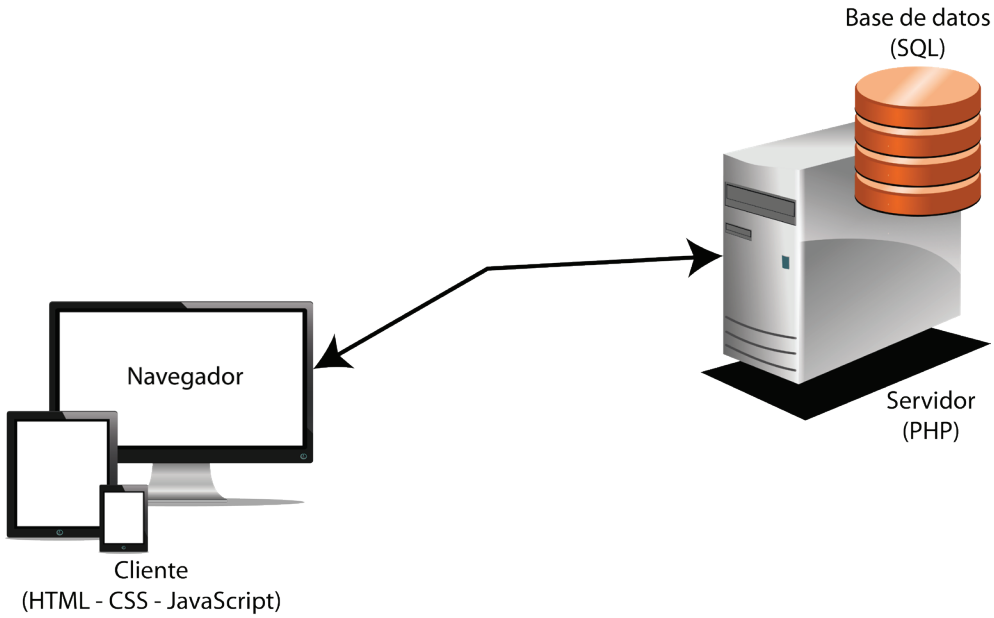
INTRODUCCIÓN

Cuando hablamos de web, es obligatorio dividirlo en dos partes bien diferenciadas: Cliente y servidor. El cliente es quien solicita el servicio y el servidor se encarga de satisfacer estas demandas.

Y ya si nos centramos en sitios web y su programación, el área de cliente es la que se desarrolla en el navegador, es decir, contendría la estructura de la página web desarrollada mediante un lenguaje llamado **HTML (HyperText Markup Language)**, a la que debemos darle un diseño mediante hojas de estilo o **CSS (Cascading Style Sheets)**. Con estas dos herramientas se crearía una página estática, propia de otros tiempos. Si queremos añadirle interactividad y dinamismo necesitaremos el lenguaje de programación por excelencia que es **Javascript**.

En el área de servidor contaremos con los servicios que vayan a ser requeridos por los clientes con sus peticiones. Habitualmente, como mínimo, se tendrá una base de datos y un programa con el que el usuario pueda acceder y tratar la información guardada. Para crear y manipular los datos utilizaremos **SQL (Structured Query Language)** y el lenguaje de programación con el que se establecerá la conexión con el usuario será **PHP (Hypertext Preprocessor).**

Si conseguimos unir todas estas características y potenciarlas seremos capaces de desarrollar una web que responda a todas las expectativas de los usuarios.



Cliente/Servidor

Un claro ejemplo de esta interacción entre cliente y servidor lo podemos encontrar en cualquier página de comercio electrónico. Entramos a ella mediante un navegador, elegimos el artículo que queremos comprar buscando en una base de datos que se encuentra en servidor. Este servidor mediante un programa nos responderá, por ejemplo, si está disponible o no, y la posibilidad de comprarlo disminuyendo así de su stock.

A lo largo de este libro desarrollaremos un proyecto de creación de una página web de e-commerce en el que incluiremos todas las opciones para entender la complejidad y la interrelación entre ellos.

1

PROGRAMACIÓN CLIENTE

1.1 PROGRAMACIÓN CLIENTE

Tal como hemos comentado en la introducción a este libro, el modelo cliente es la programación que se ejecuta en el navegador. HTML y CSS proporcionan lo que es la estructura y el diseño al sitio web, pero añadiendo el lenguaje de programación Javascript conseguimos el dinamismo y la flexibilidad que hará nuestra web mucho más atractiva.

Al cargarse al mismo tiempo que el HTML en el navegador y residir en el cliente, hace que JavaScript sea rápido a la hora de obtener una respuesta el usuario y que pueda seguir funcionando si se produce una desconexión temporal a Internet.

Este curso se centra en la programación del sitio web mediante Javascript, pero en el siguiente apartado daremos una pincelada de HTML y CSS, como base principal.

1.2 PROGRAMACIÓN DE PÁGINAS WEB

Antes de comenzar a estudiar el lenguaje de programación Javascript, haremos una breve reseña de utilización del lenguaje HTML y de las hojas de estilo

CSS, ya que cuando se habla de programación web se suele hacer referencia a todo el conjunto aunque HTML y CSS no sean lenguajes de programación en sí.

Lo primero que tenemos que hacer a la hora de crear una página web es definir su estructura. Para ello utilizaremos HTML. La estructura básica suele tener el formato que se muestra en la ilustración, aunque no es de estricto complimiento.

