Tema	JavaScript

1 Introducción

Es un lenguaje de programación interpretado y se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

El núcleo de JavaScript puede extenderse para varios propósitos, complementándolo con objetos adicionales, por ejemplo:

- **Client-side JavaScript** extiende el núcleo del lenguaje proporcionando objetos para controlar un navegador y su modelo de objetos (o DOM, por las iniciales de Document Object Model). Por ejemplo, las extensiones del lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, tales como clicks del ratón, ingreso de datos al formulario y navegación de páginas.
- **Server-side JavaScript** extiende el núcleo del lenguaje proporcionando objetos relevantes a la ejecución de JavaScript en un servidor. Por ejemplo, las extensiones del lado del servidor permiten que una aplicación se comunique con una base de datos, proporcionar continuidad de la información de una invocación de la aplicación a otra, o efectuar manipulación de archivos en un servidor.

Ejemplo de incluir javascript en el mismo documento HTML

```
<html>
<head>
</head>
<body>
   <script type="javascript">
        Alert("Un mensaje de prueba");
  </script>
</body>
</html>
```

```
<html>
<head>
        <script type="text/javascript">
                Function saludo(usuario){
                        alert("Hola"+usuario);
                }

        </script>
</head>
<body>
        <script type="text/javascript">
                saludo("pepe");
        </script>
</body>
</html>
```

Ejemplo de incluir javascript en el mismo documento HTML

```
<head>
        <script type="text/javascript" src="/js/codigo.js"></script>
</head>
```

En el archivo código.js se declaran las funciones de javascript. No es necesario utilizar <script></script>.

Variables

La sintaxis para definir e inicializar una variable en JavaScript es:

```
var nombre = valor;
```

siendo nombre el nombre de la variable y valor el valor que se le quiere asignar inicialmente. Una variable se puede definir sin inicializar, en cuyo caso tomará el valor undefined.

En JavaScript se distinguen los siguientes tipos de datos:

- **Number** para valores numéricos, independientemente de si son enteros o en coma flotante.
- **String** para cadenas de caracteres.
- **Boolean** para valores lógicos: verdadero (true) y falso (false).
- **Null** para el valor nulo o vacío (null).
- **Undefined** para el valor desconocido (undefined). Por ejemplo, cuando se define una variable y no se le asigna un valor de inicialización, esta toma el valor undefined. Debes tener en cuenta que en JavaScript los valores null y undefined no son equivalentes.
- **Symbol** para valores inmutables, similares a las constantes de otros lenguajes.
- **Object** para cualquier tipo de objeto, ya sea del propio JavaScript o definido por el usuario.

Ejemplos

```
var a = "cadena";
a = a.toUpperCase();    // a pasa a valer "CADENA"
a = a.length():  // a pasa a valer 6
a=a.indexof ("de") //a pasa a ver 2
a= a.substring(2,5); //a pasa a valer "den"
var a = 3;
a = a.toUpperCase();    // ERROR!!!
```

Algunas funciones básicas

- ✓ **Alert("Mensaje/n Otro mensaje"):** Muestra un mensaje por pantalla. /n es un salto de página.
- ✓ **document.write("Hola Mundo!"):** inserta cadena "Hola Mundo!" en página Web
- ✓ **document.writeln("Hola</br> Mundo!"):** inserta "Hola Mundo!" y un espacio al final. Como puedes comprobar puedes introducir etiquetas HTML.
- ✓ **var user=Prompt ("Escribe un nombre de usuario:"):** Sirve para pedir información al usuario y guardarlo en una variables .

