APUNTES UD-4.
EL *DOM* Y EL *BOM*.

Se trabajará el concepto de *DOM* y de *BOM*. La ventaja que tiene JavaScript es poder acceder a ambos objetos, lo cual consigue que las aplicaciones sean interactivas.

4.1. Introducción.

Accediendo al DOM y al BOM, el programador va a poder conocer la potencia de la programación del lado del cliente. Este hecho permitirá que mucha de la lógica de las aplicaciones web corra en este lado dedicándose el servidor a temas más específicos de back-end.

4.2. Formularios.

El trabajo con campos es la base de las aplicaciones web. Con HTML5 es posible comprobar el contenido de muchos de los campos de una web. Pero, a veces es necesario ciertas validaciones y acciones extra que pueden realizarse en JavaScript.

4.2.1. La validación de campos.

Ejemplo donde se validará un campo de texto donde sólo se podrá introducir un número entre 5 y 10:

Ejemplo: ***validación de campos.***

```
<!DOCTYPE html><html><body>
        <p>Introduzca un número entre 5 y 10:</p>
        <input id="aqui">
        <button onclick="dale()">Comprueba</button>
        <p id="msg"></p>
        <script>
            function dale(){
                var msg, x, err;
                msg = document.getElementById('msg');
                msg.innerHTML =" ";
                x = document.getElementById('aqui').value;
                try{
                    if(x == "")throw "Está vacío";
                    if(isNaN(x))throw "No es un número";
                    x = Number(x);
                    if(x < 5) throw "No llega";
                    if(x > 10) throw "Se pasa";
                }
                catch(err){msg.innerHTML = "Resultado: " + err;}
            }
```

4.2.1. La validación de campos.

En el código se ve que existe un campo “input” de texto que es el que va a validarse con identificador (id) “aqui”. A parte del campo por validar , se añade un botón que, al presionarlo, ejecute la función “dale()”, que es la encargada de validar el campo anterior.

Para la validación se utilizan las sentencias “try” y “catch”, de tal manera que, si el campo de texto contiene algún valor incorrecto, el código lanzará una excepción recogida con la sentencia “catch”. El bloque de “catch” se encargará de mostrar el texto lanzado en la excepción.

La validación anterior es posible realizarla solamente con HTML.

4.3. Expresiones regulares.

- Se pueden utilizar en los métodos ***exec*** y ***test*** del objeto **RegExp**, y también en los métodos ***search***, ***replace*** y ***match***.
- Una expresión regular es una secuencia de caracteres que forman un patrón determinado.
- Se utilizan para realizar operaciones de búsqueda y reemplazamientos en strings.
- Son un concepto heredado del UNIX donde se utilizan mucho en comandos del sistema como ***grep***, ***find***, ***awk***.
- Pueden ser sencillas o complejas dependiendo del objetivo que se necesite conseguir.

Las expresiones regulares son ampliamente utilizadas en la validación de campos por su potencia y efectividad.

4.3. Expresiones regulares.

La sintaxis de una expresión regular:

```
/patrón/modificadores
```

Un ejemplo sencillo de una expresión regular sería el siguiente:

```
var patron = /kursaal/i;
```

Donde ***/kursaal/i*** es la expresión regular. El objetivo es buscar la cadena ***"kursaal"*** de forma case-insensitive (modificador i).

4.3. Expresiones regulares.

Aplicado a un código concreto:

```
var cadena = "Aprende DAW en el instituto Kursaal";
var patron = /kursaal/i;
var pos = cadena.search(patron);
```