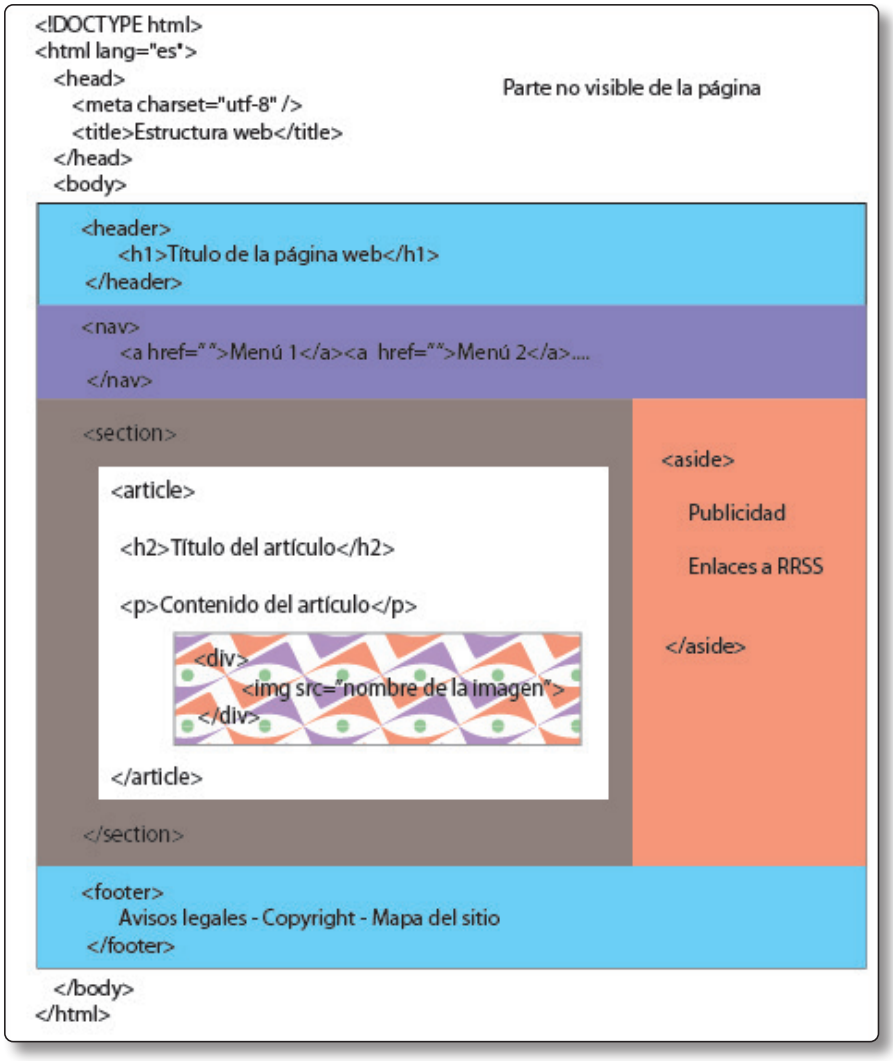


Figura 1.1. Estructura de una página web

Una vez definida la estructura del sitio web, hay que dotarle de estilo a la misma para que sea atractiva para el usuario. Esto se puede realizar de varias formas, de forma directa o embebida, aunque es recomendable incorporar uno o varios archivos de extensión .css y enlazarlos con el HTML en la zona <head>.

```
<head>
<meta charset="utf-8">
<title>Título de la página</title>
<link rel="stylesheet" href="estilos.css">
</head>
```

En este archivo, se va dando formato a cada sección de HTML, indicando si tiene bordes o no, color de fondo, bordes redondeados, color de la letra, sombras y muchas opciones más.

Ejemplo:

Con lo que siguiendo con nuestro ejemplo anterior, contaríamos con dos archivos:

index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Título de la página </title>
<link rel="stylesheet" href="estilos.css">
</head>
<body>
<header>
    <h1>Título principal</h1>
</header>
<nav>
   <ul>
```

```
        <li><a href="">Menu 1</a></li>
        <li><a href="">Menu 2</a></li>
        <li><a href="">Menu 3</a></li>
        <li><a href="">Menu 4</a></li>
    </ul>
</nav>
<section>
 <article>
          <h2>Título del artículo</h2>
          <p>Contenido del artículo ... </p>
            <div>
                <img src="imagen.jpg">
            </div>
        </article>
    </section>
    <aside>
        <p>Publicidad </p>
        <p>Enlaces a RRSS</p>
    </aside>
    <footer>
        Avisos legales - Copyright - Mapa del sitio
    </footer>
</body>
</html>
```

estilos.css

```
header{
    background-color: cornflowerblue;
    height: 50px;
    padding: 5px;
}

ul {
  list-style-type: none;
```

```
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: blueviolet;
}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover {
  background-color: #111;
}

aside{
    float: right;
    width: 29.9%;
    background-color: burlywood;
    text-align: center;
    min-height: 520px;
    margin=1%;
}
section{
    float: left;
    width: 69.9%;
    background-color: darkkhaki;
    min-height: 520px;
}
```

```
article{
    background-color: aliceblue;
    margin: 2%;
}

article p{
    padding: 1%;
}

img{
    margin: 2%;
    border: 2px solid black;
    border-radius: 5px;
    box-shadow: 10px 10px 10px rgba(0, 0, 0, .5);
    width: 90%;
}

h1, h2{
    text-align: center;
}

footer{
  /*  float: left;
    width: 100%;*/
    background-color: darkgray;
    text-align: center;
   /* min-height: 0px;*/
    padding: 5px;
}
```

que nos mostraría la siguiente página básica:

PROYECTO

Comenzaremos creando un sitio web para la venta de un producto. Se recomienda que no se realice de muchos, sino que se tomen dos o tres productos de ejemplo.

Por ejemplo, puede ser una página de venta de zapatos, ropa o electrodomésticos.

Para este sitio, vamos a crear tres páginas sencillas:

- Índice
- Quienes somos – Contacto
- Productos

Recuerda que la página debe ser sencilla, responsive y se deben agregar enlaces a redes sociales.

1.3 INTRODUCCIÓN A JAVASCRIPT

Podemos decir que Javascript es un lenguaje basado en scripts, es decir, pequeños programas sencillos para conseguir un resultado que le va a proveer a la página las opciones dinámicas que necesita para hacerla más interactiva con el usuario brindándole animaciones, acciones que se activan al pulsar botones o mensajes emergentes de aviso.

JavaScript es un lenguaje interpretado, es decir, no necesita ser compilado y es interpretado por el navegador, con la desventaja que, en algunas ocasiones, podremos obtener resultados diferentes si cambiamos de navegador.

Además es un lenguaje orientado a objetos, que hacen que el código sea más claro e intuitivo. Como su nombre indica, se basa en scripts o guiones que se insertan en el lenguaje HTML. Esto se puede hacer de tres formas, igual que ocurre con css:

- Insertándolo dentro del contenido html. Si se va a introducir de esta forma, es conveniente saber que el navegador lee HTML de forma lineal.