Ejemplos de un programa

```
Document.writeln(3+2);  // Resultado 5
Var num= Prompt ("Inserta un número:");
Var num2=Prompt("Inserta el segundo número:");
Document.writeln("La variable num es "+ num + "la variable num2 es" +num2 +"</br>");
Var resultado= parseInt(num)+parseInt(num2);
Document.writeln("La resta  resta es "+ resultado) :
```

*Nota: El signo + sirve para concatenar una frase con el valor de la variable. La función **parseInt** transforma los tipos Cadena de caracteres (String) en números enteros.*

Sintaxis Javascript

Estructuras de control

Código	Denominación

`if(condición) {`
`  código true`
`}`
`else {`
`  código false`
`}`
	Estructura de control condicional

`(condición) ? valor_true : valor_false`	Operador de control condicional

`switch(expresión) {`
`  case condición_1:`
`  código condición_1`
`  [break;]`
`  …`
`  case condición_n:`
`  código condición_n`
`  [break;]`
`  default:`
`  código default`
`}`
	Estructura de control condicional basada en la evaluación del valor de una expresión

`for(ini; condición; actualización){`
`  código bucle`
`}`
	Bucle condicional

`while(condición) {`
`  código bucle`
`}`
	Bucle incondicional

`do {`
`  código bucle`
`} while(condicion)`
	Bucle incondicional

`break`	Salida de un bloque de código
`continue`	Parada de la ejecución de un bucle para volver a evaluar la condición de parada

`try {`
`  código js`
`}`
`catch(err) {`
`  código que gestiona el error`
`}`
`finally {`
`  código a ejecutar haya o no error`
`}`
	Captura y gestión de errores en el código

Operadores

Operador	Denominación
`=`	Asignación
`+, -, *, /, %`	Operadores aritméticos (actúan sobre valores numéricos)
`++, --`	Incremento, decremento
`+=, -=, *=, /=, %=`	Operación aritmética + Asignación
`+, +=`	Concatenación de Strings
`==, !=, >, <, >=, <=`	Comparadores de valor
`===` (igual valor e igual tipo) `!==` (distinto valor y distinto tipo)	Comparadores de valor y tipo
`&&, ||, !`	Operadores lógicos

Comentarios

Otro aspecto de sintaxis común entre *C/Java* y *JavaScript* es la forma en que se definen los comentarios, es decir, texto que no va a ser interpretado:

Comentarios
`// Esto es un comentario de texto de una sola línea.`
`/* Los comentarios como este, entre barra y asterisco, pueden ocupar más de una línea. */`

Eventos HTML

La sintaxis general en el documento HTML para asignar una funcionalidad determinada a un elemento HTML es:

```
<tag onXXX="tu código para XXX">
    ...
</tag>
```

Con este código estaríamos indicando al navegador que cuando el elemento definido por la etiqueta (tag) HTML (una imagen —<img>—, el cuerpo del documento —<body>—, una tabla —<table>—, etc.) "lance" el evento XXX, se ejecute el código que se le proporciona en cada caso. Como puedes ver, esta asignación es equivalente a la asignación del valor de cualquier propiedad HTML, por lo que, al igual que ocurre con el resto propiedades, el valor que se da a la propiedad puede ir entre comillas simples (') o dobles (").

Estos son algunos de los eventos HTML que más habitualmente se procesan en las aplicaciones web.

- ✓ ***click*:** se produce cuando el usuario hace click sobre el elemento HTML.
- ✓ ***mouseover*:** se produce cuando el usuario entra con el ratón dentro del elemento HTML.
- ✓ ***mouseout*:** se produce cuando el usuario sale con el ratón del elemento HTML.
- ✓ ***focus*:** se produce cuando el elemento HTML recibe el foco.
- ✓ ***blur*:** se produce cuando el elemento HTML pierde el foco.
- ✓ ***change*:** se produce cuando el valor del elemento HTML cambia (para <*input*>, <*select*> y <*textarea*>).

- ✓ ***keydown/keypress*:** se producen cuando el usuario presiona una tecla estando dentro del elemento HTML. Aunque realmente son eventos distintos, en la mayoría de las ocasiones se pueden considerar equivalentes.
- ✓ ***keyup*:** se produce cuando el usuario suelta una tecla estando dentro del elemento HTML.
- ✓ ***submit*:** se produce cuando se hace el submit de un formulario (para <*form*>).
- ✓ ***load***: se produce cuando finaliza la carga del elemento HTML. Típicamente se utiliza para validar la carga del documento.
- ✓ ***unload***: se produce cuando se descarga la página (para <*body*>).
- ✓ ***error***: se produce cuando se produce un error al cargar un fichero externo.

