navegador actual.
- ***appName***: devuelve el nombre oficial del navegador.
- ***appVersion***: devuelve la versión del navegador.
- ***product***: casualmente, por cuestiones de compatibilidad, siempre devuelve Gecko.
- ***platform***: devuelve una cadena de texto que representa la plataforma del navegador
- ***onLine***: nos indica si el navegador está actualmente online u offline.
- ***language***: muestra el idioma elegido por el usuario para la interfaz del navegador.
- ***cookieEnabled***: devuelve un booleano que indica si el navegador tiene activadas las cookies.
- ***userAgent***: devuelve la cadena de agente usuario del navegador actual.

- ***geolocation***: devuelve un objeto ***geolocation*** que puede servir para conocer la ubicación del dispositivo que ejecuta la aplicación. Más información

Así como uno de los **métodos** más importantes: ***javaEnabled()***, que nos permite saber si el navegador tiene o no habilitado Java.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Objetos del navegador. Navigator</title>
    </head>

    <body>
       <p id="navegador"></p>
        <script>
        /* CARACTERÍSTICAS:
        - Guarda información sobre el navegador.
        - No hay un estándar público pero la mayoría de los navegadores soportan
este objeto. */

        //PROPIEDADES DEL OBJETO NAVIGATOR:
        var texto = "";
        //appCodeName: codeName del navegador
        texto += "<br/>CodeName: "+navigator.appCodeName;
        //appName: nombre del navegador
        texto += "<br/>Name: "+navigator.appName;
        //appVersion: versión del navegador
        texto += "<br/>Version: "+navigator.appVersion;

        //product: motor del navegador
        texto += "<br/>Motor: "+navigator.product;
        //platform: plataforma del navegador
        texto += "<br/>Plataforma: "+navigator.platform;
        //onLine: navegador online/offline
        texto += "<br/>OnLine: "+navigator.onLine;

        //language: idioma del navegador
        texto += "<br/>Idioma: "+navigator.language;
        //cookieEnabled: cookies activadas
        texto += "<br/>Cookies: "+navigator.cookieEnabled;
        //userAgent: cabecera user-agent devuelta por el navegador al servidor
        texto += "<br/>UserAgent: "+navigator.userAgent;
        document.getElementById("navegador").innerHTML = texto;

        //geolocation: devuelve un objeto geolocation que puede servir para
localizar la posición del usuario.

        //MÉTODO DEL OBJETO NAVIGATOR:
```

```
        //javaEnabled(): devuelve true o false si está activado Java.
        alert("Java activado: "+navigator.javaEnabled());

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

Objetos del navegador (BOM). Screen

representa una **referencia al objeto screen** (pantalla) asociado a la ventana que está siendo visualizada. Y al igual que los objetos anteriores, no es necesario poner ***window.screen***: si utilizamos screen directamente nos estaremos refiriendo a lo mismo.

En esta lección vamos a ver cuáles son las propiedades más importantes de este objeto:

- ***width***: representa el ancho de la pantalla.
- ***height***: representa el alto de la pantalla.
- ***availWidth***: representa el ancho de la pantalla sin la barra de tareas.
- ***availHeight***: representa el alto de la pantalla sin la barra de tareas.
- ***colorDepth***: indica la profundidad de color que estamos utilizando en nuestra pantalla.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Objetos del navegador. Screen</title>
    </head>

    <body>
       <p id="pantalla"></p>
        <script>
           /* CARACTERÍSTICAS:
            - Contiene la información de la ventana del visitante.
            - No hay un estándar público pero la mayoría de los navegadores soportan
Screen */

            //PROPIEDADES DEL OBJETO SCREEN
            var texto = "";

            //Tamaño pantalla
```

```
            texto += "<br/>Ancho: "+screen.width;
            texto += "<br/>Alto: "+screen.height;

            //Tamaño pantalla sin barra de tareas
            texto += "<br/>Ancho sin barra: "+screen.availWidth;
            texto += "<br/>Alto sin barra: "+screen.availHeight;

            //Profundidad de color de la pantalla
            texto += "<br/>Profundidad: "+screen.colorDepth;

            //Resolución de color en bits por píxel
            texto += "<br/>Resolución: "+screen.pixelDepth;

            document.getElementById("pantalla").innerHTML = texto;

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

Objetos del navegador (BOM). History

Con **History** podemos manipular el **historial de sesión del navegador** con facilidad: es decir, podemos consultar las URL de las páginas que ha visitado un usuario en el navegador.

