# TEMA 3. Utilización de los objetos predefinidos de JavaScript

## Índice




**Autor: Roberto Marín**


**Asignatura: Desarrollo Web Entorno Cliente (DAW2)**

**Año: 2021/2022**

# 3.1. Objeto nativo Date y Math

https://www.w3schools.com/js/js_dates.asp

**Empleo del objeto Date: método getDate(), getMonth(), getFullYear(), con 3 casos diferentes.**

```
   Caso a: con numeros:
    <script>
        let f = new Date();
        document.write(f.getDate() + "/" + (f.getMonth() + 1) + "/" + f.getFullYear() + "<br>");
    </script>

    Caso b: con los nombres:

    <script>
        let meses = new Array("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "A
gosto", "Septiembre", "Octubre", "Noviembre", "Diciembre");
        let f = new Date();
        document.write(f.getDate() + " de " + meses[f.getMonth()] + " de " + f.getFullYear() + "
<br>");
    </script>

    Caso c: Dia de la semana:
    <script>
        let diasSemana = new Array("Domingo", "Lunes", "Martes", "Miércoles", "Jueves", "Viernes
", "Sábado");
        let f = new Date();
        document.write(diasSemana[f.getDay()] + "<br>");
    </script>
```

https://www.w3schools.com/js/js_math.asp

https://www.w3schools.com/js/js_random.asp

### Empleo del objeto Math: .exp, .pow y .random

```
//Math.exp(10) Explicación: Devuelve el número E elevado a 10
document.write(Math.exp(10));
document.write("<br>");
//Math.pow(2,10)) Explicación Devuelve 2 elevado a 10
document.write(Math.pow(2,10));
document.write("<br>");
//Ejercicio: Haz un script que calcules dos números de forma aleatoria, por ejemplo a y b. Luego
 opera “a” elevado a “b”.
var a = Math.round(Math.random()*100000000);
var b = Math.round(Math.random()*10);
var resultado= Math.pow(a,b);
document.write(resultado);
document.write("<br>");
```

### Empleo del objeto Math: diferencias entre .floor y .round

```
//diferencias entre Math.floor y Math.round.
document.write(Math.floor(Math.random()*10)); //Redondeo hacia abajo
document.write("<br>");
document.write(Math.round(Math.random()*10)); //Redondeo hacia arriba
document.write("<br>");
```

Ejercicio: Haz un script que muestre la fecha actual y por consola saca el mes actual. Haz una comparación y si es Octubre muestra por pantalla la hora actual.

```
//Ejercicio: Haz un script que muestre la fecha actual y por consola saca el mes actual. Haz una
 comparación y si es Octubre muestra por pantalla la hora actual.
let fecha_actual = new Date();
let  mes= fecha_actual.getMonth()+1;
console.log(mes);
if (mes==10){
document.write(fecha_actual);
document.write("<br>");
}else{
console.log("Como no es Octubre No hago nada");
}
```

Ejercicio: Haz un script que calcule el área de un círculo según un radio introducido por teclado por el usuario. Debes emplear para su cálculo el objeto nativo Math de JavaScript. Ayuda: Área de un círculo es igual a π por el radio al cuadrado.

```
let  r = prompt("Ingresa el radio de un c\xedrculo:");
let  area = Math.PI * Math.pow(r, 2);
alert("El \xe1rea del c\xedrculo es de: " + area + " cms cuadrados");
```

Ejercicio: Crea un script para visualizar un reloj digital. Crea un objeto fecha de tipo Date() para poder calcular la fecha actual y que se actualice permanentemente. Emplear setInterval para actualizar el reloj cada segundo. Mejorar el ejercicio mostrando únicamente las horas, minutos y segundos.

```
let  myVar = setInterval(myTimer, 1000);

function myTimer() {
    let  d = new Date();
    let  t = d.toLocaleTimeString(); //Devuelve el valor de la hora en un string,
    document.getElementById("demo").innerHTML = t;
}
```

# 3.2. Objeto nativo Number

Objeto que representa un número de cualquier tipo.

Propiedades y métodos del objeto Number:
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Number

https://www.w3schools.com/js/js_number_methods.asp

**Ejercicio:** Emplea un método del objeto Number para sacar 3 decimales del número PI. Ayuda: Emplea el objeto PI.

```
<script type="text/javascript">
        alert("Propiedad MAX_VALUE: " + Number.MAX_VALUE)
        alert("Propiedad MIN_VALUE: " + Number.MIN_VALUE)
        let maximoSeguroDeOperar = Number.MAX_SAFE_INTEGER;
        let maximoSeguroDeOperar1 = Number.MAX_SAFE_INTEGER + 1;
        let maximoSeguroDeOperar2 = Number.MAX_SAFE_INTEGER + 2;
        console.log("Maximo Total:" + maximoSeguroDeOperar); // En número máximo seguro es:
9007199254740991
        console.log("Maximo Total:" + maximoSeguroDeOperar1);
        console.log("Maximo Total:" + maximoSeguroDeOperar2); // si le sumamos 2 al número
máximo seguro, nos sigue dando lo mismo. CUIDADO! Mejor usar BIGINT
        alert("Propiedad NaN: " + Number.NaN)
        alert("Propiedad NEGATIVE_INFINITY: " + Number.NEGATIVE_INFINITY)
        alert("Propiedad POSITIVE_INFINITY: " + Number.POSITIVE_INFINITY)
        var N1 = new Number(3.141592653589793);
        alert("Letra Pi formateada: " + N1.toPrecision(3))
    </script>
```