Ejemplo

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

<script>
        function mOver(obj) {
         obj.innerHTML = "Thank You"
        }

        function mOut(obj) {
        obj.innerHTML = "Mouse Over Me"
        }
function mDown(obj) {
  obj.style.backgroundColor = "#1ec5e5";
  obj.innerHTML = "Release Me";
}

function mUp(obj) {
  obj.style.backgroundColor="#D94A38";
  obj.innerHTML="Thank You";
}
</script>
</head>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;margin:20px;">
Mouse Over Me</div>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;margin:20px;">
Click Me</div>

</body>
</html>
```

2 DOM- Document object model

Cuando un navegador carga un documento HTML crea el modelo **HTML DOM** (*Document Object Model*) del documento. El modelo HTML DOM es un estándar W3C que define clases que representan:

- **Elementos HTML**, por ejemplo, una tabla (etiqueta <table>), un párrafo (etiqueta <p>) e incluso el propio documento HTML.
- **Propiedades de los elementos HTML**, por ejemplo, el borde de una tabla (atributo *border*) o el ancho de una imagen (atributo *width*).
- **Métodos** para acceder a los objetos que representan los elementos HTML.
- **Eventos** para capturar la interacción del usuario de la aplicación con los elementos HTML en la pantalla del navegador.

De esta forma, cuando el navegador recibe e interpreta un documento HTML, crea un objeto del tipo correspondiente por cada uno de los elementos HTML que contiene, inicializando y dando valor a sus propiedades en función del valor estático de los atributos indicados en el código HTML. En esta unidad aprenderemos a trabajar con el modelo DOM de un documento HTML para implementar funcionalidad *JavaScript* que permita:

- Crear, modificar o eliminar dinámicamente elementos HTML.
- Modificar dinámicamente el valor de las propiedades de los elementos HTML del documento.
- Modificar dinámicamente los atributos CSS de un determinado elemento.
- Interactuar con el usuario de la aplicación.

El árbol DOM

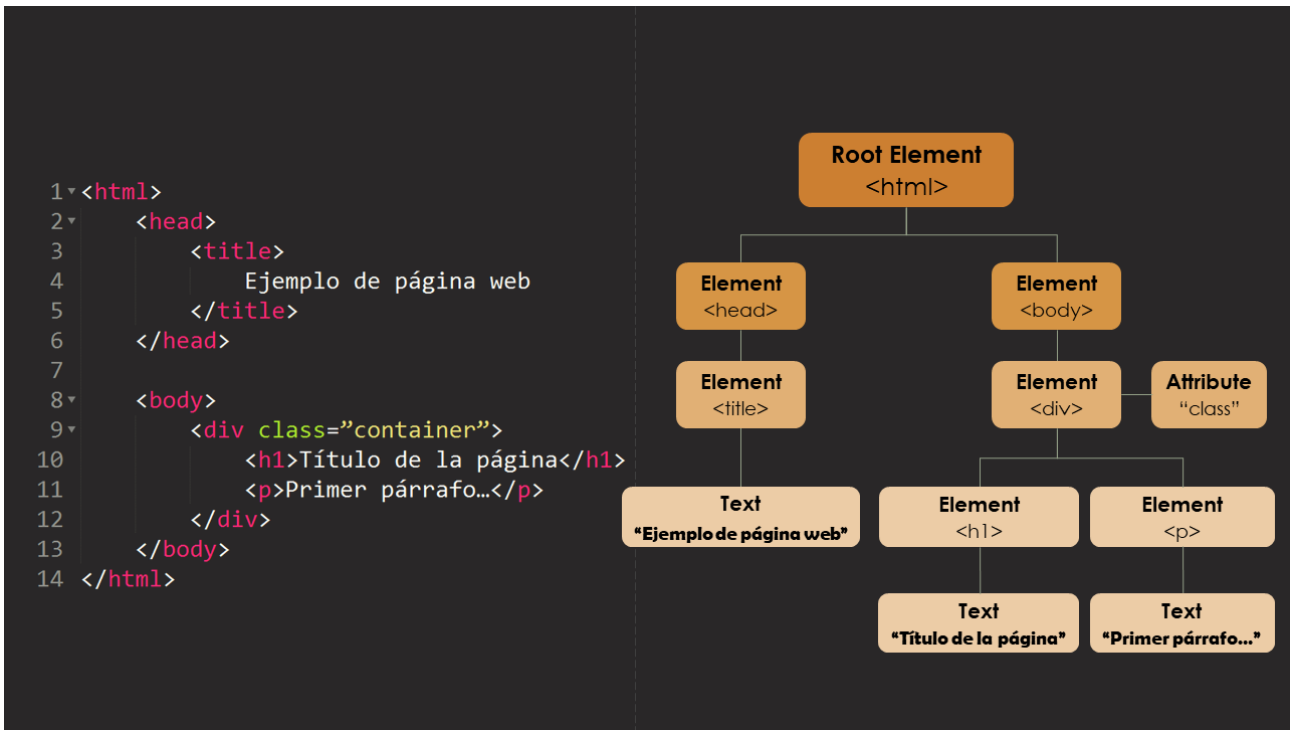
El modelo HTML DOM define una estructura arbórea de objetos que refleja la naturaleza jerárquica del lenguaje HTML, de ahí que muchas veces hablemos del **árbol DOM** del documento. En la figura de ejemplo se muestra cuál sería el árbol DOM (a la derecha) del primer documento HTML que visteis en el curso (a la izquierda).

El **elemento raíz del árbol** representa el tag <html> del documento. Las primeras semanas del curso visteis que dentro de este tag se definen dos bloques de primer nivel, uno delimitado por la etiqueta <head> con metainformación del documento, y otro delimitado por la etiqueta <body> con el cuerpo del documento.

Teniendo esto en cuenta, el nodo raíz de un árbol DOM (<html>) contiene dos **nodos hijos**, uno que representa el elemento <head> (etiqueta <head> y todo su contenido en el documento) y el otro el elemento <body> (etiqueta <body> y todo su contenido en el documento).