Al ejecutar el código anterior, el valor de la variable “pos” sería 28 porque la cadena “kursaal” aparece en dicha posición.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "gbusmor510@g.educaand.es";
        var patron = /\w/g; // \w selecciona los caracteres de una cadena
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "gbusmor510@g.educaand.es";
        var patron = /\d/g; // \d selecciona los dígitos de una cadena
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "gbusmor510@g.educaand.es";
        var patron = /\W/g; // \W selecciona los NO caracteres de una cadena
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "Aprende DAW en el Instituto Kursaal";
        var patron = /\S/g; // \S elimina los espacios en una cadena
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "Aprende DAW en el Instituto Kursaal";
        var patron = /r+/g; // r+ busca si la cadena tiene una o más erres.
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body>
</html>
```

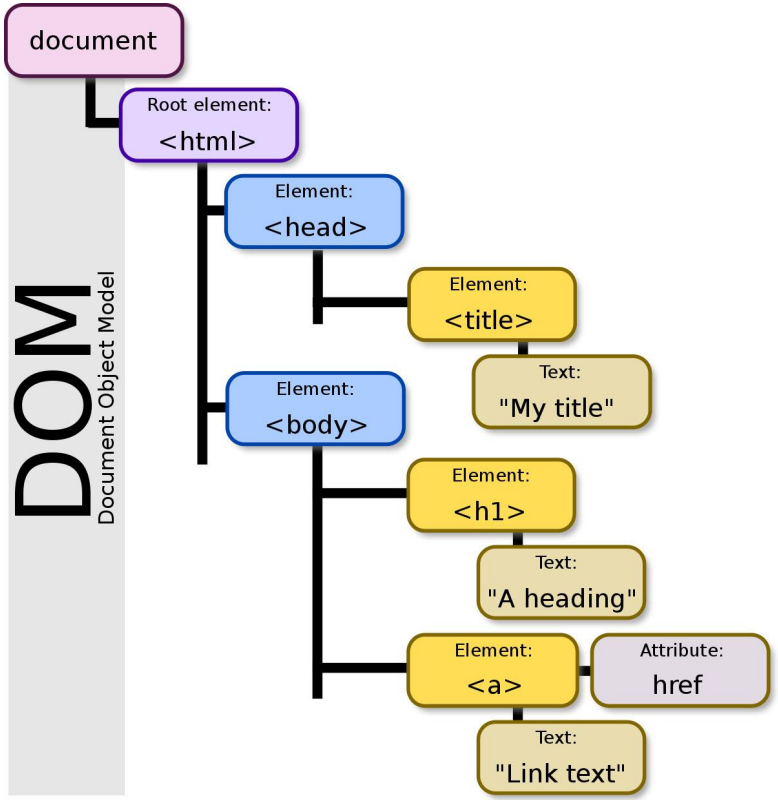
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "Aprende 10 o 20 veces más rápido en el Instituto
Kursaal";
        var patron = /\d{2}/g; // busca si hay una secuencia de dos digitos.
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Expresiones regulares</title>
</head>
<body>
    <script>
        var cadena = "Aprende 100 o 200 veces más rápido en el Instituto
Kursaal";
        var patron = /\d{2,3}/g; // busca si hay una secuencia de 2 o 3
digitos.
        var resultado = cadena.match(patron);
        console.log(resultado);
    </script>
</body></html>
```

4.4. DOM. *Document Object Model*.

Según el W3C, el DOM es una plataforma e interfaz de un documento que permite a los script y los programas acceder y modificar dinámicamente su contenido, estructura y estilo. JavaScript puede acceder al DOM de cualquier página web. Al contrario que las páginas HTML estáticas, con JavaScript, se puede hacer que una web sea dinámica cambiando el contenido HTML de cualquier elemento, cambiando el estilo CSS de esta, reaccionando a los eventos del DOM o incluso añadiendo o eliminando elementos HTML de tu web.

4.4. DOM. *Document Object Model*.



4.4. DOM. *Document Object Model*.

Cuando se carga un documento en el navegador, se crea el objeto ***document***, el cual, además de ser la raíz de todo el documento HTML, tiene una serie muy amplia de propiedades y métodos. Algunas de ellas se citarán a continuación:

- Propiedades: ***activeElement***, ***cookie***, ***domain***, ***images***, ***links***, ***referrer***.
- Métodos: ***open()***, ***querySelector()***, ***getElementById()***, ***createEvent()***.

4.4. DOM. *Document Object Model*.

Con el método ***getElementById()*** podemos obtener el identificador de un elemento HTML. Una vez seleccionado el elemento concreto, mediante la propiedad ***inner.HTML***, podemos cambiar el contenido de dicho elemento.

Además de buscar elementos por el ***"id"***, se pueden buscar elementos por el ***"tag"*** utilizando el método ***getElementsByTagName()*** o también por la clase, utilizando el método ***getElementsByClassName()***.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Ejemplo getElementById</title>
</head>
<body>
    <p id="texto">Curso de <strong>JavaScript</strong></p>
    <p id="texto2">Curso de <strong>ReactJS</strong></p>
    <script>
        var a = document.getElementById("texto");
        var b = a.getElementsByTagName("strong");
        alert (b[0].innerHTML);
    </script>
</body>
</html>
```

4.4. DOM. *Document Object Model*.

Vamos a ver un ejemplo más completo de utilización del DOM con JavaScript. A continuación, se muestra un ejemplo de test sobre la nomofobia. Que es el miedo a estar sin teléfono móvil.

Ejemplo: ***test Nomofobia***

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Test Nomofobia</title>
</head><body>
    <form id="formulario1">
        <ol>
            <input type="checkbox" name="nomo">
            <strong>1.</strong>Cuando mandas un mensaje por WhatsApp esperas siempre
el doble
            chech. Si no aparece vuelves a abrir el terminal para revisarlo al rato.
        </ol>
        <ol>
            <input type="checkbox" name="nomo">
            <strong>2.</strong>Antes de acostarte siempre miras el móvil a ver si
tienes
            mensajes o notificaciones.
        </ol>
```

```
<br>
    </form>
    <p>
        Recuerda marcar todas las cuestiones arriba expuestas que creas que siempre realizas.
    </p>
    <button onclick="myFunction()">Comprueba si sufres nomofobia</button>
    <h3><p id="resultado"></p></h3>
    <h4><p id="consejo"></p></h4>
```

A continuación, comenzamos la programación de nuestra función myFunction dentro de las etiquetas de <script></script>:

Ejemplo: ***test Nomofobia***

```
<script>
        function myFunction(){
            var cont = 0;
            var x = document.forms["formulario1"];
            var i;
            for (i = 0; i < x.length; i++){
                if(x.elements[i].checked){cont++;}
}


 document.getElementById("resultado").innerHTML = "Resultado: En principio no
 tienes nada de qué preocuparte.";
 document.getElementById("consejo").innerHTML = "Consejo: Sigue así. Nada que
 aconsejar.";
```

Ejemplo: ***test Nomofobia***

```
if (cont>5){
document.getElementById("resultado").innerHTML = "Resultado: Empiezas a tener signos de dependencia del móvil.";
document.getElementById("consejo").innerHTML = "Consejo: Puedes utilizar técnicas como apagar el móvil cuando no lo necesitas, cuando duermes, etc."
}

 if (cont>6){
 document.getElementById("resultado").innerHTML = "Resultado: Tienes una gran dependencia del móvil.";
 document.getElementById("consejo").innerHTML = "Consejo: Deberías seguir un plan de <strong>desintoxicación</strong> del móvil como por ejemplo dejar el móvil en casa cuando vas as comprar, apagarlo durante la noche, apagarlo durante horas de clase o trabajo, etc.";
 }
```

```
if (cont>8){
document.getElementById("resultado").innerHTML = "Resultado: Tus síntomas de dependencia son muy preocupantes.";
document.getElementById("consejo").innerHTML = "Consejo: Además de todas las técnicas de los apartados anteriores deberías plantearte un plan de desintoxicación del móvil que consista en esta una o dos semanas sin utilizarlo. Si ves que no puedes hacerlo por ti mismo, pide ayuda a un profesional.";
            }
        }
    </script>
</body>
</html>
```

4.4.1. El acceso al DOM. *document.querySelector()*.

Este método devuelve el primer elemento que encuentre y que concuerde con el selector que se le introduzca como parámetro. Solamente devolverá el primer elemento que encuentre y no todos. Para seleccionar todos los elementos que concuerden con un selector, habrá que utilizar el método ***querySelectorAll()***.

Ejemplo: ***querySelector().***

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Ejemplo querySelector</title>
</head>
<body>
    <h3 class="ejemplo">Introduzca una expresión numérica</h3>
    <label for="texto">Introducir datos:</label>
    <input id="texto" type="text">
    <button>Ejecutar expresión</button>
    <p class="ejemplo">Un párrafo de la clase ejemplo.</p>
    <p>otro párrafo.</p>
    <button onclick="laFuncion()">Cambiar color</button>
```

Ejemplo: ***querySelector().***

```
<script>
        function laFuncion(){
        document.querySelector(".ejemplo").style.backgroundColor = "red";
        document.querySelector("p").innerHTML = "rojo";
        }
        var input = document.querySelector('input');
        var boton = document.querySelector('button');
        var parrafo = document.querySelector('p');
        boton.onclick = function(){
            var dato = input.value;
            parrafo.innerHTML = eval(dato);
        }
    </script>
```

Seleccionamos el primer "input", el primer "botón" y el primer "párrafo".

Asignamos una función al botón y usamos "eval" para ejecutar la expresión introducida en "input".

4.5. Eventos del DOM

JavaScript puede reaccionar a cualquier evento que ocurra en una página web. A continuación, se muestran algunos de los eventos que pueden ser capturados:

- Pulsar una tecla.
- Enviar un formulario.
- Modificar un campo de texto.
- Cuando el ratón pasa sobre un elemento.
- Cuando se carga una imagen.
- Cuando el usuario hace clic sobre un elemento.
- Cuando se carga una página web.

Ejemplo: ***eventosDOM***

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Eventos del DOM</title>
</head>
<body>
    <div onmouseover="dentro(this)" onmouseout="fuera(this)">Instituto Kursaal</div>
    <script>
        function dentro(obj){
            obj.innerHTML = "Desarrollo Web en Entorno Cliente"
        }
        function fuera(obj){
            obj.innerHTML = "Instituto Kursaal"
        }
    </script>
</body>
</html>
```

4.5. Eventos del DOM.

Todos los eventos que ocurren a un objeto están basados en el objeto event. Este objeto contiene las propiedades y métodos que son comunes a todos los eventos. Actualmente, los programadores prefieren el método addEventListener en vez de añadir el evento en todos y cada uno de los elementos HTML afectados. Esta forma de programar es más limpia, puesto que independiza el código HTML del código JavaScript. el W3C pone énfasis en cambiar el modelo tradicional de registro de eventos por este nuevo método.

Ejemplo: ***miraCookies***

```
<!DOCTYPE html>
<html><head>
    <meta charset="UTF-8">
    </head>
<body onload="miraCookies()">
   <p id="muestracookies" ></p>
   <script>
    function miraCookies(){
        var text = "";
        if (navigator.cookieEnabled == true){
            text = "Las cookies están habilitadas." ;
        } else{
            text = "Las cookies están deshabilitadas." ;
        }
        document.getElementById( "muestracookies").innerHTML = text;
    }
    </script></body>
</html>
```

4.6. Manejadores de eventos.

Cuando ocurre un evento de un objeto del documento o página web, es posible con JavaScript asociarlo a una función. Con el método ***addEventListener***, es posibler asociar evento y función Javascript desde el propio script limpiando de código el HTML.