Concretamente vamos a hablar de una propiedad de este objeto:

- ***length***, que como os podéis imaginar, almacena el número de páginas que contiene nuestro historial.

Y, por supuesto, los métodos, que pueden ser de gran utilidad:

- ***back***: carga la URL anterior del historial (es decir, sería como hacer “atrás” en el navegador).
- ***forward***: lo inverso al caso anterior: carga la URL siguiente del historial (si la hay).
- ***go***: va a una página en concreto del historial que pasamos por parámetro.

Como veis, de **History** veremos una propiedad y tres métodos nada más, pero nos van a servir para movernos por el historial utilizando Javascript sin necesidad de interactuar con el propio navegador.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<title>Objetos del navegador. History</title>
	</head>

	<body>
		<script>
			/* CARACTERÍSTICAS:
			- Guarda información de las URLs visitadas por el usuario dentro de una ventana del explorador.
			- No hay un estándar público pero la mayoría de los navegadores soportan este objeto. */

			//PROPIEDADES DEL OBJETO HISTORY
			//length: número de URLs en el historial de la página.
			alert("URLs del historial: "+history.length);

			//MÉTODOS DEL OBJETO HISTORY
			//back: carga la url anterior en el historial
			function atras(){
				history.back();
			}

			//forward: cargar la url siguiente en el historial
			function adelante(){
				history.forward();
			}

			//go(<numero|url>): va a una página concreta del historial. Si indicamos un número positivo va hacia delante un nº de páginas; si indicamos un número negativo va hacia atrás un nº de páginas; si indicamos una url o parte de ella va a la url en concreto.
			function ir(){
				var numero = prompt("Indica un número para moverte en el historial");
				history.go(numero);
			}

		</script>
		<button onclick="atras()">Atrás</button>
		<button onclick="adelante()">Adelante</button>
		<button onclick="ir()">Ir a una página</button>