Para construir el árbol DOM del documento, esta definición se aplica **de manera recursiva** hasta llegar a las etiquetas HTML que en su interior ya no contienen nuevas

etiquetas HTML, como ocurre por ejemplo con las etiquetas <title>, <h1> y <p> en el documento de la figura.



Algo importante que debes tener en cuenta a la hora de trabajar con HTML DOM es que, como se ilustra en la figura, **todos los componentes del documento HTML** se convierten en un **nodo** del árbol DOM. Aquí vamos a centrarnos en tres tipos de nodos:

- **Element**, representa un elemento HTML del documento (*title*, *body*, *div*, *p* o cualquier otro elemento identificado por una etiqueta HTML).
- **Attribute**, representa un atributo de un elemento HTML (*class*, *id*, *width*...). Siempre son hijos de un nodo de tipo *Element*.
- **Text**, representa el contenido (información) de un elemento HTML. Siempre son hijos de un nodo de tipo *Element* y son la hoja de una rama del árbol DOM.

El punto de acceso al árbol DOM de cualquier documento HTML que ha sido interpretado por un navegador web es un objeto de tipo ***HTMLDocument*** que se almacena en la variable predefinida ***document***. Este objeto se crea de forma automática siempre que el navegador carga un documento HTML y es accesible desde cualquier parte del código *JavaScript*.

- ***document.documentElement*** representa al nodo raíz.
- ***document.head*** representa el nodo head, hijo del nodo raíz.
- ***document.body*** representa el nodo body, también hijo del nodo raíz.

Métodos para localizar los elementos HTML

Estos son los posibles criterios de búsqueda de nodos en el árbol DOM, así como el método de la variable *document* al que habría que invocar para realizar la búsqueda:

- Para buscar por **identificador** de elemento HTML (atributo *id*) utilizaremos el método *getElementById("id")* que devuelve el elemento con el id que se le pasa como parámetro o null en caso de no existir ninguno.
- Para buscar por el **nombre** del elemento HTML (atributo *name*) utilizaremos el método *getElementsByName("nombre")* que, dependiendo del navegador, devuelve un objeto de tipo *NodeList* o uno de tipo *HTMLCollection* con todos los elementos cuyo nombre coincide con el que se le pasa como parámetro.
- Para búsquedas de elementos de un determinado **tipo** utilizaremos el método *getElementsByTagName("etiqueta")* que, dependiendo del navegador, devuelve un objeto *NodeList* o *HTMLCollection* con todos los elementos HTML del tipo (etiqueta) que se le pasa como parámetro.
- Finalmente, para buscar los nodos de una **clase CSS** determinada (atributo *class*) utilizaremos el método *getElementsByClassName("clase")* que, dependiendo del navegador, devuelve un objeto *NodeList* o *HTMLCollection* con todos los elementos HTML a los que se les haya asignado la clase que se le pasa como parámetro.

Diferencias entre *NodeList* y *HTMLCollection*