Ejemplo: ***addEventListener***

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Ejemplo addEventListener</title>
</head>
<body>
    <button id="unboton">Dale</button><p id="texto"></p>
    <script>
        function MuestraFecha(){
        document.getElementById("texto").innerHTML = Date();
        }
  document.getElementById("unboton").addEventListener("click",MuestraFecha);
    </script>
</body>
</html>
```

Asigna al contenido HTML de “texto” la fecha actual.

Asocia el evento “click” del botón con la función MuestraFecha.

4.6. Manejadores de Eventos.

Actividad propuesta 4.1.

Del ejemplo expuesto en la diapositiva anterior (35), desliga el código JavaScript a otro fichero y haz que funcione la página web. Coloca el código JavaScript en una subcarpeta scripts para tener una estructura de proyecto más ordenada.

4.7. Nodos del DOM.

Cuando el navegador recibe la página, la interpreta y va creando una estructura en forma de arbol con los elementos recibidos llamada DOM.

Hasta que el navegador no haya construido totalmente el árbol (DOM) cargándose la página por completo, no será posible acceder hasta él.

En el DOM, existen muchos tipos de nodo, pero generalmente, los programadores web utilizan sobre todo cuatro nodos cuando desean manipular el DOM:

4.7. Nodos del DOM.

1. ***Attr***. Este nodo representa los atributos de las etiquetas, que tienen el formato atributo = valor.
2. ***Document***. Es el nodo más importante que constituye la raíz del que derivan los demás nodos del DOM.
3. ***Element***. Cada etiqueta tendrá un nodo de tipo element. Estos nodos pueden tener atributos y, además, otros nodos podrán derivar de ellos.
4. ***Text***. En este nodo, se almacena el texto de una etiqueta.

4.7.1. ¿Cómo se accede a los nodos (elementos HTML)?.

En JavaScript, se puede acceder a los nodos del DOM utilizando las funciones que el DOM proporciona. Las operaciones más comunes:

- Acceder al valor de un atributo.
- Establecer el valor de un atributo.
- Crear o añadir nuevos nodos.
- Borrar nodos.
- Mover nodos de su lugar en el árbol.

4.7.1. ¿Cómo se accede a los nodos (elementos HTML)?.

Lo normal es acceder a los nodos de forma directa mediante alguna de las siguientes funciones:

- ***getElementById()***. Es el más utilizado. Se accede por el identificador que le ha dado el programador al elemento y, generalmente, este identificador es único en la página.
- ***getElementsByTagName()***. Acceder a los nodos mediante su tipo de etiqueta. Se accederá a todos los nodos que tengan un tipo de etiqueta concreta y esta llamada devolverá un array de elementos.
- ***getElementsByName()***. Acceder a los nodos por su nombre. En HTML5, ya no se utiliza dado que el nombre (name) ha sido reemplazado por su identificador (id).

4.7.1. ¿Cómo se accede a los nodos (elementos HTML)?.

- ***getElementsByClassName().*** Se accede al elemento por el nombre de la clase (CSS) o selector CSS a la que pertenece. La llamada a este método devuelve una colección (array) de elementos, el cual se podrá ir recorriendo para acceder a todos ellos.
- ***querySelector()***. Es otro método del DOM que permite acceder al primer elemento que concuerde con el selector que se le pasa como parámetro. Es un método parecido al anterior.

Ejemplo: ***getElementsByClassName()***.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Ejemplo getElementByClassName </title>
</head>
<body>
<p class="parrafo">Primer párrafo.</p>
<p class="parrafo">Segundo párrafo.</p>
<button onclick="dale()">Cambia el texto</button>
<script>
    function dale(){
        var p = document.getElementsByClassName( "parrafo");
        p[0].innerHTML="Párrafo primero.";
        p[1].innerHTML="Párrafo segundo.";
    }
</script></body>
</html>
```

Se accede a los elementos del tipo <p> y se modifica su contenido.

4.7.2. ¿Cómo se crea un nuevo nodo?

Los pasos para crear un nuevo nodo son los siguientes:

1. Crear un nuevo nodo de tipo Element.
2. Crear un nuevo nodo de tipo Text (para albergar el contenido del elemento anterior).
3. Vincular el nodo Text al nodo Element.
4. Vincular el nodo Element a la página de tal manera que el nodo esté en el lugar correspondiente donde se quiera insertar el elemento.

4.7.2. ¿Cómo se crea un nuevo nodo?

