BOM - Objetos del navegador. Window

- C.F.G.S. DAW
- 6 horas semanales
- Mes aprox. de impartición: Nov

Carmelo José Jaén Díaz

- iPasen - cjaedia071@g.educaand.es

Índice

OBJETIVO

- Identificar el concepto de BOM con los objetos que lo componen y saber utilizarlo en el desarrollo de aplicaciones web.

GLOSARIO

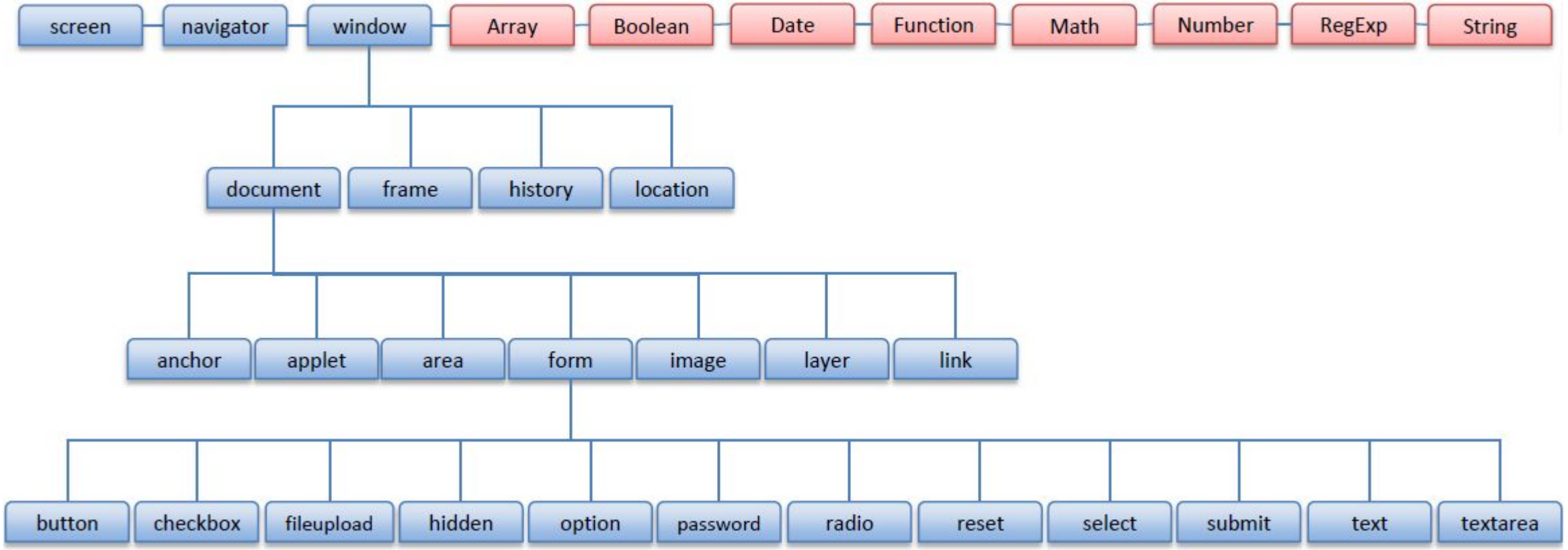
BOM (Browser Object Model). Objeto en el que se representa el navegador, no solo el documento. Mediante el BOM se puede acceder a datos como el historial de navegación, dimensiones de la ventana, etcétera.

DOM. Plataforma e interfaz de un documento que permite a los scripts y programas acceder y modificar dinámicamente su contenido, estructura y estilo.

Tag. Término en inglés que significa “etiqueta” y hace referencia a una palabra clave que sirve para describir un documento.

INTRODUCCIÓN

Los **objetos** de JavaScript se **ordenan** de modo **jerárquico**, tal y como se puede ver en el siguiente esquema.



BOM

El **BOM** (*Browser Object Model*) en su momento fue implementado por los navegadores para que JavaScript pudiese hacer uso de sus métodos y propiedades de forma uniforme.