Las variables de tipo ***NodeList*** representan un conjunto ordenado de nodos del árbol DOM, mientras que una de la clase ***HTMLCollection***representa un conjunto ordenado de elementos HTML. En muchos casos, ambas clases son equivalentes en cuanto a la información que contienen y los métodos implementados en el motor *JavaScript* del navegador las utilizan de manera indistinta. La principal diferencia entre un *NodeList* y una *HTMLCollection* son los mecanismos que ofrecen para acceder a los elementos que contienen.

Una variable de tipo ***NodeList*** solo permite el acceso a partir del **índice** del elemento en la lista, es decir, que ofrece un mecanismo de acceso análogo al de un array. La sintaxis también es análoga a la que se utiliza para acceder a un array que, a su vez, es también análoga a la sintaxis para acceder a arrays estáticos en *C* o *Java*. Así, *lista[n]* devuelve el nodo n-ésimo de la lista, teniendo en cuenta que el índice del primer elemento es 0 y no 1. Para saber el tamaño de la lista, un *NodeList* dispone del atributo de solo lectura *length*.

Por su parte, la clase ***HTMLCollection*** también permite acceder a sus elementos por **índice**, para lo que también dispone del atributo *length*. Sin embargo, los objetos de tipo *HTMLCollection* también ofrecen mecanismos de acceso a sus elementos por **id**, análogos a los de un diccionario. De esta forma, una colección de elementos HTML dispone de tres “métodos” para acceder a sus elementos:

- *item(n)*: permite el acceso a los elementos de la colección por índice, devolviendo el elemento n-ésimo de la colección. De nuevo, se debe tener en cuenta que el primer elemento de la colección es el elemento 0.

- *namedItem("id")*: permite acceder a los elementos de la colección como si fueran un diccionario, devolviendo el/los elementos cuyo nombre es igual al id que se le pasa como parámetro.
- *[ítem]*: es una forma abreviada de acceder a los métodos anteriores de forma que *c[0]* devuelve el primer elemento de la colección, mientras que *c["id"]* devuelve el/los elementos cuyo nombre es "id".

Propiedades innerHTML o OuterHTML

Como has podido comprobar en el ejercicio anterior, modificar la estructura del documento HTML con los métodos y propiedades de *HTMLDocument* y *HTMLHtmlElement* puede requerir múltiples llamadas a diferentes métodos para construir las relaciones entre los nodos del árbol DOM y asignar valor a sus atributos.

Otra forma de hacer esto mismo es utilizar la propiedad de lectura/escritura ***innerHTML*** de los nodos de tipo *Element*. Con esta propiedad tenemos acceso al contenido HTML de un elemento, es decir, al código HTML contenido dentro de ese elemento. La propiedad *outerHTML* que también da acceso al contenido HTML del elemento, la diferencia entre ambas es que mientras *outerHTML*incluye la etiqueta que representa al propio elemento, en *innerHTML* solo se incluye el código HTML de sus descendientes.

Ejemplo