El código de los pasos anteriores sería el siguiente:

```
//Crear nodo de tipo Element
var parrafo = document.createElement("p");
//Crear nodo de tipo Text
var contenido = document.createTextNode("Kursaal");
//Vincular el nodo Text coo hijo del nodo Element
parrafo.appendChild(contenido);
//Vincular el nodo Element como hijo de la página
document.body.appendChild(parrafo);
```

Funciones usadas:

- createElement().
- createTextNode().
- appendChild().

4.7.3. ¿Cómo se elimina un nodo?

Si tenemos un elemento HTML en una web como el siguiente:

```
<p id="instituto">IES Kursaal Algeciras</p>
```

Para eliminarlo, habrá que llamar desde el nodo padre a la función ***removeChild()***, la cual requerirá como parámetro el nodo que se va a borrar. Para acceder al nodo padre de un elemento HTML, se puede utilizar la propiedad ***parentNode***.

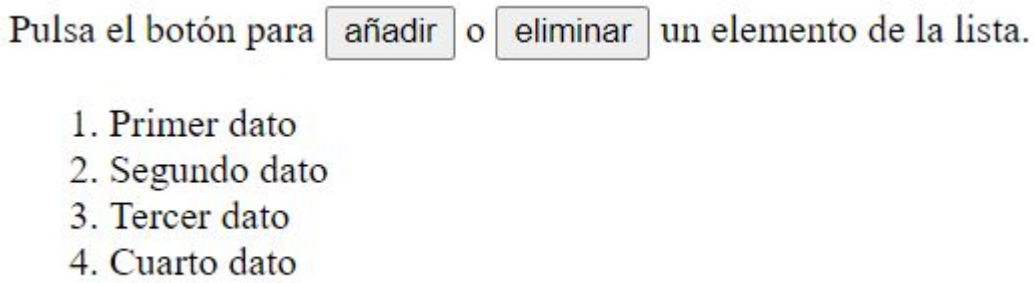
```
var pInstituto = document.getElementById("instituto");
pInstituto.parentNode.removeChild(pInstituto);
```

4.7.3. ¿Cómo se elimina un nodo?

Cuando se elimina un nodo, se eliminarán todos los nodos hijos que pueda poseer. De esa manera, eliminar un nodo puede equivaler a eliminar más de un elemento.

4.7.4. Práctica guiada: creación y eliminación de elementos.

Se va a mostrar una lista de elementos y dos botones: añadir y eliminar. Pulsando el botón añadir, se podrá dar de alta un nuevo elemento a la lista y, pulsando eliminar, se eliminará el último elemento par.



4.8. Propiedades del DOM.

A continuación vamos a ver algunas propiedades avanzadas del DOM.

4.8.1. El acceso a nodos

El manejo de los métodos de acceso es importante, puesto que el programador puede, por ejemplo:

1. **Conocer el número de enlaces que tiene una página web.**

```
var = numEnlaces = document.getElementsByTagName("a");
```

4.8.1. Acceso a los nodos.

La línea anterior almacenará todos los enlaces de la página (en un array de objetos), pero para saber cuantos enlaces hay en dicho array, habrá que utilizar la función length:

```
var mensaje = “Se han encontrado “+numEnlaces.length+”enlaces”;
```

2. Conocer la dirección a la que apunta el último enlace.

Bastará con acceder a la última posición del array y preguntar por la propiedad ***href*** del objeto contenido en él.

```
var=mensaje="El penúltimo enlace apunta a "+numEnlaces[numEnlaces.length-1].href;
```

4.8.1. Acceso a los nodos.

3. Conocer el número de enlaces que están en el tercer párrafo del documento. Bastaría con extraer todos los párrafos del documento, para posteriormente, extraer los enlaces.

```
var parrafos =
document.getElementsByTagName( 'p');
enlaces = parrafos[2].getElementsByTagName('a');
numEnlaces = enlaces.length;
```

Ahora en numEnlaces, se almacenará el número de enlaces contenidos en el tercer párrafo.

4.8.2.Capturar ciertos eventos.

Para capturar el evento click en una página, se le puede asignar al evento ***document.onclick*** una función creada por el programador. si se desea saber dónde ha hecho clic el usuario en la página web, habría que colocar la siguiente línea:

```
document.onclick = hahechoclikaqui;
```

El script ***"hahechoclickaqui"*** será una función que se disparará cuando el usuario haga clic. El código de esta función puede ser el siguiente:

```
function hahechoclickaqui(elEvento){
    var evento= elEvento || window.event;
}
```

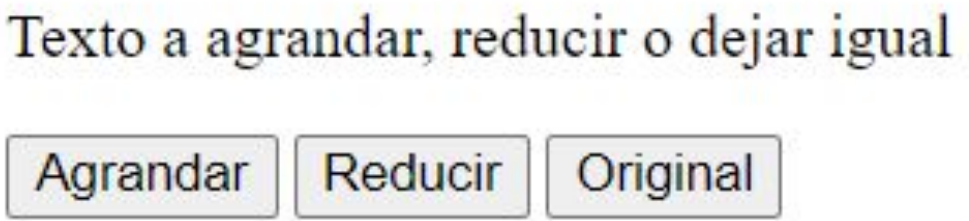
4.8.2.Capturar ciertos eventos.