**Solución:** desde ECMAScript 2020 se introducen un nuevo tipo de dato BigInt.
Los BigInit se crean con el constructor BigInt o con la sintaxis 10n, donde "n" se coloca después del número literal. BigInts permite la representación y manipulación de enteros mayores que Number.MAX_SAFE_INTEGER

NOTA1: Las operaciones con los tipos de dato BigInt no son compatibles con el objeto Math. Por lo tanto habrá que utilizar los operadores visto en la Unidad Didáctica 2, en vez de las propiedades del objeto Math.
Ejemplo: Usar ** con los BigInt para potencias en vez de Math.pow

NOTA2: No se pueden mezclar operaciones entre números BigInt y “Enteros ó Decimales”.

Ejemplos de empleo de tipos de datos BigInt: https://es.javascript.info/bigint

# 3.3. Objeto nativo String

Las cadenas de texto en Javascript son consideradas como objetos de la clase String, esto quiere decir que tienen sus propiedades, y sus métodos.
La unica propiedad que tiene definida el objeto String es la propiedad length, la cual ya hemos visto en páginas anteriores, y que nos indica el número de caracteres que tiene la cadena de texto.

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String

El objeto String es el que más métodos o funciones tiene, a continuación se ven algunos de los más habituales. Recordar que la sintaxis para aplicar el método a una cadena de texto es la habitual de Javascript, se escribe primero la variable que contiene el texto seguida de un punto y después el método.

- toString(): Convierte cualquier elemento en una cadena de texto, por lo que en realidad no se aplica a cadenas de texto, sino a cualquier variable.

- concat(): junta dos cadenas, de modo que a la cadena texto1 se le suma la cadena texto2 que pasamos como parámetro en el paréntesis. El resultado es una nueva cadena, exactamente igual que si las sumaramos: nuevoTexto = texto1 + texto2.

- toUpperCase(): Crea la misma cadena de texto, pero con todos los caracteres en letras mayúsculas.

- toLowerCase(): Crea la misma cadena de texto, pero con todos los caracteres en letras minúsculas.

- indexOf(): Dentro del paréntesis pasamos como parámetro un caracter. La función lo busca dentro de la cadena, y devuelve un número que indica la posición del carácter encontrado. Si hay más de un caracter da la posición del primero. Hay que tener en cuenta que se empieza a contar desde 0, por lo que si está en primer lugar, el valor devuelto será 0. Si no se encuentra el carácter, devuelve -1. Podemos pasar como parámetro más de un caracter (una palabra por ejemplo). Se buscará si la cadena indicada está dentro del texto y nos dirá en qué posición empieza.

- lastIndexOf(): Es igual que el anterior, pero se empieza a buscar desde el final de la cadena. Sin embargo el número que devuelve es su posición empezando a contar desde el principio de la cadena por el 0.

- substring(): Devuelve una porción del texto delimitada por los números que se pasan como parámetros. El primer número indica la posición en la que está el principio de la cadena que devuelve. El segundo indica el final y es opcional; si no se indica la cadena devuelta irá desde la posición del primer número hasta el final; Si se indica irá hasta el carácter inmediatamente anterior; por lo que el carácter indicado en la segunda posición no está incluido en la cadena resultante.

- split(): Convierte la cadena de texto en un array. Dentro del paréntesis escribiremos el caracter que separa los elementos del array. Ejemplo: si queremos separarlo en palabras pondremos el espacio en blanco split(" "), o si queremos que cada elemento del array sea una letra escribiremos split(""). Si no se pasa ningún argumento el caracter por defecto es la coma.

- replace(buscarTexto, textoNuevo): Este método reemplaza una porción de texto por otra. Para ello el primer parámetro (buscarTexto) que le pasamos es el trozo de texto que hay que reemplazar. El segundo parámetro (textoNuevo) es el texto por el cual va a ser reemplazado el primero.

- charAt(): Este método devuelve el carácter que se encuentra en la posición indicada por el número que se le pasa como parámetro. Las posiciones se empiezan a contar desde el 0.

- charCodeAt(): Este método devuelve el código Unicode del carácter que se encuentra en la posición indicada por el número que se le pasa como parámetro. Las posiciones se empiezan a contar desde el 0.

**Ejercicio:** Emplea los métodos de String para realizar las siguientes operaciones con la siguiente cadena de texto: “Numero de letras de esta cadena”.