	</body>
</html>
```

Objetos del navegador (BOM). Location

Podemos decir que **representa la ubicación, o URL**, del objeto al que está vinculado. En otras palabras, básicamente **almacena la información de la URL** (o dirección de Internet) de la página que se está ejecutando en el momento.

Las **propiedades** de **Location** que veremos en esta lección son las siguientes:

- ***href***: que indica cuál es el HREF (URL) de la página y permite cambiarla por otra.
- ***hostname***: almacena el host de la página.
- ***pathname***: almacena la ruta (lo que hay despúes del hostname) de la página.
- ***protocol***: indica cuál es el protocolo de la página (http, https, etc.)
- ***hash***: en caso de que haya un hash o ancla de página, la muestra (Ej. #quienes-somos)
- ***host***: a diferencia de hostname, en este caso almacena el nombre del hostname y el puerto.
- ***origin***: incluye el nombre del protocolo, el hostname y el puerto.
- ***search***: almacena el querystring de la página (Ej. www.web.com/index.html?user=ada).

Además, veremos tres **métodos** del **objeto Location:**

- ***assign(<url>):*** permite asignar un nuevo documento a la página.
- ***reload():*** recarga la página (como si pulsáramos F5 o las flechas circulares).
- ***replace(<url>):*** sustituye una página por otra haciendo desaparecer su historial.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<title>Objetos del navegador. Location</title>
```

```
	</head>

	<body>
		<p id="location"></p>
		<script>
		/* CARACTERÍSTICAS:
		- Guarda información de la URL actual.
		- No hay un estándar público pero la mayoría de los navegadores
soportan este objeto. */

		//PROPIEDADES DEL OBJETO LOCATION
		var texto = "";
		//href: HREF (URL) de la página
		texto += "<br/>Href: "+location.href;
		//hostname: nombre del host de la página
		texto += "<br/>Hostname: "+location.hostname;
		//pathname: pathname de la página
		texto += "<br/>Pathname: "+location.pathname;
		//protocol: protocolo de la página
		texto += "<br/>Protocol: "+location.protocol;
		//hash: hash o ancla de la página (Ej. www.web.com/index.html#indice)
		texto += "<br/>Hash: "+location.hash;

		//host: nombre del hostname y el puerto
		texto += "<br/>Host: "+location.host;
		//origin: nombre del protocolo, hostname y el puerto
		texto += "<br/>Origin: "+location.origin;
		//search: querystring de la página (Ej.
www.web.com/index.html?user=ada)
		texto += "<br/>Search: "+location.search;
		document.getElementById("location").innerHTML = texto;

		//MÉTODOS DEL OBJETO LOCATION
		//assign(<url>): asigna un nuevo documento a la página
		function nuevoDocumento(){
			location.assign("http://www.google.com")
		}

		//reload(): recarga la página
		function recarga(){
			location.reload()
		}

		//replace(<url>): sustituye la página por otra. DESAPARECE SU
HISTORIAL.
		function sustituye(){
			location.replace("http://www.google.com");
		}
		</script>
		<button onclick="nuevoDocumento()">Carga un nuevo documento</button>
		<button onclick="recarga()">Recarga la página</button>
		<button onclick="sustituye()">Sustituye la página</button>
```

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

Objetos nativos. String

Un objeto de tipo **String** representa una serie de caracteres dentro de una cadena. O lo que es lo mismo, en una variable de tipo cadena vamos a poder almacenar palabras, frases… cualquier cosa formada por caracteres alfanuméricos y simbólicos.

En esta lección vamos a ver tres operaciones (llamémoslo así) esenciales en una cadena: la **instanciación**, la **concatenación** y la propiedad ***length***.

Instanciación: o creación de un "ejemplar" de un objeto. Como ya vimos en la lección anterior, podemos hacerlo de dos maneras: o utilizando la palabra ***new*** (no recomendado) o creando **datos primitivos**.

Concatenación: una operación propia de las cadenas que permite **unir dos cadenas de texto** en una.

Propiedades: en el caso de String podemos hablar de ***length***, que nos permite conocer el número de caracteres de la cadena.

Métodos de búsqueda en cadenas:

- ***charAt***
- ***charCodeAt***
- ***fromCharCode***
- ***indexOf***
- ***lastIndexOf***
- ***search***
- ***match***

Y además, el **método de comparación** ***localeCompare***.

También:

- Métodos para saber si una cadena **incluye**, **empieza** o **termina** con otra.
- Métodos para **concatenar** una cadena o **repetirla** tantas veces como necesitemos.
- Métodos de **extracción** de subcadenas dentro de una cadena.
- Métodos para cambiar una cadena a **mayúsculas** o **minúsculas**.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<title>Objetos nativos. String</title>
	</head>

	<body>
		<script>
		//INSTANCIACIÓN:
		var daw = "Desarrollo de aplicaciones web";
		var dam = 'Desarrollo de aplicaciones multiplataforma';
		var asir = "Administración de 'Sistemas Informáticos' en Red";
		var smr = "Sistemas \"Microinformáticos\" y Redes";
		var ciclos = new String("");

		//CONCATENACIÓN DE CADENAS
		ciclos += "Hay 3 ciclos de Grado Superior: \n";
		ciclos += daw + ", " + dam + " y " + asir + "\n";
		ciclos += " y 1 ciclo de Grado Medio: \n";
		ciclos += smr;

		alert (ciclos);