```
…
<h1> Titulo principal </h1>
<script type="text/javascript">
alert("Bienvenido a nuestra página web");
</script>
…
```

- Definiendolo en el head del html

```
…
<head>
<title>Título de la página</title>
<script type="text/javascript">
   alert("Bienvenido a nuestra página web");
</script>
```

```
</head>
…
```

- Creando un fichero externo .js

HTML

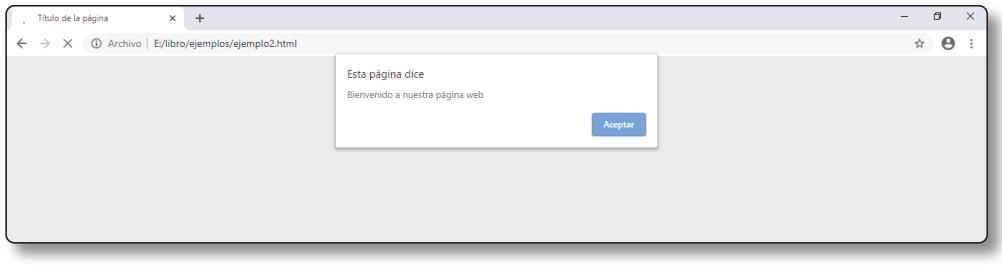
```
...
<head>
<meta charset="utf-8">
<title>Título de la página</title>
<link rel="stylesheet" href="estilos.css">
<script type="text/javascript" src="script.js"></script>
</head>
```

Archivo script.js

```
alert("Bienvenido a nuestra página web");
```

Al igual que en css, se recomienda siempre la utilización de uno o varios archivos externos para la claridad y el mantenimiento del código. Por ejemplo, si necesitamos una calculadora, crearemos un archivo con el nombre calculadora.js donde colocaremos todas las funciones referentes a la misma.

El navegador aceptará el código, sea cual sea la opción anterior elegida, se nos mostrará el siguiente mensaje antes de mostrarnos la página web



Consideraciones previas

Hay algunas consideraciones que tenemos que tener en cuenta antes de comenzar a trabajar con JavaScript:

- JavaScript distingue entre mayúsculas y minúsculas.
- Aunque no es obligatorio, es recomendable finalizar las sentencias en ;
- Como en HTML no se tienen en cuenta los espacios en blanco o las nuevas líneas.
- Los comentarios se colocan mediante // si es un comentario de una línea y entre /* y */ si es un comentario multilínea
- Para evitar errores de código, antes de empezar a escribir el código se coloca las etiquetas de comentario multilínea de HTML <!-- //-->, así si el navegador no entiende la etiqueta <script>, la ignora siguiendo en código HTML, con lo que todo el script quedará envuelto en un comentario.

```
<script>
<!--
   Código de Javascript
//-->
</script>
```

- Utilizamos la etiqueta <noscript> cuando el navegador no puede ejecutar JavaScript

```
<noscript> Para ver correctamente esta página web,
necesita un navegador que ejecute Javascript
</noscript>
```

1.4 FUNDAMENTOS DE PROGRAMACIÓN

Tenemos dos formas de realizar un programa, utilizando la programación estructurada o bien utilizando la programación orientada a objetos. Javascript es un lenguaje que puede utilizar cualquiera de las dos metodologías.

La programación estructurada, tal como su nombre indica, es una estructura que se va ejecutando paso a paso de forma secuencial.

La programación orientada a objetos, se basa precisamente en esto, en crear objetos que se tratan como un todo e interactuar entre ellos.

1.4.1 Variables

Una variable es un espacio que se guarda en memoria y va cambiando de contenido según las necesidades del programa. En el caso de Javascript no es obligatorio declararlas, aunque si es aconsejable.

Además, JavaScript es un ***lenguaje no tipado***, es decir, una vez declarada la variable se puede almacenar en ella cualquier tipo de datos, un número, una letra, un string..es decir asigna el tipo por inferencia o deducción.

Una variable se declara mediante la instrucción **var**

```
var numero;
```

y también se puede iniciar o inicializar la variable con un determinado valor

```
var numero=1;
```

si no se asigna valor a la variable, esta tendrá como contenido **undefined.**

Se pueden declarar múltiples variables en una solo línea escribiendo var y los nombres separados por coma.

```
var numero1, numero2
```

Aunque se le puede dar el nombre que queramos, con unas determinadas especificaciones:

- Sólo puede contener caracteres alfanuméricos (a-z, A-Z, 0-9), y los símbolos: _, $.
- No puede comenzar por número.
- No puede ser una de las palabras reservadas utilizadas por el lenguaje (var, if, for, etc.).

Es aconsejable que se nombre dependiendo del uso que le vayamos a dar, para luego tener una referencia a la hora de utilizarla, por ejemplo:

```
var nombreCliente = "Juan Pérez";
```

1.4.1.1 AMBITO DE LAS VARIABLES

El ámbito de las variables es el lugar donde estarán disponibles. Por lo general, está disponible en el lugar donde se ha declarado, por ejemplo, las variables declaradas dentro de una página web estarán accesibles dentro de ella.

- **Variables globales:** las variables globales son las que están declaradas en el ámbito más amplio, es decir las declararemos en la parte superior del script. Son accesibles desde cualquier lugar, incluidas funciones, manejadores de eventos, etc.
- **Variables locales:** las variables locales son aquellas que se declaran en lugares más acotados como una función y sólo se tendrán acceso a ellas dentro del lugar donde se han declarado.

Aunque es posible, no es aconsejable declarar variables locales y globales con el mismo nombre, ya que puede crear confusión sobre qué variable se está utilizando en un momento determinado.

1.4.1.2 TIPOS DE DATOS

JavaScript interpretará que tipo de dato se está utilizando en el momento en que se le asigna el valor y realizará la conversión automática cuando cambiemos de tipo de dato. Los tipos básicos de datos son:

- **Entero/decimal:** permite cualquier número que sea necesario. Para indicar que es negativo basta con poner el signo menos adelante y el . se utiliza como separador decimal, pudiéndose omitir el 0 si el número no tiene parte entera, es decir, se puede utilizar .34 como número válido. También admite los número en notación científica, con E o e de manera indistinta.

- **Carácter/cadena de caracteres o strings:** a diferencia de otros lenguajes javascript utiliza el valor string tanto para un carácter como para una cadena de texto. Para definirlo se pueden utilizar tanto comillas simples como comillas dobles, aunque es preferible utilizar las comillas dobles para HTML y dobles para Javascript para que el navegador determine cuales son las de apertura y cierre.

- **Booleanos**: puede contener valores verdadero o falso únicamente y se suele utilizar para determinar si se cumple o no una condición.

Ejemplo:

```
var numero1=1;
//declaramos un número entero llamado numero1
var numero2=2.5;
//declaramos numero2 de tipo decimal o real
var numero3=numero1+numero2;
//numero3 valdrá número1+numero2, es decir 3.5
var texto = "Hola";
```

```
//declaramos un string (cadena) de valor Hola
var mezcla = texto + numero1; //mezcla dará por resultado Hola1
var cobrado=verdadero;
//cobrado es de tipo booleano (verdadero o falso)
```

También hay que tener en cuenta que si se quiere añadir caracteres especiales a un texto se debe agregar el carácter \ dentro de los apostrofes del string:

Secuencia	Resultado	Secuencia	Resultado	Secuencia	Resultado
\\	\	\"	"	\"	"
\n	Salto de línea	\t	tabulador	\b	Retorno de carro

Ejemplo:

```
var cadena="Esta cadena lleva el carácter \\ seguido de tres
saltos de línea \n \n\n y prosigue comilla simple \" y doble
\"\n"
```

Javascript también permite utilizar variables sin que se hayan declarado previamente. Cuando encuentre una variable no declarada, la crea y permite su utilización, lo que facilita la programación.

1.4.1.3 ARRAYS

Javascript como cualquier lenguaje de programación, también trabaja con arrays, que son variables que, mediante un índice, permiten organizar datos bajo un mismo nombre. Pero a diferencia de otros lenguajes, en Javascript cada uno de los valores del array puede ser de distinto tipo.

Como cualquier variable, el array se declara mediante la opción var, aunque para que javascript reconozca que efectivamente es un array hay que indicárselo:

```
var array = [];
```

y a continuación añadir los valores:

```
array[0]=valor0;
```

o bien agregando los valores a la hora de la declaración:

```
var array = [valor0, valor1, valor2, … ]
```

Así mismo, para acceder a cada elemento del array se utiliza el índice de la posición, es decir, el valor de array[0] será valor0. Si se intenta acceder a un índice de un array inexistente, el resultado será **undefined**. En Javascript el número de elementos del array se ajusta dinámicamente.

1.4.2 Operadores

Dependiendo del tipo de datos que estemos utilizando, se necesitan diferentes operadores:

- ***Matemáticos*** (+, - , *, / , %) realizan las operaciones matemáticas entre números, % devuelve el resto de la división, por lo que es un número entero.

```
var valor1 = 1;
var valor2 = 2;
var suma=valor1 + valor2;                //suma valdrá 3
var resta = valor2-valor1;               //resta valdrá 1
var multiplicacion =  valor1 * valor2;
//multiplicacion valdrá 2
var division = valor2/valor1;
//división valdrá 2
var resto =valor2%valor1;                //resto será 0
```

- ***Asignación*** (=, +=, -=, *=, /=, %=) para darle valor a una variable. En el caso de utilizarlo junto a un operador matemático, primero hará la operación y a continuación le asignará el valor a la variable.

```
var valor1 = 1;
valor1 += 2;            //valor1 pasará a valer 3
```

- ***Concatenación*** (+) se utiliza en cadenas de caracteres para unir textos.

```
var valor1 = "Nombre";
var valor2 = "Apellidos";
Nombre = valor1 + " " + valor2;
//Nombre será "Nombre Apellidos"
```

- ***Comparación*** (== , !=, <, >, <=; >=) se utilizan para comparar dos variables. Hay que tener cuidado en confundir el operador de asignación (=) con el igual (==). Devuelven un valor booleano, siendo true si la condición se cumple y false en caso contrario, siendo muy útil para opciones condicionales

```
var valor1 = 1;
var valor2 = 2;
if (valor1 >= valor2)   //devuelve false ya que 1<2
if (valor1 == valor2)   //devuelve false
if (valor1=valor2)
//devuelve true ya que el valor1 pasará a valer 2
```

- ***Lógicos*** (&& (y), || (o), ! (no)) se basan en las tablas de verdad, y es verdadero cuando se cumplen todas las condiciones, o es verdadero cuando se cumple una de las condiciones enumeradas, y no devuelve el valor contrario, es decir, verdadero si la condición es falsa.

```
var valor1 = true;
var valor2 = false;
if (valor1 && valor2)
//devuelve false ya que no cumple las dos
```

```
if (valor1 || valor2)
//devuelve true ya que una de las dos es verdadera
if (!valor1)
//devuelve false por ser el valor contrario
```

- ***Incremento/Decremento:*** el incremento/decremento permite agregar/restar una unidad a una determinada variable. Dependiendo de la posición en donde se coloque el operador, se incrementará/decrementará la variable antes o después de realizar otra operación (n++, --n)

```
var valor1 = 3;
valor1++;                    //valor1 pasa a valer 4
```

Si intentamos realizar una operación matemática con variables no numéricas, el resultado será **NaN** (Not a Number), lo que evitará que el programa deje de ejecutarse.

```
var valor1, valor2;
valor1 = 3;
resultado = valor1+valor2;
//resultado será NaN ya que valor2 no tiene valor
```

1.4.3 Interacción con el usuario

Existen tres formas básicas de interactuar con el usuario:

- ***alert***: envía un mensaje de información. Incluye un botón Aceptar.

```
alert("Mensaje informativo");
```

- ***confirm***: se utiliza para confirmar una tarea. Incluye dos botones: Aceptar y Cancelar, por lo que se obtiene un valor booleano (verdadero o falso) para realizar o no la acción determinada.

```
valor = confirm("Mensaje con acción a confirmar");
```

- ***prompt:*** solicita información al usuario.