Como se puede observar en la figura anterior, el objeto raíz del **BOM** es el objeto ***window***, que está soportado por cualquier navegador y que, a su vez, contiene otros objetos de menor rango jerárquico, como puede ser el objeto ***document*** (muy importante también) u otros.

BOM

Las diferencias entre **BOM** y **DOM** serían las siguientes:

- Con el **DOM**, JavaScript puede acceder a los elementos de un documento o página web mediante su estructura interna.

- Con el **DOM**, no se puede acceder a ciertos aspectos del navegador como puede ser la **URL** o las dimensiones de la ventana (navegador) ni tampoco se puede cerrar o redimensionar la ventana del navegador, gestionar cookies, etc. Para ello, se utiliza el **BOM**, que es otra estructura arborescente similar al DOM y algo más completa al añadírsele otra serie de objetos.

- A diferencia del DOM, el elemento raíz es el objeto window.

BOM

Por lo tanto, podrían ejecutarse las siguientes líneas de código:

```
h = window.document.getElementById("header");
```

O bien:

```
h = document.getElementById("header");
```

Y el resultado de ambas líneas sería el mismo.

BOM

FUNDAMENTAL

El objeto ***document*** en el **BOM** y en el **DOM**

Como se puede observar, en el BOM existe un objeto document al igual que en el DOM. En el DOM, el objeto document es la raíz del árbol de objetos de la página web, mientras que en el BOM se puede acceder a propiedades como:

- ***document.URL***. Contiene la URL de la página web actual.
- ***document.referrer***. Contiene la URL desde que se accedió a la página actual.
- ***document.title***. Se accede al texto de la etiqueta <title> de la página web.
- ***document.lastModified***. Contiene la fecha de última modificación de la página web.

INTRODUCCIÓN

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 se encuentra en lo **más alto de la jerarquía**.
- Posee **más propiedades y métodos** que los demás objetos de JavaScript.
- En el mismo nivel se encuentran los objetos predefinidos del lenguaje, estudiados anteriormente.

INTRODUCCIÓN

El objeto **window** tiene una serie de propiedades, tales como:

- ***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.

INTRODUCCIÓN

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 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

INTRODUCCIÓN

Por último, dentro de window se encuentran otros objetos del navegador que veremos más adelante, en lecciones posteriores:

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

CARACTERÍSTICAS

- Representa una ventana abierta en un navegador.

- Si una ventana tiene etiquetas de tipo <iframe> (ver RA2 DIW), el navegador crea un objeto window para el documento HTML inicial y uno para cada <iframe>.

- Todos los objetos, funciones, variables, ... que contiene una ventana son miembros del objeto window.
    - Las funciones son su métodos
    - Las variables globales y el DOM son sus propiedades.

Importante: *No todos los métodos son aplicables a todos los navegadores, por lo que hay que comprobar la compatibilidad de los mismos (ver W3SCHOOLS o MDN Web Docs)*

PROPIEDADES DE *Window*.
Nombre de la ventana.

Podemos encontrar todas las propiedades y métodos de *Window* en W3Schools.com>JavaScript>JS References. Adicionalmente, podemos comprobar cuales de estos son compatibles en los diferentes navegadores.

A continuación, vamos a trabajar con diferentes propiedades de window.

Tal y cómo se mencionaba en la diapositiva anterior, el navegador crea un objeto window para el documento HTML inicial. Si deseamos identificarlo con un nombre:

```
window.name = "Mi ventana";
```

PROPIEDADES DE *Window*.
Nombre de la ventana.

```
window.name = "Mi ventana";
```

Mediante la variable texto, mostraremos la salida de las diferentes propiedades de window.

```
let texto = "";

//Nombre de la ventana
texto += "<br/>Nombre: "+window.name;
```

PROPIEDADES DE *Window*.
Tamaño con *toolbar* y *scrollbar*.

Mediante la variable texto, mostraremos la salida de las diferentes propiedades de window.

```
let texto = "";

//Tamaño de la ventana con toolbar y scrollbar
texto += "<br/>Ancho externo: "+window.outerWidth;
texto += "<br/>Alto externo: "+window.outerHeight;
```

PROPIEDADES DE *Window*.
Tamaño sin *toolbar* y *scrollbar*.