		//PROPIEDADES
		//length: devuelve la longitud de una cadena
		alert ("La longitud de la cadena ciclos es: "+ciclos.length);

		//MÉTODOS:
		//BÚSQUEDA:
		//CharAt(<num>): devuelve el caracter de una posición.
		alert ("El caracter 16 de la cadena ciclos es: "+ciclos.charAt(16));
		//indexOf(<caracter>): devuelve la primera posición de un caracter en
una cadena.
		alert ("La primera aparición de la letra a en ciclos es:
"+ciclos.indexOf("a"));
```

```
            //lastIndexOf(<caracter>): devuelve la última posición de un caracter
en una cadena.
            alert ("La última aparición de la letra a en ciclos es:
"+ciclos.lastIndexOf("a"));
            //search(<cadena|expresión>): buscar una cadena o expresión regular en
otra cadena.
            alert ("Busca la cadena 'web' en la variable daw:
"+daw.search("web"));
            //match (<expresión regular>): busca una expresión regular en otra
cadena.

            //COMPARACIÓN:
            //localeCompare(<cadena>): devuelve -1 (antes), 0 (igual), 1 (después)
            alert ("¿Es daw menor que dam? "+ daw.localeCompare(dam));

            //INCLUYE, EMPIEZA O TERMINA (Version ES6)
            //includes(<cadena>): devuelve true si la cadena incluye el parámetro.
            alert ("¿Incluye 'aplicaciones' daw? "+daw.includes("aplicaciones"));
            //startsWith(<cadena>): devuelve true si la cadena empieza con el
parámetro.
            alert ("Empieza daw con la palabra 'aplicaciones'?
"+daw.startsWith("aplicaciones"));
            //endsWith(<cadena>): devuelve true si la cadena finaliza con el
parámetro.
            alert ("Acaba daw con la palabra 'aplicaciones'?
"+daw.endsWith("aplicaciones"));

            //CONCATENACIÓN Y REPETICIÓN
            //concat(<cadena>): concatena a la cadena el parámetro:
            alert ("Concatena daw con dam \n"+daw.concat(dam));
            //repeat(<número>): repetir la cadena el número de veces que aparece
como parámetro
            alert ("Repetir daw \n"+daw.repeat(3));

            //EXTRACCION
            //slice(<posicion inicial>,<posicion final>): devuelve la cadena
comprendida entre ambas posiciones:
            alert ("La cadena que hay entre el 0 y el 1 en daw es:
"+daw.slice(0,1));
            //substring(<posicion inicial>,<posicion final>): devuelve la cadena
comprendida entre ambas posiciones:
            alert ("La cadena que hay entre el 0 y el 1 en daw es:
"+daw.substring(0,1));
            //substr(<posicion inicial>,<número de caracteres>): devuelve la
cadena comprendida entre la posición inicial y el número de caracteres:
            alert ("La cadena de 5 caracteres que hay a partir del caracter 7 es:
"+daw.substr(5,7));
            //split(<caracter>,[<número de veces]): divide la cadena en un array
de subcadenas separadas por el caracter del primer parámetro
            alert("La cadena daw separada por espacios es: "+daw.split(" ",2));
            //trim(<cadena>): extrae los caracteres de la cadena eliminando los
espacios del principio y del final.
            alert ("La cadena sin espacios quedaría: \n"+"            Hola,
```

```
caracola               ".trim());

              //MÁYÚSCULAS Y MINÚSCULAS
              //toLowerCase(): devuelve la cadena en minúsculas:
              alert (daw.toLowerCase());
              //toUpperCase(): devuelve la cadena en mayúsculas:
              alert (daw.toUpperCase());

              //MÉTODOS ESPECIALES:
              //toString: devuelve el valor del objeto String.
              //valueOf: devuelve el valor primitivo del objeto.

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

Template strings. Plantillas de cadena de texto

Este tipo de plantillas literales se delimitan con el **carácter de comillas o tildes invertidas** (acento grave (` `), en lugar de las habituales comillas sencillas o dobles.