```
valor = prompt("Mensaje con la información a solicitar",
[valor por defecto](opcional));
```

1.4.4 Conversión de tipos

Si queremos saber qué tipo de datos es una variable, utilizaremos la opción typeof, por ejemplo

```
var numero = 1;
//declara una variable número y le asigna el valor 1
alert(typeof numero);
//muestra un mensaje con la opción "number"
```

Hemos visto que podemos concatenar un string con un número y JavaScript automáticamente lo convierte a string.

Ejemplo:

```
var mezcla = "Hola " + 21;  //mezcla tiene el valor "Hola 21"
```

El problema surge cuando solicitamos un valor, y a continuación queremos realizar una suma de valores, al tomarlo como cadena de caracteres lo que hará es una concatenación en lugar de una suma.

Ejemplo:

```
var numero1 = prompt("Introduzca el primero número", 0);
//pide el primer número
var numero2 = prompt("Introduzca el segundo número", 0);
//pide el segundo número
var numero3 = numero1 + numero2;
//En lugar de sumar, concatena los valores
alert("El resultado es " + numero3);
//Si numero1=0 y numero2=0, numero3 = 00
```

Este error sólo ocurre cuando utilizamos el operador +, ya que se puede utilizar tanto para sumar número como para concatenar cadenas. En este caso, debemos convertir los valores de cadenas a número, para ello utilizamos dos instrucciones:

- ***parseInt***: convierte una cadena en número entero.
- ***parseFloat***: convierte una cadena en número decimal.

Entonces, siguiendo con el ejemplo anterior:

```
var numero1 = prompt("Introduzca el primero número", 0);
//pide el primer número
var numero2 = prompt("Introduzca el segundo número", 0);
//pide el segundo número
var numero3 = parseInt(numero1) + parseInt(numero2);
//Con conversión de valores
alert("El resultado es " + numero3);
//Si numero1=0 y numero2=0, numero3 = 0
```

1.4.5 Sentencia condicional

1.4.5.1 SENTENCIA CONDICIONAL IF-ELSE

Una sentencia condicional permite elegir realizar una acción dependiendo de las circunstancias que ocurran. Se suelen utilizar en la vida cotidiana, como por ejemplo, si llueve => llevar paraguas.

En JavaScript, las sentencias condicionales se utilizan mediante la sentencia if

```
if (condición) acción;          //if (llueve) llevar paraguas
```

También podemos considerar realizar una acción diferente si la condición no se cumple, es decir si llueve => llevar paraguas sino => llevar gafas de sol. En este caso, la opción sino se traduce en JavaScript como else

```
if (condición) acción;          //if (llueve) llevar paraguas
else  acción;                   //else llevar gafas de sol
```

La cláusula else es no obligatoria e incluso puede dejarse vacía.

Cuando se realizaran varias acciones dentro del if, deberemos encuadrarlas dentro de {} para que se consideren un conjunto, por tanto se recomienda siempre utilizar las llaves, aunque sea una única acción, para evitar errores y tener un código más claro. Volviendo al ejemplo:

```
if (condición) {
   Accion1;
   Acción2;
}
else {
   Acción3;
}
```

Además, podemos querer combinar más de una condición, con lo cual podemos anidar los if. En nuestro ejemplo: Si llueve => llevar paraguas, sino => si hace calor => ir a la playa => ir a esquiar

```
if (condición1) acción;          //if (llueve) llevar paraguas
else                             //sino
   if(condición2) acción;        //si (hace calor ) ir a la playa;
   else acción;                  //sino ir a esquiar
```

La condición es una expresión booleana, y se pueden utilizar los operadores de comparación y lógicos vistos anteriormente.

Ejemplo:

Consideremos la evaluación de unos alumnos, si su nota es inferior a 5, será "Suspenso", si es inferior a 7 tendrá "bien", si es inferior a 9 tendrá "Notable", sino será "Sobresaliente". También tenemos que tener en cuenta que la nota no puede ser menor que 0 ni mayor de 10. Traduciéndola a código:

```
var nota = prompt("Escribe la nota del alumno", 0);
var calif;
if (nota<0 || nota>10)
    alert("La nota debe estar entre 0 y 10");
else
    if (nota<5)
        calif="suspenso";
    else
        if (nota<7)
            calif="bien";
        else
            if (nota<9)
                calif="notable";
            else
                calif="sobresaliente";
alert("La calificación del alumno es " + calif);
```

1.4.5.2 SENTENCIA SWITCH

Cuando tenemos varias condiciones podemos utilizar la sentencia switch en lugar de utilizar ifs encadenados, con la única diferencia que solo permite evaluar valores concretos, no intervalos. Esta instrucción realiza diferentes acciones dependiendo de la variable o expresión. La forma de utilización es:

```
Switch(variable o expresión){
case valor1: acciones; break;
case valor2: acciones; break;
…
default: acciones; break
}
```

La instrucción **break** se utiliza para salir/saltar la opción switch, terminando la evaluación y continuando con el resto del código. Esta instrucción, aunque opcional es recomendable.

La instrucción **default** representa las acciones que se ejecutaran sino se han cumplido ninguna de las opciones anteriores. También es opcional.

Ejemplo:

```
switch(new Date().getDay()){
//Date().getDay() devuelve el dia de la semana en número
case 0:
        dia="Domingo";
        break;
case 1:
        dia="Lunes";
        break;
case 2:
        dia="Martes";
        break;
case 3:
```

```
        dia="Miércoles";
        break;
case 4:
        dia="Jueves";
        break;
case 5:
        dia="Viernes";
        break;
case 6:
        dia="Sábado";
        break;
}
```

1.4.6 Bucles: while / do while / for

Un bucle es una estructura repetitiva que realiza la iteración mientras o hasta que cumple una condición o simplemente se repiten un número determinado de veces. Los bucles se pueden anidar dentro de otro al igual que las sentencias condicionales.

En JavaScript tenemos tres tipos de instrucciones repetitivas:

1.4.6.1 WHILE

While significa mientras, por tanto, el bucle se repetirá mientras se cumpla la condición.

```
while (condición){
instrucciones;
}
```

Condición es una expresión que da un resultado true o false, que hará que se ejecuten o no las instrucciones.

Ejemplo:

Como ejemplo sencillo, le preguntaremos al usuario hasta que número quiere que se imprima e iremos mostrando los números

```
var NumeroFinal= prompt("Escribe desde 0 hasta el número .. ",
0);                                 //pregunta el número
var i=0;
//define una variable a 0
while (i<Numero){
//mientras i sea menor que el número dado
        document.write(i++ + "<br>");
//escribe i y un salto de línea e incrementa i
}
```

1.4.6.2 DO .. WHILE

El bucle do .. while, es similar al while, con la diferencia que en este caso primero se hacen las expresiones y a continuación se evalúa la condición, así que las acciones como mínimo se realizaran una vez.