- Calcula la longitud de la cadena.
- Sacar el carácter del índice en la posición 5
- Sacar la posición de la primera letra “s”
- Sacar la posición de la ultima letra “e”
- Buscar en la cadena “letras” y cambiar por “caracteres”

```
<script>
let  texto = new String("Numero de letras de esta cadena");
document.write("N\xfamero de letras de la cadena de texto: " + texto.length + "<br>");
document.write("el carácter del índice en la posición 4: " + texto.charAt(4) + "<br>");
document.write("La posición de la primera letra “s”: " + texto.indexOf("r") + "<br>");
document.write("La posición de la ultima letra “e”: " + texto.lastIndexOf("e") + "<br>";
document.write("Cambiar letras x caracteres: " + texto.replace("letras", "caracteres")+"<br>");
</script>
```

# 3.4. Objeto nativo Array

Conjunto de datos ordenados por posiciones y todos asociados en una sola variable. Los datos pueden ser de cualquier tipo de dato y además se crear una array que tenga una cadena en la primera posición, un número en la segunda ó un objeto en la tercera.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

**Ejercicio:** Empleando el siguiente Array y con los métodos propios, calcula:

- La longitud del array.
- Sacar por pantalla la posición 4.
- Quita el último elemento de la matriz y se muestra por pantalla.
- Insertar dos nuevos elementos al principio de la matriz.

Array ("uno", "dos", "tres", “cuatro”, “cinco”, “seis”)

```
<script>
let arr = new Array('one', 'two', 'tres', 'cuatro', 'cinco', 'seis');
//let arr = ['one', 'two', 'tres', 'cuatro', 'cinco', 'seis'];
document.write(arr.length + "<br/>");
document.write(arr[3] + "<br/>");
document.write(arr.pop() + "<br/>");
document.write(arr.push("nuevo1", "nuevo2"));
</script>
```

# 3.5. Objeto nativo Boolean

Propiedades del objeto Boolean:

Constructor: Devuelve la función que creó el objeto Boolean.
Prototype: Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Boolean:

toString(): Convierte un valor Boolean a una cadena y devuelve el resultado.
valueOf(): Devuelve el valor primitivo de un objeto Boolean.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Boolean

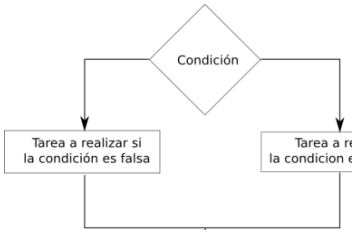
```
//Si especificas cualquier objeto, incluido un objeto Boolean cuyo valor es false, como valor
inicial de un objeto Boolean,
//el nuevo objeto Boolean tiene un valor de true.

let myFalse = new Boolean(false);   // valor inicial de false
let g = Boolean(myFalse);           // valor inicial de true
let myString = new String('Hola');  // objeto string
let s = Boolean(myString);          // valor inicial de true
```

# 3.6. Estructuras de control

## 3.6.1. Estructuras condicionales

Permite realizar bifurcaciones en el flujo de las instrucciones. Llegados a un punto, se establece un condición que permitirá ejecutar un tipo de instrucciones u otras. Funciona de la misma manera que en el resto de lenguajes de programación.




### a) If-else

```
let edad = prompt("Dime tu edad");
if (edad >= 65) {
alert('Eres mayor de 65 años');}
else if (edad>= 18){
alert('Eres mayor de 18 años y menor de 65 años');}
else {
alert('Tienes menos de 18 años')}
```

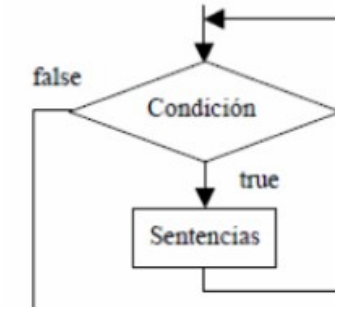
### b) Switch-case

Con la idea de minizar la estructura compleja de varios if else, se puede usar un switch-case. Se ejecuta una istrucción y depende del resultado de la misma se realizará una acción u otra.

```
let mes = new Date().getMonth();
switch(mes){
    case 0: console.log("Enero"); break;
    case 1: console.log("Febrero"); break;
    case 2: console.log("Marzo"); break;
    case 3:console.log("Abril"); break;
    case 4:console.log("Mayo"); break;
    case 5:console.log("Junio"); break;
    case 6: console.log("Julio"); break;
    case 7: console.log("Agosto"); break;
    case 8: console.log("Septiembre"); break;
    case 9: console.log("Octubre"); break;
    case 10: console.log("Noviembre"); break;
    case 11: console.log("Diciembre"); break;
    case 12:
    case 13:
    case 14: console.log("Esto es una prueba"); break;
    default: console.log("Ha fallado el objeto Date")}
```

# 3.6.2. Estructuras de repetición

Este tipo de estructuras de control se llaman también bucles y se emplean para que de manera repetitiva se pueda ejecutar varias veces una instrucción. Funciona de la misma manera que en el resto de lenguajes de programación.




### a) For

```
for (a=0; a<10; a++)
{console.log(a);
if (a==7) {break};}
```

```
let a=0;
for (;;)
{console.log(a);
if (a>=7) {break};
a++;}
```

### b) While & Do-While

Se ejecuta mientras se cumple la condición.

```
let a = 0;
while (a<5) {
alert (a);
a++;
}
alert('Sale de la condición cuando a vale:' +a);
do {
alert('Estoy dentro while y hago si o si el do')
} while (a !==5);
```

Nota: En bucle do-while es similar al while, pero la condición se comprueba al final. Se emplea mucho para comprobar las opciones de un menu, se ejecutará hasta que el usuario no elija “salir del programa”. Se necesita comprobar la opción que el usuario a elegido al final del programa, ya que al principio aún no sabemos la opción que ha escogido.