Su uso más habitual es que pueden contener **marcadores o expresiones incrustadas**, que se identifican por el símbolo del dólar y a continuación la expresión entre llaves (`${expresión}`) y **cadenas de texto de más de una línea**.

Las distintas **sintaxis** básicas de las **template strings** son las siguiente:

```
`cadena de texto` //Literal
`línea 1 de la cadena de texto
 línea 2 de la cadena de texto` //Cadena de texto de más de una línea
`cadena de texto ${expresión} texto` //Expresiones
```

En resumen, las ***template strings*** son literales de texto que habilitan el uso de:

- Expresiones incrustadas (${expresion})
- Cadenas de texto de más de una línea.
- Interpolación expresiones.

> *Para escapar una comilla o tilde invertida en una plantilla literal (o template string) de #Javascript debes poner delante una barra invertida (\).*

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>ES6 - Plantillas de cadena de texto / Template
strings</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1">
  <script src="14_ES6_TemplateStrings.js"></script>
</head>
<body>
    <h1>ES6 - Plantillas de cadena de texto / Template
strings</h1>
</body>
</html>
```

```
/*
Las plantillas de cadena de texto (template strings) son literales
de texto que habilitan el uso de:
- Expresiones incrustadas (${expresion})
- Cadenas de texto de más de una línea.
- Interpolación expresiones.
- Plantillas con una función de preprocesado (las veremos más
adelante).
- NOTA: para escapar una comilla dentro de una plantilla se usa \
*/

//SINTAXIS
/*
```

```
`cadena de texto`

`línea 1 de la cadena de texto
 línea 2 de la cadena de texto`

`cadena de texto ${expresión} texto`

*/

//Cadenas de más de una línea
//ES5
console.log("Hola, \n\
caracola");
//ES6
console.log(`Agur,
yogur`);

//Interpolar expresiones
let num1 = 2;
let num2 = 4;
//ES5
console.log("La suma es " + (num1 + num2) + "\ny la resta " +
(num1 - num2) + ".");
//ES6
console.log(`La suma es ${num1 + num2}
y la resta ${num1 - num2}.`);
```

Objetos nativos. Number

Los objetos nativos **Number** permiten trabajar con valores numéricos.

A diferencia de otros lenguajes de programación, en Javascript **solo hay un tipo de datos numérico**. Es decir, no hablamos de números enteros, dobles, en coma flotante… **¡todos los números se pueden representar con objetos de tipo *Number*!**

En esta lección vamos a ver varias operaciones con Number muy útiles:

- Cómo **cambiar de base entre números**: pasar a binario, octal o hexadecimal.
- Cómo representar el **-infinito** y el **+infinito**… ¡sí, hay dos infinitos diferentes!
- Cómo **comprobar si un dato es un número** o no.
- Cómo **representar un número como un objeto** y como un **dato primitivo**.
- Conoceremos las propiedades del objeto: **MAX_VALUE**, **MIN_VALUE**, **NEGATIVE_INFINITY**, **POSITIVE_INFINITY** y, por supuesto, **NaN**.
- Aprenderemos cómo definir el número máximo de cifras y de decimales con .**toFixed** y .**toPrecision** y cómo representar en exponencial con .**toExponential**.
- Veremos cómo devolver un valor primitivo y una cadena a partir de un número con .**valueOf** y .**toString** y cómo devolver el valor numérico de una variable con el propio **Number**().
- Y cómo no, uno de los métodos estrella que **utilizaremos cientos de veces** al capturar un número por teclado: el método que nos permite parsear una cadena a entero, .**parseInt** (o a decimal, con .**parseFloat**).

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

```
	<meta charset="UTF-8">
	<title>Objetos nativos. Number</title>
</head>

<body>
	<script>
		/* CARACTERÍSTICAS:
		- Solo hay un tipo de datos numérico.
		- Podemos representar: 34, 34.05, 123e4, 123e-4, 0xFF */