Mediante la variable texto, mostraremos la salida de las diferentes propiedades de window.

```
let texto = "";

//Tamaño de la ventana sin toolbar ni scrollbar
texto += "<br/>Ancho interno: "+window.innerWidth;
texto += "<br/>Alto interno: "+window.innerHeight;
```

PROPIEDADES DE *Window*.
Scroll horizontal y vertical.

Mediante la variable texto, mostraremos la salida de las diferentes propiedades de window.

```
let texto = "";

//Scroll horizontal y vertical
texto += "<br/>Scroll horizontal: "+window.pageXOffset;
texto += "<br/>Scroll vertical: "+window.pageYOffset;
```

PROPIEDADES DE *Window*.
Distancia de la esquina superior izquierda.

Mediante la variable texto, mostraremos la salida de las diferentes propiedades de window.

```
let texto = "";

//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 DE *Window*.
Visualización de las propiedades anteriores

Por último, incluimos en el código HTML una etiqueta *párrafo* con el identificador “ventana” y mostramos los valores de las propiedades anteriores.

```
<p id="ventana"></p>

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

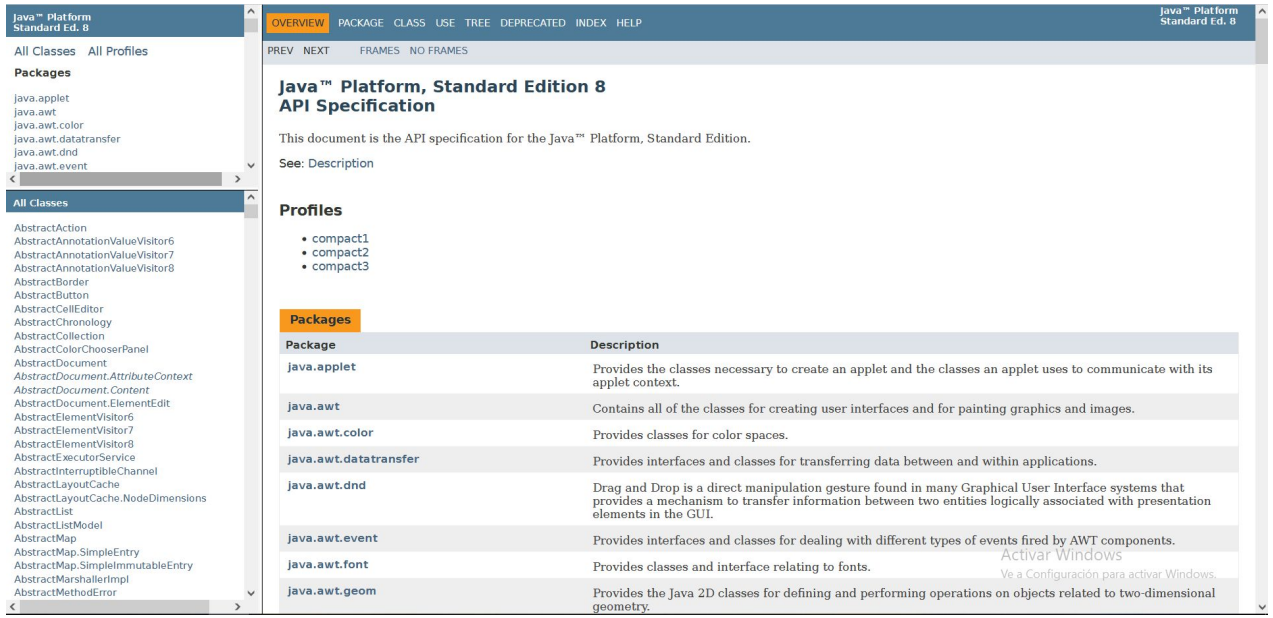
La salida del código anterior puede observarse en el siguiente enlace:

https://codepen.io/Carmelo-Jos-Ja-n-D-az/pen/NPKjzVq

MARCOS (*frames*)

Los marcos o frames nos permiten **dividir una web en varias ventanas** que pueden cargar otras páginas webs. Se trata de ventanas independientes incorporadas en una misma página y que nos permiten distribuir la información de forma organizada.