Por lo tanto, en evento.clientX y evento.clientY, se almacenarán las coordenadas donde el usuario haya hecho clic.

4.9.Modificando el DOM.

Ya se ha visto en apartados anteriores cómo modificar la estructura del DOM añadiendo o eliminando nodos, accediendo a ellos y capturando ciertos eventos. En este apartado, se va a trabajar con mayor profundidad la modificación del DOM cambiando propiedades de los elementos como textos, mostrando y ocultando elementos, deshabilitando objetos o comprobando la información introducida en campos de texto.

4.9.1. Cambiar el tamaño del texto.



Ejemplo: ***Cambiar tamaño texto***

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

    <title>Cambiar Tamaño Texto</title>
</head>
<body>
    <p id="parrafo1">Texto a agrandar, reducir o dejar igual </p>
    <button type="submit" onclick="modificarTexto('aumentar',0.05);">Agrandar</button>
    <button type="submit" onclick="modificarTexto('reducir',0.05);">Reducir</button>
    <button type="submit" onclick="modificarTexto('original',0.05);">Original</button>
```

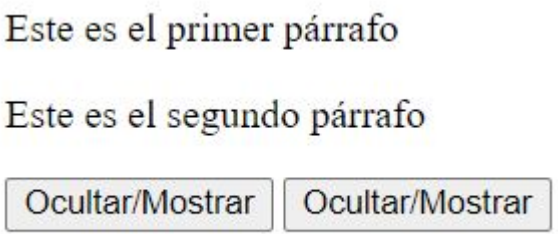
```
<script>
        var tamaño = 1;
            var tamañoOriginal = 1;
            function modificarTexto(elEvento, pixel){
                var elemento = document.getElementById('parrafo1');
                switch(elEvento){
                    case 'aumentar':
                if (tamaño > 2){
                    alert('Superado el tamaño máximo');
                    break;
                }else{
                    tamaño = tamaño + pixel;
                    break;
                }
```

Ejemplo: ***Cambiar tamaño texto***

```
case 'reducir':
                if (tamaño < .9){
                    alert('Superado tamaño mínimo');
                    break;
                }else{
                    tamaño = tamaño - pixel;
                    break;
                }
                case 'original':
                    tamaño = tamañoOriginal;
                    break;
                }
                elemento.style.fontSize = tamaño+'em';
            }
    </script>
</body></html>
```

4.9.2. Mostrar y ocultar elementos de una web.

Cambiar el estilo es muy útil en muchas aplicaciones JavaScript. A continuación, se presenta un ejemplo más sofisticado en el que se muestra o se oculta un elemento de una página web. Para ello, se han utilizado dos clases en CSS y, mediante JavaScript, se irá cambiando la clase al elemento del DOM:



Ejemplo: ***Mostrar/ocultar elementos web***

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

    <title>Mostrar y ocultar elementos de una web</title>
    <style>
        .oculto{
            visibility: hidden;
        }
        .visible{
            visibility: visible;
        }
    </style>
```

```
</head>
<body>
    <p id="parrafo1">Este es el primer párrafo</p>
    <p id="parrafo2">Este es el segundo párrafo</p>
    <button type="submit" onclick="ocultar('parrafo1');">Ocultar/Mostrar</button>
    <button type="submit" onclick="ocultar('parrafo2');">Ocultar/Mostrar</button>
    <script>
        function ocultar(parrafo){
            var elemento = document.getElementById(parrafo);
            if (elemento.className != 'oculto'){
                elemento.className = 'oculto';
                //también funcionaría la siguiente línea pero se ha preferido
                //modificar el estilo css
                //elemnento.style.visibility = 'hidden';
```

```
}else{
        elemento.className = 'visible';
        //también funcionaría la siguiente línea pero se ha preferido
        //modificar el estilo CSS
        //elemento.style.visibility = 'visible';
            }
        }
    </script>
</body>
</html>
```

De la manera anterior, se puede ocultar y mostrar un elemento de una página web de forma alternativa. Cambiar la clase CSS a cualquier elemento del DOM puede dar al programador muchos recursos para hacer formularios o páginas web más interactivas.

4.9.3.Deshabilitar objetos.

En ocasiones, los programadores necesitan deshabilitar ciertos elementos de la interfaz dependiendo de la interacción con el usuario. A continuación, se muestra el código de un botón que hace que se desactiven o activen ciertos elementos de la página web.