```
do{
instrucciones;
} while (condición)
```

Ejemplo:

Para realizar el mismo ejemplo anterior, haremos:

```
var NumeroFinal= prompt("Escribe desde 0 hasta el número .. ",
0);                                 //pregunta el número
var i=0;                            //define una variable a 0
do{
      document.write(i + "<br>");
```

```
//escribe i y un salto de línea
while (++i<Numero)
//incrementa i y compara con el número, y realiza nuevamente el
bucle mientras menor que el número dado
}
```

1.4.6.3 FOR

El bucle FOR es uno de los más utilizados, y realiza las expresiones un número determinado de veces.

```
for (expresión, condición, operación){
Instrucciones
}
```

Ejemplo:

Utilizando el mismo ejemplo anterior

```
var NumeroFinal= prompt("Escribe desde 0 hasta el número .. ",
0);                                  //pregunta el número
for (var i=0; i<Numero, i++){
//mientras i inicializado a cero sea menor que el número dado,
realiza y luego suma 1 a i
       document.write(i + "<br>");
//escribe i y un salto de línea
}
```

En JavaScript las sentencias pueden llevar una etiqueta que los identifique para poder hacer referencia a ellas en otro momento.

```
bucle: for (var i=0; i<Numero, i++){
//mientras i inicializado a cero sea menor que el número dado,
```

```
realiza y luego suma 1 a i
        document.write(i + "<br>");
//escribe i y un salto de línea
}
```

1.4.7 Funciones

Cuando una serie de instrucciones se repiten en distintos programas o en diversas partes de un programa, se crea una función a la que luego se llama cada vez que sea necesario. También se utilizan para resolver problemas complejos a través de trozos de código pequeños que hacen más sencilla su comprensión.

Existen dos tipos de funciones, aquellas que devuelven algún valor, y aquellas que no devuelven nada, sino que solo realizan acciones. Para devolver el resultado se utiliza la sentencia **return** seguida de la variable que se quiera devolver. Una vez que se ejecuta esta sentencia se vuelve al programa principal, por lo que si hubiera alguna instrucción posterior, no se ejecutaría. Hay que tenerlo en cuenta, y colocar esta sentencia como la última de la función.

La función puede contener parámetros o argumentos que son cualquier tipo de variable que sea necesaria para la realización de las acciones correspondientes. El parámetro que recibe la función no tiene un tipo de dato determinado, sino que lo interpreta JavaScript. Si tiene varios argumentos, se deben escribir en el orden adecuado de cómo se han declarado.

Además, también permite abstraer los problemas para que se puedan utilizar simplemente nombrándolas sin conocer el contenido de la función en sí, sólo sé que al darle unos determinados parámetros se obtiene un resultado sin necesidad de saber cómo se llega a él. Se crean librerías de funciones y los programadores pueden utilizarlas como si de cajas negras se trataran.

El formato de las funciones es:

```
function NombreFunción (arg1, arg2,..){
}
```

Una función debe tener un nombre descriptivo de cuál es su cometido, que será único y claro.

Como ejemplo la funcion suma, una función para sumar dos números. Dentro de ella tendremos dos maneras de realizarla, la primera de ellas no devuelve ningún valor, sino que muestra en la web el resultado de la suma y en el segundo ejemplo el resultado es devuelto a la función o programa principal mediante la palabra reservada **return**.

```
function sumar(x,y){
    var suma=x+y;
    document.write("La suma es"  + suma);  //escribe el resultado
}

function sumar(x,y){
    var suma=x+y;
    return suma;                           //devuelve el resultado
}
```

Cuando necesitemos utilizar la función suma, escribiremos

```
suma(3,4);  //En el primer caso, para que imprima el resultado

document.write("La suma es" + suma(3,4));
//se llama a la función en el momento de escribir el resultado
```

PROYECTO

Crear un archivo script.js

Crear un script en JavaScript que pregunte el nombre del usuario. Si no escribe el nombre dará un mensaje de error. Si no, escribirá Bienvenido y nombre de la persona en la parte superior derecha de la página.

1.5 OBJETOS EN JAVASCRIPT

La Programación Orientada a Objetos se basa en tratar los componentes de la programación como entes de la vida real, caracterizándose por la encapsulación y la abstracción para tratarlo como un todo. Un objeto es una entidad en sí, que tiene sus propiedades y sus métodos. Trasladándolo a la vida real , un objeto sería una pelota que tiene unas propiedades como color, material, etc. y una serie de métodos que se pueden realizar con ella como tirarla, botarla, patearla, etc. pero podemos nombrar la pelota como objeto sin necesidad de conocer sus propiedades o métodos.

Javascript, no es exactamente un lenguaje de programación orientado a objetos (POO), sino un lenguaje de comandos basado en Objetos, ya que puede crearlos y desarrollarlos para hacer uso de ellos. Entonces, los objetos son entidades compuestas por dos elementos:

- **Atributos**: propiedades que contienen datos
- **Métodos**: para desarrollar operaciones con los atributos, es decir, funciones que determinan la conducta de ese objeto.

El objeto se define como cualquier variable con la palabra reservada var, por ejemplo:

```
var pelota = new Object();  //crea un objeto de nombre pelota
```

1.5.1 Atributos

Los atributos o propiedades de un objeto son las características propias de ese objeto y se accede a ella mediante la notación de puntos.

```
Objeto.atributo
```

Volviendo al ejemplo de la pelota, hemos dicho que un atributo puede ser el color o el material. Para definirlo pondremos

```
var pelota = new Object();  //crea un objeto de nombre pelota
pelota.color = "Rojo";
pelota.material  = "plástico";
```

1.5.2 Métodos

Los métodos son funciones que se realizan con los atributos de ese objeto, es decir es una capacidad o un comportamiento de ese objeto. En nuestro ejemplo, podría ser tirarla. Al igual que las propiedades, a los métodos podemos acceder mediante la notación del punto.