		//CAMBIO DE BASE ENTRE NÚMEROS: toString()
		var base10 = 128;
		alert (base10+" en base 2 es "+base10.toString(2));
		alert (base10+" en base 8 es "+base10.toString(8));
		alert (base10+" en base 16 es "+base10.toString(16));

		//INFINITO Y -INFINITO: Infinity, -Infinity
		var inf = 2;
		while (inf != Infinity){
			alert (inf);
			inf = inf*inf; //inf *= inf;
		}

		var div1 = 2, div2 = 0;
		alert ("División 2/0="+2/0);
		alert ("División -2/0="+(-2/0));
		alert ("Typeof: "+(div1/div2));

		//NOT A NUMBER: NaN
		var atipico = 100/"casa";
		var atipico2 = 100/"10";
		alert ("100/'casa'="+atipico);
		alert ("100/'10'="+atipico2);
		alert ("¿Es un NaN atipico? "+isNaN(atipico));
		alert ("¿De qué tipo es NaN? "+typeof(atipico));

		//NÚMEROS COMO OBJETOS
		var num = 123;
		var num2 = new Number(123);
		var num3 = new Number(123);
		alert ("num: "+typeof(num)); //Devuelve number
		alert ("num2: "+typeof(num2)); //Deuelve object
		alert ("num == num2: "+(num==num2));
		alert ("num === num2: "+(num===num2));
		alert ("num2 === num3: "+(num3===num2));

		//PROPIEDADES
		var maximo = Number.MAX_VALUE; alert("Maximo="+maximo);
		var minimo = Number.MIN_VALUE; alert("Mínimo="+minimo);
		var neginf = Number.NEGATIVE_INFINITY; alert("-Inf="+neginf);
		var posinf = Number.POSITIVE_INFINITY; alert("+Inf="+posinf);
		var numnan = Number.NaN; alert ("NaN="+numnan);
```

```
                    //MÉTODOS:
                    //toFixed(<num decimales>): cadena con el número específico de
decimales indicado:
                    var x = 9.8765;
                    alert (x.toFixed(0));
                    alert (x.toFixed(2));
                    alert (x.toFixed(6));

                    //toPrecision(<num cifras>): cadena con el número específico de
cifras indicado:
                    var y = 9.8765;
                    alert (y.toPrecision());
                    alert (y.toPrecision(2));
                    alert (y.toPrecision(6));

                    //toExponential: cadena con el número redondeado a notación
científica.
                    var z = 123456789;
                    alert (z.toExponential(2));

                    //valueOf: devuelve el valor primitivo de un objeto.
                    //toString: devuelve la cadena de un número.
                    var j = 123;
                    alert(j.toString());
                    alert((123).toString());
                    alert((123+7).toString());

                    //MÉTODOS GLOBALES PARA CONVERTIR VARIABLES EN NÚMEROS:
                    //Number(): devuelve el valor numérico de una variable:
                    alert (Number(true)); // Devuelve 1
                    alert (Number(false)); // Devuelve 0
                    alert (Number(new Date())); // Devuelve el número de
milisegundos desde el día 1/1/1970
                    alert (Number("10")); // Devuelve el número de la cadena
                    alert (Number("casa")); // Devuelve NaN

                    //parseInt() / parseFloat():
                    alert (parseInt("10.6")); //Devuelve el número
                    alert(parseInt("10 20")); //Devuelve el primer número
                    alert(parseInt("10 casa")); //Devuelve el primer número
                    alert(parseInt("casa 10")); //Devuelve NaN

            </script>

        </body>
</html>
```

Objetos nativos. Math

El **objeto nativo Math** tiene propiedades y métodos para **constantes y funciones matemáticas.**

Math.PI, que contiene el número Pi.

El metodo *Math.random()*, que genera números aleatorios;

Math.max() junto con *Math.min()* que seleccionan el mayor o menor de una lista de valores.

En el siguiente ejemplo vemos también otras propiedades y metodos.