### c) For in

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/for...in

Se recorren las claves-propiedades dentro del array

```
let alumno = ["Alberto","Lopez",25];
for( let i in alumno){
console.log(i);
}
```

Se podría solucionar el problema iterando sobre la clave del Array

```
let alumno = ["Alberto","Lopez",25];
for( let i in alumno){
console.log(alumno[i]);
}
```

Nota 1: No es una buena praxis iterar sobre las propiedades de un array empleando for in. Porque pueden verse añadidas nuevas claves del array y al listar esas propiedades, además de los índices numéricos se listarán dichas propiedades (Array.prototype)

Nota 2: Se complementará la información en la Unidad Didactica 4.

### d) For of

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

Se iteran sobre los elementos del Array.

```
let alumno = ["Juan","Perez",25];
for( let i of alumno){
console.log(i);
}
```

Nota 1: Se puede usar break, continue y return
Nota 2: Se puede usar no solo en Arrays, el resto de objetos, como ejemplo Strings.
Nota 3: Corrige los inconvenientes del forEach, en el cual no hay forma de detener o cortar el bucle).

# 3.7. Objetos de alto nivel

## 3.7.1. Navigator

https://developer.mozilla.org/es/docs/Web/API/Navigator

| Propiedades | Descripción |
|---|---|
| appCodeName | Cadena que contiene el nombre del código del cliente. |
| appName | Cadena que contiene el nombre del cliente. |
| appVersion. | Cadena que contiene información sobre la versión del cliente. |
| cookieEnabled | Determina si las cookies están o no habilitadas en el navegador. |
| platform | Cadena con la plataforma sobre la que se está ejecutando el programa cliente. |
| userAgent | Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appCodeName y appVersion. |
| plugins | Array que contiene todos los plug-ins soportados por el cliente. A partir de NS 3. |

**Ejercicio:** Empleando los objetos propios del navegador, comprueba:

- Si tiene cabecera en la petición de HTTP.
- Muestra por pantalla el nombre y la versión de vuestro navegador.
- Determina si las cookies están habilitadas

```
<script>
document.write("Navegador: " + navigator.cookieEnabled + "<br>");
document.write("Navegador: " + navigator.appName + "<br>");
document.write("Versi\xf3n: " + navigator.appVersion + "<br>");
if (navigator.userAgent) {
document.write("El navegador SI est\xe1 preparado para mostrar la cabecera de la peticion HTTP"
)
} else {
document.write("El navegador NO est\xe1 preparado para mostrar la cabecera de la peticion HTTP"
)
}
</script>
```

# 3.7.2. Screen

**https://developer.mozilla.org/es/docs/Web/API/Screen**

| Propiedades | Descripción |
|---|---|
| availHeight | Devuelve la altura de la pantalla, incluyendo la barra de herramientas |
| availWidth | Devuelve la anchura de la pantalla, incluyendo la barra de herramientas |
| colorDepth | Devuelve la profundidad de paleta de colores para mostrar imágenes. |
| height | Devuelve la altura de la pantalla |
| pixelDepth | Devuelve la resolución de la pantalla (bits/pixel) |
| width | Devuelve la anchura de la pantalla |

**Ejercicio:** Empleando los objetos propios del navegador, comprueba:

- Altura total y disponible de la pantalla.
- Anchura total de la pantalla.
- Anchura disponible de la pantalla.
- Profundidad de color de la pantalla.

```
<script>
document.write("Altura total: " + screen.height + "<br>");
document.write("Altura disponible: " + screen.availHeight + "<br>");
document.write("Anchura total: " + screen.width + "<br>");
document.write("Anchura disponible: " + screen.availWidth + "<br>");
document.write("Profundidad de color: " + screen.colorDepth + "<br>");
</script>
```

# 3.7.3. History

https://developer.mozilla.org/es/docs/DOM/Manipulando_el_historial_del_navegador
(Se ampliarán los conceptos en el tema 5 en la sección del árbol DOM)

| Propiedades | Descripción |
|---|---|
| current | Cadena que contiene la URL completa de la entrada actual en el historial. |
| length | Número de entradas del historial (cuántas direcciones han sido visitadas). |
| next | Cadena que contiene la URL completa de la siguiente entrada en el historial. |
| previous | Cadena que contiene la URL completa de la anterior entrada en el historial. |

| **Método** | **Descripción** |
|---|---|
| back() | Vuelve a cargar la URL del documento anterior dentro del historial. |
| forward() | Vuelve a cargar la URL del documento siguiente dentro del historial |
| go(posicion) | Vuelve a cargar la URL del documento especificado por posición dentro del historial. Posición puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial; o puede ser una cadena de caracteres, en cuyo caso representa toda o parte de una URL que esté en el historial. |