```
Objeto.Metodo(parámetros)
```

En el ejemplo

```
pelota.tirar(distancia)
```

1.5.3 Manejadores de eventos

Los eventos son las acciones que los usuarios realizan, como, por ejemplo, hacer un clic de ratón o pulsar una tecla y es el uso más habitual que tiene Javascript.

Un manejador de evento, es una función que se realiza cuando se dispara ese evento, bien por el usuario o bien por el sistema. JavaScript da respuesta a cada uno de los eventos disponibles cuando se produzcan.

Aunque se pueden incluir los manejadores de eventos directamente en HTML, no es conveniente mezclar, sino escribirlo en el código JavaScript. Los principales eventos los podemos distinguir como:

- **Evento generales**
  - **Onchange**: al cambiar el contenido
  - **Onerror**: al dar error
  - **onMove**: al mover
  - **onResize**: al cambiar el tamaño
- **Eventos de enfoque**
  - **Onblur:** cuando se pierde el foco de un elemento de un formulario.
  - **Onfocus**: al recibir el foco
- **Eventos de carga**
  - **Onabort:** al detener la carga
  - **Onload**: al cargar una página
  - **OnUnload**: al abandonar o salir de una página
- **Eventos de teclado**
  - **onKeydown:** al pulsar una tecla
  - **onKeypress:** al mantener pulsada una tecla
  - **onKeyup:** al soltar una tecla que estaba pulsada
- **Eventos de formulario**
  - **OnReset:** al iniciar un formulario
  - **OnSubmit**: al enviar un formulario
- **Eventos de ratón**
  - **Onclick**: al hacer clic con el botón izquierdo del ratón sobre el elemento
  - **onDblClick**: al hacer doble clic con el botón izquierdo del ratón
  - **onDragDrop**: al arrastrar y soltar
  - **onDragstart**: al arrastrar con el ratón

- **onMousedown:** al pulsar un botón del ratón
- **onMousemove:** al mover el puntero del ratón
- **onMouseout**: cuando el ratón sale de la zona del objeto
- **OnMouseover**: al pasar el ratón por encima del objeto
- **onMouseup:** al dejar de pulsar un botón del ratón
- **Onselect**: al seleccionar texto o un elemento
- **onSelectstart**: al comenzar una selección con el ratón

Ejemplo:

En el siguiente ejemplo, cuando hagamos clic sobre el botón Púlsame, se escribirá Hola Mundo en el p con identificador demo de la página

```
<button onclick="funcion()">Pulsame</button>
//HTML coloca un botón en la pagina
<p id="demo"></p>
//Espacio HTML donde se escribirá Hola Mundo cuando se pulse el
botón
<script>
function funcion() {
  document.getElementById("demo").innerHTML = "Hola Mundo";
}
</script>
```

Otra forma de incluir un manejador de evento es utilizar el método AddEventListener sobre el objeto, para agregar un detector de eventos que se activa cuando un usuario haga clic en un botón. Este método requiere tres parámetros:

```
addEventListener (evento, función, capture tipo booleano)
```

Ejemplo:

El mismo ejemplo anterior pero con addEventListener

```
<button id="myBtn">Pulsame</button>
<p id="demo"></p>
<script>
  document.getElementById("myBtn").addEventListener("click",
function(){
    document.getElementById("demo").innerHTML = "Hola Mundo";
  });
</script>
```

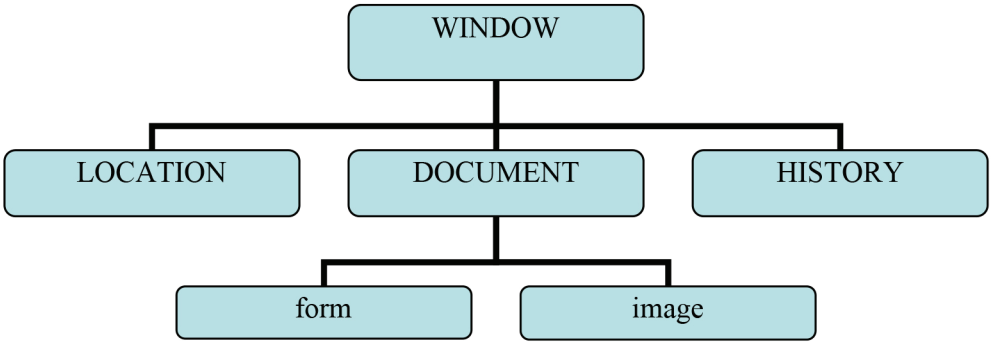
1.5.4 Modelo de objetos del documento (DOM)

Dentro de los tipos de objetos, podemos hacer una selección entre:

- Tipos definidos por el programador
- Tipos **predefinidos** de Javascript: son objetos ya creados y disponibles para que el programador pueda trabajar con ellos.
- **Modelo de objetos del documento**: aunque el DOM no es exclusivo de Javascript, es un estándar creado por W3C, está íntimamente relacionado con este lenguaje, ya que lo utiliza continuamente para acceder y modificar las páginas web dinámicamente, por lo que necesita identificar con precisión cada elemento. Según este modelo, el navegador representa en memoria los elementos de una página web cuyo elemento superior es document.

1.5.4.1 OBJETO WINDOW

Este modelo tiene una jerarquía de objetos, cuyo principal es WINDOW, que define la ventana principal sobre la que se está trabajando.



Por defecto, toda página web contiene los objetos window, document, location, history, de los que hablaremos detalladamente a continuación. Salvo estos objetos, la mayoría de objetos del navegador representan los elementos presentes en los documentos HTML, teniendo su correspondencia con las etiquetas.

Tal y como hemos comentado anteriormente, al no ser exactamente un lenguaje POO, sino basados en objetos, no se utiliza el concepto de herencia, los objetos situados en un nivel inferior son propiedades de los de nivel superior.

Métodos del Objeto window

Es decir, por cada etiqueta body se genera un objeto window que tendrá sus métodos. Por ejemplo, para abrir una nueva ventana, utilizamos el método open:

```
window.open(URL,nombre, propiedades)
```

Este utiliza dos parámetros obligatorios y uno opcional, el primero URL indica la dirección de la página web que se cargará en esa nueva ventana, nombre identifica la nueva ventana para utilizarlo en HTML. Las propiedades deben aparecer entre comas para indicar si se colocan las barras de herramienta, de dirección, etc.

Ejemplo:

```
window.open(http://www.google.com,"","width=550,height=249,menub
ar=no")
window.alert("Bienvenido al sitio web")
```

Close es otro método de window que no necesita ningún parámetro.

```
window.close()
```

Otra propiedad de la ventana del navegador es la barra de estado y tiene de nombre **status**, por tanto si queremos colocar un mensaje en la barra de estado:

```
window.status="Esta es la barra de estado"
```

Cuando se tiene un único objeto window, no es necesario indicarlo expresamente. Es decir, cualquier método que se llame a sin ninguna referencia, realmente estamos usando la referencia window. Esto lo utilizamos ya cuando vimos las interacciones con el usuario ya que los métodos alert, confirm y prompt en definitiva, son métodos del objeto window.

Un aspecto importante es que cualquier variable no declarada, pasa a ser propiedad del objeto window, con independencia de dónde se utilice esa variable.