```
<html>
   <head>
      <meta charset="utf-8"/>

      <script type="text/javascript">
         function rellenarTextAreas() {
            document.getElementById("outer").value = document.getElementById("contenedor").outerHTML;
            document.getElementById("inner").value = document.getElementById("contenedor").innerHTML;
         }
      </script>
   </head>
   <body>
      <div id="contenedor">
         Este es un ejemplo en el que ilustrar <b>la diferencia</b> entre las propiedades <i>outerHTML</i> e
<i>innerHTML</i>
      </div>
      <div id="salida">
         <textarea id="outer" style="width:40%"></textarea>
         <textarea id="inner" style="width:40%"></textarea>
      </div>
      <input type="button" onclick="rellenarTextAreas()" value="Ver diferencias" />
   </body>
</html>
```

Ejemplos DOM

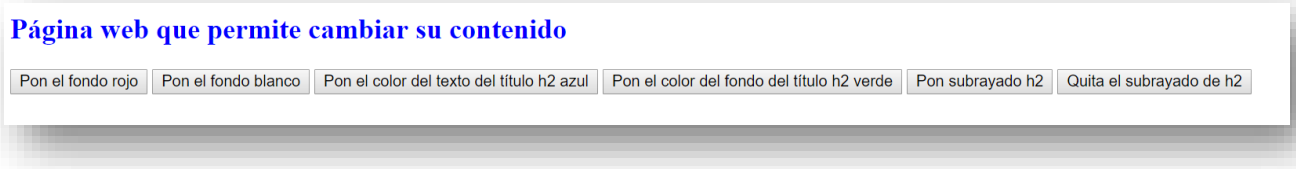
Ejemplo resuelto 1


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo DOM</title>
    <script type="text/javascript">
      function img1(){
      document.getElementById("imagen").src="1.png";
      }
      function img2(){
      document.getElementById("imagen").src="2.png";
      }

    </script>
  </head>
  <body>
    <h2>Página web que permite cambiar la imagen</h2>
    <img src="1.png" id="imagen" height="225" width="225">
    <p><button onclick="img1()">Imagen 1</button>
      <button onclick="img2()">Imagen 2</button>
    </p>
  </body>
</html>
```

Ejemplo resuelto 2


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo DOM</title>
    <script type="text/javascript">
      function cambiaheader1(){
      document.getElementById("cabecera").innerHTML="<h1>Nuevo h1</h1><p>Nuevo
párrafo. Todo esto dentro de un header</p>"
      }
  function cambiaheader2(){
      document.getElementById("cabecera").innerHTML=""
      }

    </script>
  </head>
  <body>
    <h2>Página web que permite cambiar su contenido</h2>
    <header id="cabecera"></header>
    <p><button onclick="cambiaheader1()">Cambia el contenido del header</button>
<button onclick="cambiaheader2()">Vaciar el header</button>
    </p>
  </body></html>
```

Ejemplo resuelto 3




```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo DOM</title>
    <script type="text/javascript">
      function ponrojo(){
document.body.style.backgroundColor="red";
```

```
        }
   function ponblanco(){
document.body.style.backgroundColor="white";
        }
   function colortextoh2azul(){
document.getElementById("titulo2").style.color="blue";
        }
            function colorfondoh2verde(){
document.getElementById("titulo2").style.backgroundColor="green";
        }
            function subrayah2(){
document.getElementById("titulo2").style.textDecoration="underline";
        }
            function quitasubrayah2(){
document.getElementById("titulo2").style.textDecoration="none";
        }

     </script>
   </head>
   <body>
     <h2 id="titulo2">Página web que permite cambiar su contenido</h2>
     <p><button onclick="ponrojo()">Pon el fondo rojo</button>
<button onclick="ponblanco()">Pon el fondo blanco</button>
<button onclick="colortextoh2azul()">Pon el color del texto del título h2 azul</button>
<button onclick="colorfondoh2verde()">Pon el color del fondo del título h2 verde</button>
<button onclick="subrayah2()">Pon subrayado h2</button>
<button onclick="quitasubrayah2()">Quita el subrayado de h2</button>

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

Ejemplo resuelto 4 (Al pasar el ratón por encima del botón sin pulsar)