La ventana se puede dividir en marcos horizontales o verticales, pero no ambos. De todas formas, existe la posibilidad de que un marco contenga a su vez otros marcos. Así los marcos se pueden anidar para hacer todo tipo de distribuciones.



PROPIEDADES DE *iframes*

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.

Aunque actualmente se emplean `iframes` para incrustar código embebido, algunos motivos por los que ya no se utilizan los marcos (*frames* o *framesets*) son:

- Los motores de búsqueda no indexan bien las páginas creadas con framesets.
- Ocupan espacio de la pantalla.
- Las funcionalidades de ir para adelante o para atrás en el historial de navegación del navegador no se pueden utilizar.
- Presentan problemas de usabilidad y accesibilidad web.

OTROS OBJETOS DEL NAVEGADOR

Como hemos comentado, el objeto **window** es el objeto padre del que *cuelgan* otros objetos como:

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

Aunque no es necesario escribir el *window* delante de los mismos para acceder a ellos ya que está implícito al ser el objeto padre. También ocurre con otros métodos como ***alert()***.

MÉTODOS DEL OBJETO *window*

Hasta ahora, se han estudiado todas las propiedades del objeto ***window***. A continuación, se presentan sus métodos:

- **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.

MÉTODOS DEL OBJETO *window*

Hasta ahora, se han estudiado todas las propiedades del objeto ***window***. A continuación, se presentan sus métodos:

- **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.

Importante: Es importante declarar una ventana en una variable, ya que, de este modo podemos ejecutar sus métodos posteriormente sin problemas.

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

A continuación, mediante un ejemplo guiado se pone en práctica los métodos citados para facilitar su comprensión.

En primer lugar, vamos a crear una nueva ventana

```
let miVentana; //Crear fuera de las funciones para poder acceder a ella
```

A continuación, vamos a abrir la ventana para lo cual empleamos el método ***open()*** que requiere de los siguientes parámetros ***open(<URL>,<nombre>,<especificaciones>)***.

En W3Schools, podemos consultar en qué consisten cada uno de estos parámetros.

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Una vez revisadas las opciones del método *open()* en W3Schools, vamos a definir una función para abrir una nueva ventana.

```
function crearVentana(){
        //Si optamos por abrir una web conocida, podemos hacer esto
        //En este caso, es importante poner el protocolo http ya que
        sino JS interpretaría que //estamos buscando una carpeta en
        nuestro disco duro.
        miVentana=window.open("http://www.google.com");
    }
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

A continuación, para ir testeando el comportamiento de este ejemplo guiado, en un documento HTML vinculamos el fichero JS dónde estamos implementando este código y creamos un botón en el <body> que llame a la función definida anteriormente.

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

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

También podemos optar por crear una ventana vacía con las siguientes propiedades:

```
function crearVentana(){
            //Al no pasar como argumento nada en el name, nos abrirá una
nueva ventana
            miVentana = window.open("","","width=500,height=200");
            //Para introducir código en la ventana podemos hacer algo así
            //Aunque esto se estudiará en el RA6. DOM
            miVentana.document.write("<h1>Mi ventana</h1>");
        }
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Una vez visto cómo se puede crear una ventana, veremos cómo cerrarla. Para ello, empleamos la función ***close()*** que nos permite cerrar una ventana en concreto.

```
    //close(): cierra una ventana en concreto, en este caso miVentana ya
que de otro modo cerraría la ventana principal.

    function cerrarVentana(){
        miVentana.close();
    }
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

A continuación, en el documento HTML creado previamente para testear el código, creamos otro botón en el <body> que llame a la función definida anteriormente.

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

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Ahora, vamos a ver el comportamiento del método ***resizeBy()*** cuya sintaxis es:

```
//resizeBy(<nºpix_ancho>,<nºpix_alto>): redimensiona una ventana un número de píxeles respecto a su tamaño actual

//Si la ventana tiene un tamaño de 100px 100px, se redimensiorá a 110px, 110px. Si llamamos otra vez a la función a, 120px 120px, etc.

        function redimensionarVentana(){
            miVentana.resizeBy(10,10);
        }
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

A continuación, en el documento HTML creado previamente para testear el código, creamos otro botón en el <body> que llame a la función definida anteriormente.

```
<button onclick="redimensionarVentana()">Redimensionar ventana</button>
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Si lo que queremos es redimensionar una ventana a un tamaño concreto, debemos usar el método ***resizeTo()*** cuya sintaxis es:

```
//resizeTo(<nºpix_ancho>,<nºpix_alto>): redimensiona una ventana al número de píxeles indicado
```

De manera análoga a estos métodos, podemos mover la ventana.

```
//moveTo(<nºpix_ancho>,<nºpix_alto>): mueve una ventana a una posición en concreto
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Y el método ***moveBy()***,

```
//moveBy(<nºpix_ancho>,<nºpix_alto>): mueve una ventana un número de píxeles respecto a su posición actual

        function moverVentana(){
            miVentana.moveBy(10,10);
        }
```

A continuación, en el documento HTML creado previamente para testear el código, creamos otro botón en el <body> que llame a la función definida anteriormente.

```
<button onclick="moverVentana()">Mover ventana</button>
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Si tuviéramos una ventana cuyo contenido fuera un texto extenso en el que tengamos que hacer *scroll*, podemos implementar las siguientes funciones:

```
//scrollBy(<nºpix_ancho>,<nºpix_alto>): mueve las barras de scroll un número de píxeles desde la posición actual.

//scrollTo(<nºpix_ancho>,<nºpix_alto>): mueve las barras de scroll a una posición determinada
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

El método ***focus()***,

```
//focus(): pone el foco en la ventana indicada

        function enfocar(){
            miVentana.focus();
        }
```

A continuación, en el documento HTML creado previamente para testear el código, creamos otro botón en el <body> que llame a la función definida anteriormente.

```
<button onclick="enfocar()">Enfocar ventana</button>
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

Por último, el método ***print()***,

```
//print(): imprime la ventana indicada

        function imprimir(){
            //Si optamos por esta opción imprime la ventana actual
            //Equivalente a window.print();
            print();
            //Si optamos por esta opción imprime la ventana indicada
            miVentana.print();
        }
```

MÉTODOS DEL OBJETO *window*
Ejemplo guiado

A continuación, en el documento HTML creado previamente para testear el código, creamos otro botón en el <body> que llame a la función definida anteriormente.

```
<button onclick="imprimir()">Imprimir ventana</button>
```

Finalmente, existe el método ***stop():***

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

Útil cuando la página tarda mucho en cargarse debido a que esté realizando muchas operaciones o contenga muchos datos, combinable con la instrucción de tiempo ***setTimeout()***, que estudiaremos más adelante.

MÉTODOS DEL OBJETO *window*
Cuadros de diálogo

Los cuadros de diálogo son tres métodos más del objeto ***window*** que podemos utilizar para interactuar con el usuario mediante **ventanas emergentes**.

- **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*.

Recuerda: Para poder saber lo que ha elegido un usuario en un ***prompt*** o en un ***confirm*** es necesario que tanto uno como otro los asignemos a una variable.

MÉTODOS DEL OBJETO *window*
Cuadros de diálogo - *alert()* y *prompt()*

```
//alert(<mensaje>): muestra un mensaje al usuario y no devuelve ningún valor.

        alert("Soy Carmelo, docente de DAW");

//prompt(<mensaje>[,<texto por defecto>]): muestra un mensaje al usuario y un
campo que puede contener o no texto por defecto para orientar al usuario
sobre la información a introducir. Lo que introduzca el usuario se almacenará
en una variable a la que se asigna.