**Ejercicio:** Empleando los objetos propios del navegador, crea dos botones que:

- Permita retroceder a la pantalla anterior de navegación
- Permita avanzar la pantalla de navegación.
- Calcula el número de páginas que has visitado.

```
<body>
    <form>
        <input type="button" value="Atras" onClick="history.back();">
        <input type="button" value="Adelante" onClick="history.forward();">
    </form>
    <script>
        document.write("El numero de entradas: " + window.history.length + "<br>");
    </script>
</body>
```

# 3.7.4. Location

https://developer.mozilla.org/es/docs/Web/API/Location

| Propiedades | Descripción |
|---|---|
| hash | Cadena que contiene el nombre del enlace, dentro de la URL. |
| hostname | Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL. |
| pathname | Cadena que contiene el camino al recurso, dentro de la URL. |
| protocol | Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL. |
| host | Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL. |
| href | Cadena que contiene la URL completa. |
| port | Cadena que contiene el número de puerto del servidor, dentro de la URL. |
| search | Cadena que contiene la información pasada en una llamada a un script, dentro de la URL. |

| **Método** | **Descripción** |
|---|---|
| reload() | Vuelve a cargar la URL especificada en la propiedad href del objeto location |
| replace(cadenaURL) | Reemplaza el historial actual mientras carga la URL especificada en cadenaURL |
| assign() | Carga un nuevo documento. |

**Ejercicio:** Empleando los objetos propios del navegador, calcula:

- La URL de navegación
- Protocolo empleado
- El nombre del domino del servidor

```
<body>
    <script>
        document.write("URL completa: " + location.href + "<br>");
        document.write("Pathname: " + location.pathname + "<br>");
        document.write("Protocolo: " + location.protocol + "<br>");
    </script>
    <form>
        <input type="button" value="Recargar" onClick="location.reload();">
    </form>
</body>
```

# 3.7.5. Window

En la jerarquía de objetos, tenemos en la parte superior el objeto window.

Este objeto está situado justamente ahí, porque es el contenedor principal de todo el contenido que se visualiza en el navegador. Tan pronto como se abre una ventana (window) en el navegador, incluso aunque no se cargue ningún documento en ella, este objeto window ya estará definido en memoria.

Además de la sección de contenido del objeto window, que es justamente dónde se cargarán los documentos, el campo de influencia de este objeto, abarca también las dimensiones de la ventana, así como todo lo que rodea al área de contenido: las barras de desplazamiento, barra de herramientas, barra de estado, etc.

La jerarquía de objetos, debajo del objeto window tenemos otros objetos como el navigator, screen, history, location y el objeto document. Este objeto document será el que contendrá toda la jerarquía de objetos, que tengamos dentro de nuestra página HTML.

Se muestra la jerarquía de objetos que cuelgan del objeto window: navigator, screen, document, location y history.
Atención En los navegadores más modernos, los usuarios tienen la posibilidad de abrir las páginas tanto en nuevas pestañas dentro de un navegador, como en nuevas ventanas de navegador. Para JavaScript tanto las ventanas de navegador, como las pestañas, son ambos objetos window.

Acceso a propiedades y métodos.

Para acceder a las propiedades y métodos del objeto window, lo podremos hacer de diferentes formas, dependiendo más de nuestro estilo, que de requerimientos sintácticos. Así, la forma más lógica y común de realizar esa referencia, incluiría el objeto window tal y como se muestra en este ejemplo:

```
window.nombrePropiedad
window.nombreMétodo( [parámetros] )
```

Como puedes ver, los parámetros van entre corchetes, indicando que son opcionales y que dependerán del método al que estemos llamando.

Un objeto window también se podrá referenciar mediante la palabra self, cuando estamos haciendo la referencia desde el propio documento contenido en esa ventana:

```
self.nombrePropiedad
self.nombreMétodo( [parámetros] )
```

Podremos usar cualquiera de las dos referencias anteriores, pero intentaremos dejar la palabra reservada self, para scripts más complejos en los que tengamos múltiples marcos y ventanas.

Debido a que el objeto window siempre estará presente cuando ejecutemos nuestros scripts, podremos omitirlo, en referencias a los objetos dentro de esa ventana. Así que, si escribimos:

```
nombrePropiedad
nombreMétodo( [parámetros] )
```

También funcionaría sin ningún problema, porque se asume que esas propiedades o métodos, son del objeto de mayor jerarquía (el objeto window) en el cuál nos encontramos.

Gestión de ventanas.

Un script no creará nunca la ventana principal de un navegador. Es el usuario, quien realiza esa tarea abriendo una URL en el navegador o un archivo desde el menú de abrir. Pero sin embargo, un script que esté ejecutándose en una de las ventanas principales del navegador, si que podrá crear o abrir nuevas sub-ventanas.

El método que genera una nueva ventana es window.open(). Este método contiene hasta tres parámetros, que definen las características de la nueva ventana: la URL del documento a abrir, el nombre de esa ventana y su apariencia física (tamaño, color,etc.).

Por ejemplo, si consideramos la siguiente instrucción que abre una nueva ventana de un tamaño determinado y con el contenido de un documento HTML:

```
let subVentana = window.open("nueva.html", "nueva", "height=800, width=600");
```

Lo importante de esa instrucción, es la asignación que hemos hecho en la variable subVentana. De esta forma podremos a lo largo de nuestro código, referenciar a la nueva ventana desde el script original de la ventana principal. Por ejemplo, si quisiéramos cerrar la nueva ventana desde nuestro script, simplemente tendríamos que hacer: subVentana.close();

Aquí si que es necesario especificar subVentana, ya que si escribiéramos window.close(), self.close() o close() estaríamos intentando cerrar nuestra propia ventana (previa confirmación), pero no la subVentana que creamos en los pasos anteriores.