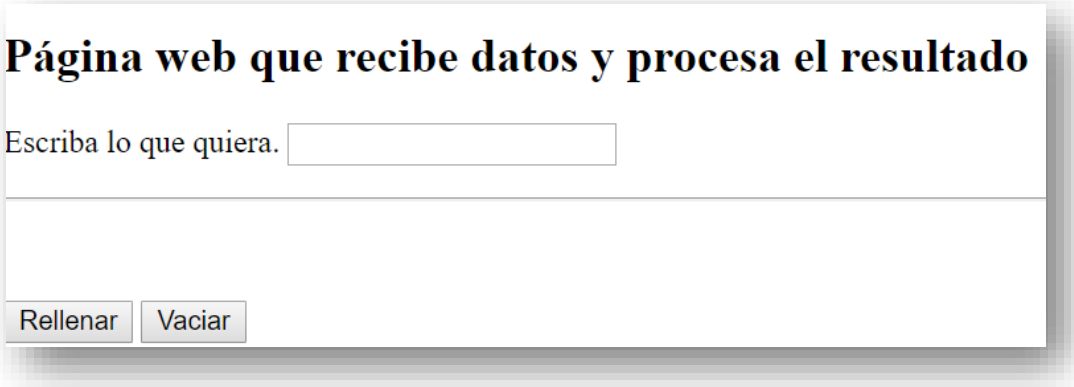
```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo DOM</title>
    <script type="text/javascript">
      function ponBorde(){
document.getElementById("imagen").style.border="5px dotted red";
       }
       function quitaBorde(){
      document.getElementById("imagen").style.border="none";
       }
       function hazvisible(){
document.getElementById("imagen").style.visibility="visible";
       }
       function hazinvisible(){
      document.getElementById("imagen").style.visibility="hidden";
       }
   function hazPeque(){
      document.getElementById("imagen").width="25";
      document.getElementById("imagen").height="25";
     }
    </script>
  </head>
  <body>
    <h2>Página web que permite cambiar la imagen</h2>
    <img src="1.png" id="imagen" height="225" width="225" onclick="hazPeque()">
    <p>Pincha en la imagen para hacerla más pequeña.</p>
<button  onmouseover="ponBorde()" onmouseout="quitaBorde()">Poner/quitar borde</button>
<button onmouseover="hazinvisible()" onmouseout="hazvisible()">Mostrar/ocultar la
imagen</button>
</body></html>
```

3 Ejemplos Entrada y Salida

Ejemplo resuelto 1


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo E/S</title>
    <script type="text/javascript">
      function responder(){
      var t1=document.getElementById("texto1").value;
      document.getElementById("p1").innerHTML="En casilla de texto texto1 has escrito: "+t1;
      }
    </script>
  </head>
  <body>
    <h2>Página web que recibe datos y procesa el resultado</h2>
    <p><label>Introduzca un texto cualquiera:</label>
      <input id="texto1" type="text"></input>
    </p>
    <hr>
    <p id="p1"> </p>
    <button  onclick="responder()">Responder</button>
  </body>
</html>
```

Ejemplo resuelto 2



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo E/S</title>
    <script type="text/javascript">
      function rellenar(){
      document.getElementById("texto1").value="Manolo";
      document.getElementById("p1").innerHTML="Se acaba de poner en la casilla de texto:
Manolo";
      }
      function vaciar(){
      document.getElementById("texto1").value="";
      document.getElementById("p1").innerHTML="Se acaba de vaciar la casilla de texto";
      }
    </script>
  </head>
  <body>
    <h2>Página web que recibe datos y procesa el resultado</h2>
    <p>
      <label>Escriba lo que quiera.</label>
      <input id="texto1" type="text" value=""></input>
    </p>
    <hr>
    <p id="p1"> </p>
    <button onclick="rellenar()">Rellenar</button>
    <button onclick="vaciar()">Vaciar</button>
  </body>
</html>
```

Ejemplo resuelto 3

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Plantilla E/S</title>
    <script type="text/javascript">
      function sumar(){
      var n1=parseInt(document.getElementById("texto1").value);
      var n2=parseInt(document.getElementById("texto2").value);
      document.getElementById("p1").innerHTML=n1+n2;
      }

      function concatenar(){
      var n1=document.getElementById("texto1").value;
      var n2=document.getElementById("texto2").value;
      document.getElementById("p1").innerHTML=n1+n2;
      }