        let respuesta = prompt("¿Cuál es tu nombre?", "Introduce un nombre");
        alert ("Hola, "+respuesta);
```

MÉTODOS DEL OBJETO *window*
Cuadros de diálogo - *confirm()*

```
//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.
      let respuesta2 = confirm("¿Te gusta javascript?");
      if (respuesta2==true) alert ("¡Me alegro!");
      else alert ("¡Pues es una pena!");

      //O bien usando un operador ternario
      alert((respuesta2) ? "¡Me alegro!" : "¡Pues es una pena!");
```

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo

En esta última parte, se estudian tres métodos del objeto ***window*** que nos permitirán ejecutar código en intervalos de tiempo.

- **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.

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *setTimeout*

A continuación, mediante un ejemplo guiado se pone en práctica los métodos citados para facilitar su comprensión.

En primer lugar, vamos a crear una función en un **script.js**

```
//setTimeout(<funcion>, <milisegundos>): indicar cuántos milisegundos tienen que pasar antes de que la función indicada en el primer parámetro se ejecute.

function saludar(){
    alert ("¡Hola!");
}
```

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *setTimeout*

A continuación, para ir testeando el comportamiento de este ejemplo guiado, en un documento HTML vinculamos el fichero JS dónde estamos implementando este código y creamos un botón en el <body> que llame a la función definida anteriormente a través del método ***setTimeout***.

```
<button onclick="setTimeout(saludar,3000)">Saluda</button>
```

Nota: *La función* ***saludar()*** *se la pasamos como argumento al método* ***setTimeout()*** *sin paréntesis.*

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *clearTimeout*

Si tras pulsar el botón, el tiempo de espera es demasiado alto o por el motivo que fuese, queremos parar la ejecución de la función asociada a ***setTimeout()***, podemos emplear ***clearTimeout()***.

```
//clearTimeout(<variable asociada a setTimeout>): si asignamos un setTimeout a una variable podemos detener su ejecución con clearTimeout.
```

Para ver el comportamiento de dicho método, creamos otro botón en nuestro documento HTML, asociándole una variable `sal` al método ***setTimeout()***:

```
<button onclick="sal = setTimeout(saludar,3000)">Saluda con variable</button>
<button onclick="clearTimeout(sal)">Cancelar saludo</button>
```

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *setInterval*

Análogamente tenemos ***setInterval()*** y ***clearInterval()***.

```
//setInterval(<funcion>, <milisegundos>): repite una función cada intervalo de tiempo.
```

A continuación, asigno el método ***setInterval()*** a la variable `int` pasándole como argumento la función `reloj`.

```
let int = setInterval(reloj, 1000);

function reloj(){
    let fecha = new Date(); //Fecha actual
    document.getElementById("reloj").innerHTML = fecha.getSeconds();
        }
```

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *setInterval*

```
function reloj(){
    let fecha = new Date(); //Fecha actual
    document.getElementById("reloj").innerHTML = fecha.getSeconds();
        }
```

La función **reloj**, guarda la fecha actual en una variable **fecha**.

A continuación, usamos la propiedad ***innerHTML*** del objeto ***document*** para sustituir el contenido de un párrafo (que implementaremos previamente en nuestro documento HTML) por el valor asignado.

Como esta función se ejecuta cada segundo (1000 ms) a través del ***setInterval()*** simulará una cuenta ascendente a modo de cronómetro.

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *setInterval*

La salida del código anterior puede verse en la imagen adjunta o en el siguiente link:

https://codepen.io/Carmelo-Jos-Ja-n-D-az/pen/LEPmRyx

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *clearInterval*

Por último, si quisiéramos parar el cronómetro podríamos hacer uso del método ***clearInterval()*** análogo al método clearTimeout().

Modificaremos el código anterior incorporando un botón para parar el tiempo.

```
<button onclick="clearTimeout(int)">Cancelar reloj</button>
```

MÉTODOS DEL OBJETO *window*
Instrucciones de tiempo - *setInterval*

La salida del código anterior puede verse en la imagen adjunta o en el siguiente link:

https://codepen.io/Carmelo-Jos-Ja-n-D-az/pen/QwLrKXL