```
<!DOCTYPE html>
<html>
     <head>
          <meta charset="UTF-8">
          <title>Objetos nativos. Math</title>
     </head>

     <body>
          <script>
               /* CARACTERÍSTICAS:
               - Permite realizar operaciones matemáticas.
               - No es un constructor: NO usamos var x =
new Math();
               - Ej: var pi = Math.PI; var raiz =
Math.sqrt(36); */

               //PROPIEDADES:
               // E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2,
SQRT2
               var pi = Math.PI;
               alert ("El número PI vale: "+pi);
```

```
            //MÉTODOS MÁS COMUNES:
            //random(): Devuelve un número aleatorio
entre 0 y 1
            alert ("Número aleatorio: "+Math.random());

            //max(<lista de valores>) // min (<lista de
valores>)
            alert ("Máximo de valores:
"+Math.max(1,5,3,9,6));
            alert ("Mínimo de valores:
"+Math.min(1,5,3,9,6));

            //round(<número>): redondeo al alza o a la
baja
            var a = 4.3;
            var b = 4.7;
            var c = 4.5;
            alert ("Redondeo de "+c+" es:
"+Math.round(c));

            //ceil(<número>): redondea siempre al alza:
            alert ("Redondeo de "+a+" es:
"+Math.ceil(a));

            //floor(<número>): redondea siempre a la
baja:
            alert ("Redondeo de "+b+" es:
"+Math.floor(b));

            //Número aleatorio entre 0 y 10:
            var aleatorio =
Math.floor(Math.random()*11);
            alert ("Aleatorio es: "+aleatorio);

            //pow(<base>,<exponente>): devuelve el valor
de la potencia
```

```
                alert ("La potencia de 3 al cuadrado es:
"+Math.pow(3,2));

                //sqrt(<numero>): devuelve la raíz del
número:
                alert("Raíz de 36 es: "+Math.sqrt(36));

                //abs(<número>): devuelve el valor absoluto
de un número
                alert ("Valor absoluto de -5:
"+Math.abs(-5));

                //Métodos trigonmétricos:
                //sin, cos, tan, asin, acos, atan, atan2

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

Objetos nativos. Date

El **objeto nativo Date** se utiliza para trabajar con fechas está formado por **milisegundos desde el día 1 de enero de 1970**

En primer lugar, hay que entender todos los modos de crear un objeto date. Aquí tenéis varios modos:

```
new Date()
new Date(milisegundos)
new Date(cadenaFecha)
new Date(año_num,mes_num,dia_num [,hor_num,min_num,seg_num,mils_num])
```

más ejemplos

Claro está, dependiendo de lo que introduzcamos entre paréntesis podemos obtener objetos con la fecha actual, con fechas elegidas por nosotros, con fechas formateadas a nuestro gusto…

Así que en esta lección veremos las siguientes cuestiones:

- Cómo crear un objeto date con la fecha actual.
- Crear fechas con cadenas como parámetro.
- Crear fechas con milisegundos como parámetro.
- Crear fechas con números como parámetros.
- Visualizar fechas con diferentes formatos: toString, toUTCString, toDateString.
- Métodos de tipo get: getDay, getDate, getMonth, getFullYear, getHours, getMinutes, getSeconds, getMilliseconds, getTime.
- Métodos de tipo set: setDay, setDate, setMonth, setFullYear, setHours, setMinutes, setSeconds, setMilliseconds, setTime.