    </script>
  </head>
  <body>
    <h2>Página web que recibe datos y procesa el resultado</h2>
    <p><label>Introduzca dos números:</label>
      <input class="w3-input w3-border" id="texto1" type="text"></input>
      <input class="w3-input w3-border" id="texto2" type="text"></input>
    </p>
    <hr>
    <p id="p1"> </p>
    <button class="w3-btn w3-blue" onclick="sumar()">Sumar</button>
    <button class="w3-btn w3-blue" onclick="concatenar()">Concatenar</button>
  </body>
</html>
```

4 Validar formularios en Javascript

Funciones utilizadas para validad formularios

- **eval(string)**
  Esta función recibe una cadena de caracteres y la ejecuta como si fuera una sentencia de Javascript.
- **parseInt(cadena)**
  Recibe una cadena. Devuelve un valor numérico resultante de convertir la cadena en un número en la base indicada.
- **parseFloat(cadena)**
  Convierte la cadena en un número y lo devuelve.
- **escape(carácter)**
  Devuelve un el carácter que recibe por parámetro en una codificación ISO Latin 1.
- **unescape(carácter)**

Hace exactamente lo opuesto a la función escape.

- **isNaN(número)**
  Devuelve un boleano dependiendo de lo que recibe por parámetro. Si no es un número devuelve un true, si es un numero devuelve false.

Fichero HTML (Vamos a comprobar si los campos de texto del nombre, edad y email son validos):

Este formulario se ejecutará cuando el usuario envie los datos a tráves del botón del formulario. Si algunos de los campos no se validan correctamente el usuario deberá introducir los datos correctamente y volver a darle al botón.

Nota: Accedemos al formulario y a sus respectivos campos utilizando el valor del atributo name

Ejemplo: document.getElementById("nombre").value (Valor del campo nombre en el formulario)

Fichero HTML:

```
<head>
<script type="text/javascript" src="validar.js"></script>
……..
</head>
<form id="formulario" name="formulario" method="post" action="mailto:correo@correo.com" >

……
<input type="button" name="envio" value="Enviar!" onclick="validacion()" />
……..
</form>
```

Fichero validar.js

```
function validacion(){
 //Asignar valores formulario a variables
var nom=document.getElementById("nombre").value;
var dir=document.getElementById("direccion").value;
var edad = parseInt(document.getElementById("edad").value);
var email=document.getElementById("email").value;
var tlf=document.getElementById("movil").value;

 //NOMBRE
nom=nom.trim();//Quitar espacios al principio y al final
if (nom=="" ){
  alert("Tienes que introducir un Nombre");
          document.getElementById("nombre").focus();
          return false;
          }

   //DIRECCION
  dir=dir.trim();
if (dir=="" ){
          alert("Tienes que introducir una Direccion");
          document.getElementById("direccion").focus();
          return false;
          }
  //EDAD
    //comprueba si es un valor numérico. No hago trim porque parseInt transformado en numérico
     if (isNaN(edad)){
       alert("Tiene que introducir un número entero en su edad.");
       document.getElementById("edad").focus();
      return false;}
      else if(edad<18 || edad>=60){
        alert("Debe ser mayor de 18 y menor de 60 años.");
```

```
        document.getElementById("edad").focus();
        return false;
      }

  //EMAIL
  // Comprueba si es un email valido y si incluye espacios no debidos
  email=email.trim();
  if( !(/^[-\w.%+]{1,64}@(?:[A-Z0-9-]{1,63}\.){1,125}[A-Z]{2,63}$/i.test(email)) ) {

   alert("Tienes que introducir una Email válido");
            document.getElementById("email").focus();
            return false;
          }
 //Móvil
 tlf=tlf.trim();
 // expresión regular de móviles de 9 cifras que empiecen por 6
 if( !(/^[6]{1}([0-9]+){8}$/i.test(tlf)) ) {

  alert("Tienes que introducir una Móvil válido");
 document.getElementById("movil").focus();
 return false;
 }
          return true;
}
```