Véase el siguiente ejemplo que permite abrir y cerrar una sub-ventana:

```
<!DOCTYPE  html>
<html>
  <head>
    <meta  http-equiv="content-type"  content="text/html;charset=utf-8">
    <title>Apertura y Cierre de Ventanas</title>
    <script type="text/javascript">
      const inicializar = () => {
        document.getElementById("crear-ventana").onclick=crearNueva;
        document.getElementById("cerrar-ventana").onclick=cerrarNueva;
      }

      let  nuevaVentana;
```

```
        const crearNueva = () =>  {
           nuevaVentana  =
window.open("http://www.google.es","","height=400,width=800");
        }

        const cerrarNueva = () =>  {
           if  (nuevaVentana) {
              nuevaVentana.close(); nuevaVentana  =  null;
           }
        }
     </script>
  </head>
  <body onLoad="inicializar()">
  <h1>Abrimos y cerramos ventanas</h1>
  <form>
     <input  type="button"  id="crear-ventana"  value="Crear Nueva Ventana">
     <input  type="button"  id="cerrar-ventana"  value="Cerrar Nueva Ventana">
  </form>
</html>
```

**Propiedades y métodos.**

El objeto window representa una ventana abierta en un navegador. Si un documento contiene marcos (<frame> o <iframe>), el navegador crea un objeto window para el documento HTML, y un objeto window adicional para para cada marco.

Propiedades del objeto Window

| Propiedad | Descripción |
|---|---|
| closed | Devuelve un valor Boolean indicando cuando una ventana ha sido cerrada o no. |
| defaultStatus | Ajusta o devuelve el valor por defecto de la barra de estado de una ventana. |
| document | Devuelve el objeto document para la ventana. |
| frames | Devuelve un array de todos los marcos (incluidos iframes) de la ventana actual. |
| history | Devuelve el objeto history de la ventana. |
| length | Devuelve el número de frames (incluyendo iframes) que hay en dentro de una ventana. |
| location | Devuelve la Localización del objeto ventana (URL del fichero). |
| name | Ajusta o devuelve el nombre de una ventana. |
| navigator | Devuelve el objeto navigator de una ventana. |
| opener | Devuelve la referencia a la ventana que abrió la ventana actual. |
| parent | Devuelve la ventana padre de la ventana actual. |
| self | Devuelve la ventana actual. |
| status | Ajusta el texto de la barra de estado de una ventana. |

Métodos del objeto Window

| Método | Descripción |
|---|---|
| alert() | Muestra una ventana emergente de alerta y un botón de aceptar. |
| blur() | Elimina el foco de la ventana actual. |
| clearInterval() | Resetea el cronómetro ajustado con setInterval(). |

setInterval() Llama a una función o evalúa una expresión en un intervalo especificado (en milisegundos).
close() Cierra la ventana actual.
confirm() Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
focus() Coloca el foco en la ventana actual.
open() Abre una nueva ventana de navegación.
prompt() Muestra una ventana de diálogo para introducir datos.

## 3.7.6. Document

Cada documento cargado en una ventana del navegador, será un objeto de tipo document.

El objeto document proporciona a los scripts, el acceso a todos los elementos HTML dentro de una página.

Este objeto forma parte además del objeto window, y puede ser accedido a través de la propiedad window.document o directamente document (ya que podemos omitir la referencia a la window actual).

| Colecciones | Descripción |
|---|---|
| anchors[] | Es un array que contiene todos los hiperenlaces del documento. |
| forms[] | Es un array que contiene todos los formularios del documento. |
| images[] | Es un array que contiene todas las imágenes del documento. |
| links[] | Es un array que contiene todos los enlaces del documento. |

| Propiedades | Descripción |
|---|---|
| cookie | Devuelve todos los nombres/valores de las cookies en el documento. |
| domain | Cadena que contiene el nombre de dominio del servidor que cargó el documento. |
| referrer | Cadena que contiene la URL del documento desde el cuál llegamos al documento actual. |
| title | Devuelve o ajusta el título del documento. |
| URL | Devuelve la URL completa del documento. |

| Método | Descripción |
| --- | --- |
| close() | Cierra el flujo abierto previamente con document.open(). |
| getElementById() | Para acceder a un elemento identificado por el id escrito entre paréntesis. |
| getElementsByName() | Para acceder a los elementos identificados por el atributo name escrito entre paréntesis. |
| getElementsByTagName() | Para acceder a los elementos identificados por el tag o la etiqueta escrita entre paréntesis. |
| open() | Abre el flujo de escritura para poder utilizar document.write() o document.writeln en el documento. |
| write() | Para poder escribir expresiones HTML o código de JavaScript dentro de un documento. |
| writeln() | Lo mismo que write() pero añade un salto de línea al final de cada instrucción. |

https://www.w3schools.com/jsref/dom_obj_document.asp

# 3.7. Almacenamiento de datos Web

Antes de HTML5, los datos de la aplicación tenían que almacenarse en cookies, incluidos en cada solicitud del servidor. El almacenamiento Web es más seguro y se pueden almacenar cantidades de datos localmente, sin afectar el rendimiento Web.