> *Cuando utilizamos getDay sobre un objeto fecha no obtenemos el día del mes, sino el día de la semana, siendo el 0 el domingo.*

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<title>Objetos nativos. Date </title>
	</head>

	<body>
		<script>
			/* CREACIÓN DE OBJETOS DE TIPO DATE.
			Nota: los meses comienzan en 0. Los días de la semana también,
siendo el 0 el domingo, el 1 el lunes... */

			//Creación de fecha actual:
			var actual = new Date();
			alert(actual);

			//Creación de fecha con cadenas:
			var d1 = new Date("Wed Mar 25 2015 09:23:45 GMT +0100 (W.
Europe Standard Time)"); //Ignora si el día de la semana está mal o los paréntesis
```

```
                    var d2 = new Date("October 12, 2016 10:30:00");
                    var d3 = new Date("January 25 2015");
                    var d4 = new Date("Jan 25 2015"); //También "25 Jan 2015"
                    //Podemos utilizar comas porque se ignoran
                    var d5 = new Date("2016-05-12T12:34:25");
                    var d6 = new Date("2016-05-12"); // YYYY-MM-DD
                    var d7 = new Date("2016/05/12"); //YYYY/MM/DD o DD/MM/YYYY
                    var d8 = new Date("2016-05"); //El día se sustiye por 1
                    var d9 = new Date("2016"); //El día y el mes se sustituyen por
1
                    //¡OJO! Incluimos 0 en días/meses inferiores a 10
                    alert (d1);
                    alert (d2);
                    alert (d3);
                    alert (d4);
                    alert (d5);
                    alert (d6);
                    alert (d7);
                    alert (d8);
                    alert (d9);

                    //Creación de fecha con milisegundos
                    var dms = new Date(86400000);
                    alert ("Fecha en ms: "+dms);

                    //Creación de fechas con números
                    var fechaLargo = new Date(2015,11,10,14,30,25);
                    var fechaCorto = new Date(2015,11,10);
                    alert ("Fecha largo: "+fechaLargo+ " Fecha corto:
"+fechaCorto);

                    var fecha = new Date();
                    alert ("Fecha: "+fecha);
                    alert ("toString: "+fecha.toString());
                    alert ("toUTCString: "+fecha.toUTCString());
                    alert ("toDateString: "+fecha.toDateString());

                    //MÉTODOS GET
                    var fc = new Date("October 1, 2016 10:30:20");
                    alert(fc.getDay()+"
"+fc.getDate()+"/"+fc.getMonth()+"/"+fc.getFullYear());

alert(fc.getHours()+":"+fc.getMinutes()+":"+fc.getSeconds()+":"+fc.getMilliseconds()
);
                    alert("Ms desde 1/1/1970: "+fc.getTime());

                    //MÉTODOS SET
                    fc.setDate(31);
                    fc.setMonth(11);
                    fc.setFullYear(2031);
                    fc.setHours(23);
                    fc.setMinutes(59);
                    fc.setSeconds(59);
```

```
                    fc.setMilliseconds(59);
                    //fc.setTime(<ms>); //Modificaríamos la fecha y la hora
contando los milisegundos desde el 1/1/1970
                    alert(fc.toString());

                    //Suma 3 meses a la fecha creada previamente fc
                    fc.setMonth(fc.getMonth()+3);
                    alert(fc.toString());

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

Objetos nativos. Boolean

Los **datos primitivos de tipo boolean** permiten almacenar un valor verdadero (***true***) o falso (***false***).

```
<!DOCTYPE html>
<html>
      <head>
            <meta charset="UTF-8">
            <title>Objetos nativos. Boolean</title>
      </head>

      <body>
            <script>
            //Función Boolean()
            alert (10>9);

            //Todos los valores reales son ciertos
            alert("100 es "+Boolean(100));
            alert("3.4 es "+Boolean(3.4));
            alert("Hola es "+Boolean("Hola"));
            alert("false "+Boolean("false"));
            alert("-15 es "+Boolean(-15));
            alert("5<6 es "+Boolean(5<6));

            //Todos los valores que no son reales son falsos
            alert("0 es "+Boolean(0));
            alert("-0 es "+Boolean(-0));
            alert("undefined es "+Boolean(undefined));
            alert("null es "+Boolean(null));
            alert("false es "+Boolean(false));
            alert("NaN es "+Boolean(NaN));
```

```
            var g;
            if (!g) // Es equivalente a g==undefined
            {
                  alert ("La g no está definida");
            }

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