Ejemplo: ***Deshabilitar objetos.***

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Deshabilitar objetos</title>
</head>
<body>
    <input type="submit" value="Boton1">
    <input type="submit" value="Boton2">
    <input type="submit" value="Boton3">
    <input type="submit" value="Boton4">
    <input type="submit" value="Boton5">
    <br/><br/>
    <button type="submit" onclick="bloquear();" id="boton">Desactivar
todos</button>
```

Ejemplo: ***Deshabilitar objetos.***

```
<script>
    function bloquear(){
        var elementos = document.getElementsByTagName('input');
        if (elementos[0].className != 'desactivo'){
        for (var i = 0; i < elementos.length; i++){
            elementos[i].disabled= true;
            elementos[i].className ='desactivo';
        }
        document.getElementById('boton').innerHTML= "Activar todos";
```

```
}else{
            for (var i = 0; i < elementos.length; i++){
                elementos[i].disabled =false;
                elementos[i].className ='activo';
            }
        document.getElementById('boton').innerHTML = "Desactivar todos";
        }
    }
</script>
</body>
</html>
```

4.9.4. Comprobar que se introduce información en un campo de texto..

Muchas veces, la validación de un campo de texto implica que contenga información. A veces los usuarios introducen espacios en blanco que hacen que un programa no funcione de forma correcta. Se muestra un código que obliga a que el usuario introduzca algo en un campo de texto con id=”campo”.

4.9.4. Comprobar que se introduce información en un campo de texto..

```
valor = document.getElementById("campo").value;
if(valor == null || valor.length == 0 || /^\s+$/.test(valor)){
    return false;
}
```

Para que se dé por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducidos sea mayor que 0 y que no se hayan introducido solo espacios en blanco.

4.9.4. Comprobar que se introduce información en un campo de texto..

- La palabra reservada ***null*** es un valor especial que se utiliza para indicar “ningún valor”. Si el valor de una variable es ***null***, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.
- La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a 0 caracteres, esto es, que no sea un texto vacío.
- Por último, la tercera parte de la condición que es una expresión regular obliga a que el valor introducido por el usuario no solo esté formado por espacios en blanco. Esta comprobación se basa en el uso de ***expresiones regulares***. Por lo tanto, solo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

4.9.4. Comprobar que se introduce información en un campo de texto.

A continuación, se mostrará como se validaría un email. Obsérvese que la expresión regular en este caso es más compleja, puesto que hay que realizar más comprobaciones:

```
valor = document.getElementById( "campo").value:
if (!(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor))){
return false;
}
```

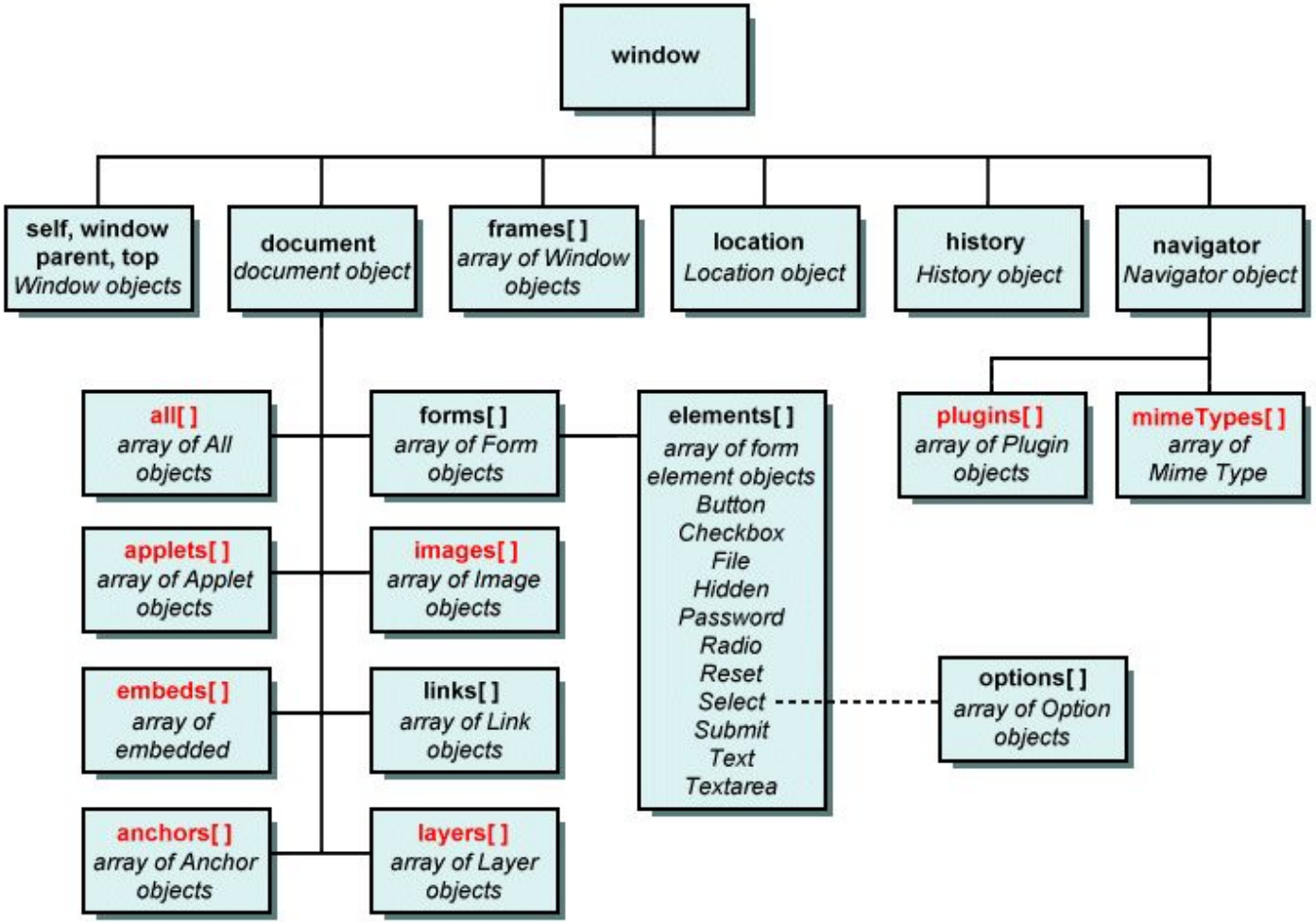
4.9.4. Comprobar que se introduce información en un campo de texto..

A veces, también el programador tiene que tratar con teléfonos. A continuación, se muestra la expresión regular para validar teléfonos:

```
valor = document.getElementById("campo").value
if (!(/^\d{9}$/.test(valor))){
    return false;
}
```

4.10. 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:


window
self, window
parent, top
Window objects
document
document object
frames[ ]
array of Window objects
location
Location object
history
History object
navigator
Navigator object
all[ ]
array of All objects
forms[ ]
array of Form objects
elements[ ]
array of form element objects
Button
Checkbox
File
Hidden
Password
Radio
Reset
Select
Submit
Text
Textarea
plugins[ ]
array of Plugin objects
mimeTypes[ ]
array of Mime Type
applets[ ]
array of Applet objects
images[ ]
array of Image objects
embeds[ ]
array of embedded
links[ ]
array of Link objects
options[ ]
array of Option objects
anchors[ ]
array of Anchor objects
layers[ ]
array of Layer objects

4.10. BOM.

El objeto raíz del BOM es el objeto ***window***, que está soportado por cualquier navegador y que, su vez, contiene otros objetos de menor rango jerárquico, como puede ser el objeto ***document*** u otros. Las diferencias entre BOM y DOM serían las siguientes:

4.10. BOM.

- 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.

4.10. BOM.

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

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

El resultado de ambas líneas de código sería el mismo.

4.10. BOM.

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

Ejemplo: ***BOM***

```
<!DOCTYPE html>
<html>
<body>
   <p id="dimension"></p>
   <script>
    var ancho = window.innerWidth;
    var alto = window.innerHeight;
    document.getElementById( 'dimension').innerHTML = "Ancho: " +
ancho + " --- Alto: " + alto;
   </script>
</body>
</html>
```

4.10.1. El objeto window.

Como se ha visto, el objeto ***window*** representa la ventana del navegador en la que se muestra un documento del DOM. El objeto ***window*** controla, entre otras cuestiones, las dimensiones de la ventana, las barras laterales de desplazamiento, la barra de estado, etc.

Actualmente, como los navegadores tienen varias pestañas, cada una de ellas tendrá un objeto window asociado. No obstante, métodos como ***window.resizeTo*** o ***window.resizeBy***, como es de entender, se aplicarán a la ventana de la aplicación y no a la pestaña específica.

4.10.2. El objeto location.

Un objeto muy útil del BOM es el objeto window.location. Algunas propiedades:

- window.location.href. Devuelve la URL de la página actual.
- window.location.hostname. Devuelve el dominio de la web.
- window.location.pathname. Devuelve el path y el fichero de la página actual.
- window.location.protocol. Devuelve e protocolo utilizado que debería ser o http: o https:
- window.location.assign() o location.assign(). Carga una nueva página web.
    - `window.location.assign("http://www.ieskursaal.es");`

4.10.3. El objeto history.

Es sabido que los navegadores tienen dos botones: uno carga la web anterior y otro que carga la siguiente web que se ha tecleado a la actual. Desde Java Script puede manejarse la historia del navegador con los métodos del objeto history:

- ***history.back()***. Cargará la página web visitada anterior a la actual.
- ***history.forward()***. Cargará la página web visitada siguiente a la actual.

4.10.4. El objeto navigator.

El objeto navigator tiene las siguientes propiedades:

- ***navigator.appName.***
- ***navigator.appCodeName.***
- ***navigator.platform.***

Las dos primeras contienen la aplicación que se está utilizando (por ejemplo, appName=Netscape y appCodeName=Mozilla) y en el tercero, la plataforma o sistema operativo utilizando (por ejemplo, Linux x86_64)

4.10.4. El objeto navigator.

¿Es posible pedir información al usuario de forma interactiva? No es normal, pero sí sería posible:

```
<!DOCTYPE html>
<html>
<body>
    <p id="cp"></p>
    <script>
        var codigopostal = prompt("Por favor introduzca su código postal",
"29700");
        if (codigopostal != null){
            document.getElementById("cp").innerHTML = "Su código postal es: "+
codigopostal;
        }
    </script>
</body>
</html>
```