A diferencia de las cookies, el límite de almacenamiento es mucho mayor (al menos 5 MB) y la información nunca se transfiere al servidor.

localStorage.*  (No expira su almacenamiento)
sesionStorage.* (Los datos se eliminan al cerrar la sesión del navegador)

https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
https://developer.mozilla.org/es/docs/Web/API/Window/sessionStorage

**Ejercicio:**
Crea una función para contar las veces que se hace click a un botón. El número de veces tiene que quedar almacenado.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        //El objeto Storage nos permite almacenar datos de manera local en el navegador
        //y sin necesidad de realizar conexión a una base de datos.

        function clicador() {
            //validacion para comprobar que el objeto es compatible con tu navegador
            if (typeof(Storage) !== "undefined") {
                if (localStorage.clickcount) {
                    localStorage.clickcount = Number(localStorage.clickcount) + 1;
                } else {
                    localStorage.clickcount = 1;
                }
                document.getElementById("result").innerHTML = "Has hecho click " + localStorage.
clickcount + " veces desde la primera vez.";
            } else {
                document.getElementById("result").innerHTML = "Tu navegador no soporta almacenam
iento web";
            }
        }
        //localStorage.setItem('Nombre', 'Alvaro Enrique') Guardar datos
        //localStorage.getItem('Nombre') Recuperar datos
        //localStorage.removeItem(Nombre) Eliminar elementos
        //localStorage.clear() Limpiar todo el storage
    </script>
</head>
<body>
    <p><button onclick="clicador()" type="button">Haz Click</button></p>
    <div id="result"></div>
    <p>Contador</p>
    <p>Cierra el navegador y comprueba el dato almacenado</p>
</body>
</html>
```

**Ejercicio:**

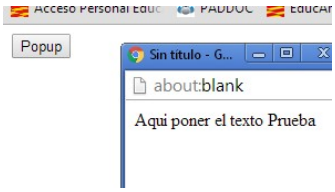
Crea la misma función pero que el número de veces se inicie a 0 cuando se cierre la sesión del navegador.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function clickCounter() {
            if (typeof(Storage) !== "undefined") {
                if (sessionStorage.clickcount) {
                    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
                } else {
                    sessionStorage.clickcount = 1;
                }
                document.getElementById("result").innerHTML = "Has hecho click " + sessionStorag
e.clickcount + " veces en esta sesion.";
            } else {
                document.getElementById("result").innerHTML = "Lo sentimos, tu navegador no
soporta el objeto Storage...";
            }
        }
    </script>
</head>
<body>
    <p><button onclick="clickCounter()" type="button">Haz Click</button></p>
    <div id="result"></div>
    <p>Contador</p>
    <p>Cierra el navegador y comprueba que no hay ningun dato almacenado</p>
</body>
</html>
```

# 3.8. Ejercicios

Ejercicio: Crea un script que mediante un botón que vaya a la pantalla anterior de navegación. Emplea windows.history.back()

Ejercicio: Crear un script que accionado mediante un enlace, abra un pop-up con una página web.
Ayuda: window.open("url", "nombre de la ventana", “tamaño”, “true/false”)
No son obligatorios cada uno de los valores.




Ampliar: centrar el popup en la pantalla y abrir una nueva página web.
Ayuda:
screen.width
screen.heigh

Ejercicio: Crea un script que recorra un bucle desde el 1 hasta el 500. Por pantalla solo mostraremos los múltiplos de 3 y el número de números que no son primos

Ejercicio: Haz un script para mostrar por pantalla el número menor y mayor. Para ello, se introducirán en el código 10 números enteros dentro de un array y se comprobarán cada una de las posiciones.
Ampliación: realizar la misma operación introduciendo por teclado 10 números.

Ejercicio: Crea un script que permita identificar la talla de una prenda de ropa, a partir de las tallas europeas. Los valores de la tallas europeas son: XXL, XL,L,M,S, XS, XXS. La talla esperada sería: Grande, Mediana y Pequeña.
Introducir valor por teclado y sacar resultado por pantalla.
Ayuda:
Grande: {XXL,XL,L}; Mediana:{S}; Pequeña: {S, XS, XXS}

Ejercicio: Calcular la longitud (L) de la circunferencia, dado su radio empleando objetos propios de Javascript.
Ayuda: L=2*π*radio.

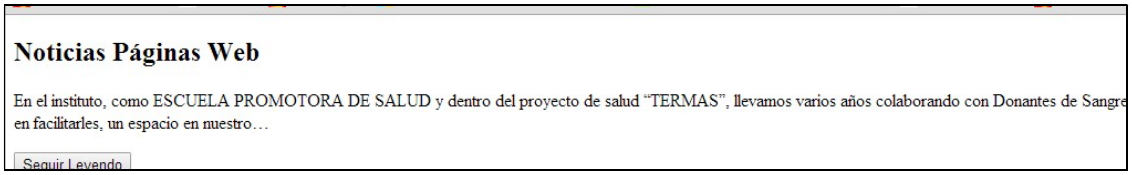
Ejercicio: La administración general de loterías europea, nos ha encomendado generar un script para rellenar la puesta de EUROMILLONES de forma aleatoria.
Nota:
Calcular de 5 números de 1 al 50
Calcular 2 estrellas del 1 al 10
Ayuda:
Math.random(). Genera un número aleatorio entre 0 y 1.
Math.ceil(). Redondea al siguiente número entero.
Math.round(). Redondea al número entero más próximo. Ej.: 5,4 =5 ó 5,6=6 ó 5,5=6
Método array.push(). Añadir un nuevo elemento al array y devolver su nuevo tamaño.

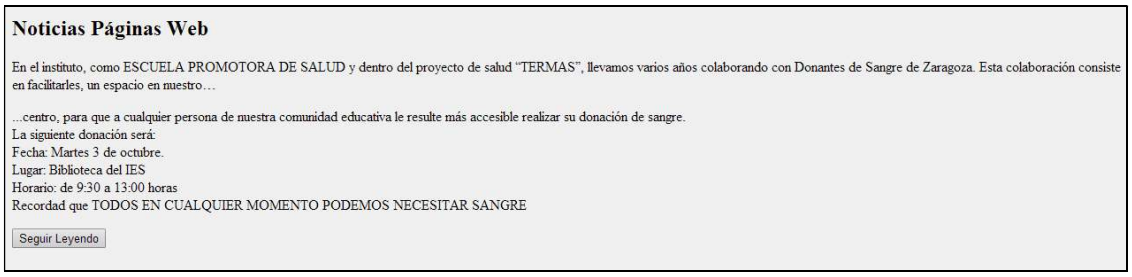
Ejercicio: Introduce el tamaño de una matriz, indicando número de filas (x) y número de columnas (y). Crea los bucles necesarios para poder mostrar por pantalla aquellas posiciones que contengan ambas coordenadas.
Ejemplo: Matriz 5x3. Solución: (5,1) (5,2)…..(1,3)…(5,3)

Ejercicio: Completar el script de la calculadora. Implementando los botones de los números (0,1….9), realizar las operaciones básicas (multiplicar, dividir, sumar y restar), botón de borrado.

Añadir un cuadro secundario para visualizar el resultado anterior calculado que se actualizará cuando se inicie una nueva operación.
Nota: Como sugerencia, se puede incluir también una hoja de estilos en CSS.
Ampliación: Introducir el cálculo de la raíz cuadrada. Aplicando la función
Math.sqrt(numero).

Ejercicio: Crea un script para insertar texto y poder seguir leyendo una noticia.

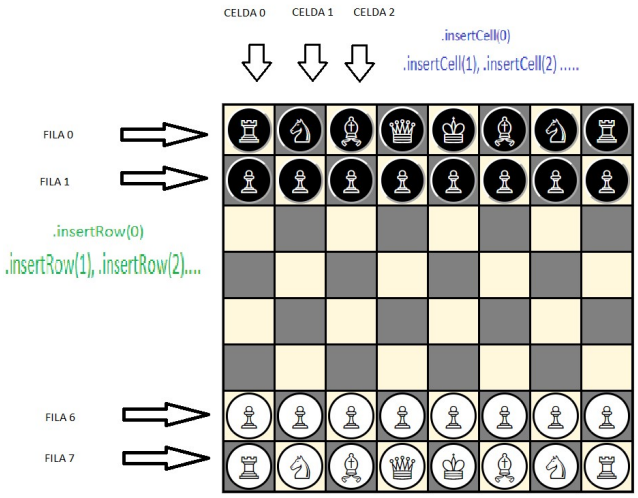



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h2>Noticias Paginas Web</h2>
    <p>En el instituto, como ESCUELA PROMOTORA DE SALUD y dentro del proyecto de salud, llevamos
 varios años colaborando con Donantes de Sangre de Zaragoza. Esta colaboracion consiste en facil
itarles, un espacio en nuestro...</p>
    <p id="demo1"></p>
    <script>
        function myFunction() {
            document.getElementById("demo1").innerHTML = "...centro, para que a  cualquier perso
na de nuestra comunidad educativa le resulte mas accesible realizar su donacion de sangre." +
                "<br>La siguiente donacion sera:" +
                "<br>Fecha: Martes 3 de octubre." +
                "<br>Lugar: Biblioteca del IES" +
                "<br>Horario: de 9:30 a 13:00 horas" +
                "<br>Recordad que TODOS EN CUALQUIER MOMENTO PODEMOS NECESITAR SANGRE";
        }
    </script>
    <button type="button" onclick="myFunction()">Seguir Leyendo</button>
</body>
</html>
```

# 3.9 Ejercicios de Ampliación

Ejercicio Ampliación 1:
Realiza una aplicación Web con Javascript que cree una tabla de 8 filas por 8 celdas, rellenando cada una con la siguiente disposición de elementos:




Para ello, emplea el siguiente archivo:

- EjercicioAmpliacion1.7z

Ejercicio Ampliación 2:
Emplea el siguiente archivo:

- EjercicioAmpliacion2.7z