```
function () {
   texto = "Variable accesible"
//variable no declarada, se le asigna un valor
};
alert (texto);                  //Muestra: "Variable accesible"
```

Es conveniente no utilizar esta característica de JavaScript puesto que generará una gran confusión de código.

1.6 LOS OBJETOS LOCATION E HISTORY

El objeto **location** representa la URL completa, pudiendo cambiarla o extraer las distintas partes de la misma.

Este objeto tiene ocho propiedades

- **href**: proporciona la URL completa de la página web cargada en el navegador.
- **hash**: representa un marcador, que aparece detrás del símbolo #
- **host**: representa el servidor al que se accede incluido el puerto utilizado
- **hostname**: representa el nombre del servidor
- **pathname**: es la estructura donde está ubicado.
- **port**: Puerto que utiliza el equipo que mantiene la página
- **protocol**: protocol que se está utilizando, http/https usualmente, aunque también puede otro como ftp.
- **search**: para determinar el criterio de búsqueda

Ejemplo:

En el ejemplo obtendremos el nombre del dominio del servidor, por ejemplo *www.google.es* si nos encontramos en esa página

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Page hostname is: "
+ window.location.hostname;
</script>
```

El objeto **history** permite acceder al historial del navegador. Con este objeto nos podemos mover a través de la lista, aunque no se puede acceder a las direcciones por motivos de seguridad. La única propiedad que tiene este objeto es **length** que lo que devuelve es el número de lugares visitados durante la sesión.

Además, asigna un índice a los lugares visitados, utilizando métodos para navegar a través del historial:

- **Back**: va hacia atrás, a la página visitada anteriormente
- **Forward**: vuelve hacia adelante en el historial, sólo si se ha ido para atrás previamente.
- **Go (número/string)** para ir a una URL determinada.

Ejemplo:

Tenemos en la página un botón de nombre Ir atrás que, cuando se pulse, irá a la página anteriormente visitada

```
<html>
   <head>
      <script>
         function atras() {
              window.history.back()
//vuelve para atrás en la historia
         }
      </script>
   </head>
   <body>
      <input type="button" value="Ir atrás" onclick="atras()">
   </body>
</html>
```

1.7 EL OBJETO DOCUMENT

El objeto document es el objeto básico del lado del cliente. Proporciona acceso a la página web, proporcionando propiedades y métodos para acceder a los objetos que componen esa ventana del navegador.

Su gran importancia es en la gran cantidad de referencias a las que hace como objetos image, form, etc. y no solo al cuerpo de la página, sino también a los de la cabecera. Como ejemplo, si queremos encontrar determinados elementos utilizaremos el punto para acceder a ellos:

- **Document.body:** devuelve el elemento body de document
- **Document.title:** devuelve el elemento title (título de la página) del document
- **Document.images:** es un array donde cada índice representa una imagen de la página
- **Document.bgcolor:** da o define el color de fondo, donde color puede ser una palabra que define el color en inglés ("red", "green") o el valor hexadecimal de RGB (#FFFFFF blanco, color por defecto)

También, utiliza métodos de los cuales los más normales son:

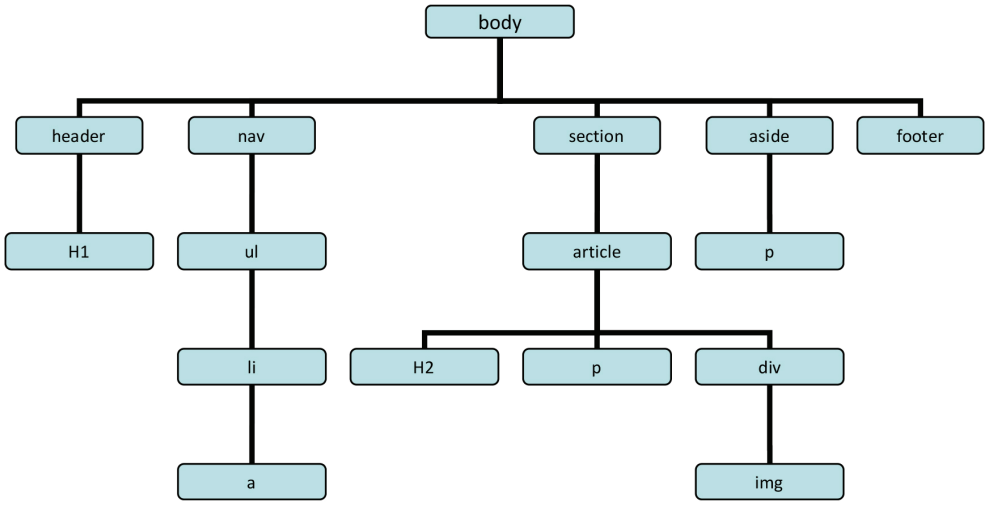
- **Open():** abrir el documento
- **Close():** cerrar el documento
- **Write/writeln(expresión,..):** escribe las expresiones HTML.

El objeto document estará compuesto por un conjunto de objetos, por lo que necesitaremos una forma para acceder a un elemento particular. Los principales son:

- **Element**: son los nodos definidos por las etiquetas HTML. Es decir, body, div o h1 serán elements.

- **Text**: el texto que está dentro de un nodo element se considera como un nodo hijo de tipo text.

Si nos remitimos al ejemplo de la página web, los elementos en forma de árbol del DOM quedaría de la siguiente manera:



Cuando recorremos un árbol, hablamos de padre cuando hablamos de un nodo superior, o hijo si hablamos de un nodo inferior. En nuestro ejemplo body tiene 5 hijos: header, nav, section, aside y footer, que son hermanos entre sí, además el padre de li es ul. El DOM proporciona más funcionalidades que permiten recorrer el árbol del documento y acceder a los nodos para crear efectos dinámicos:

- **parentNode**: proporciona el nodo padre
- **childNodes**: proporciona el conjunto de nodos hijos en un array
- **firstChild**: accede al primer nodo hijo
- **lastChild**: accede al último nodo hijo
- **previousSibling**: devuelve el nodo hermano previo
- **nextSibling**: devuelve el nodo hermano siguiente

Ejemplo:

Si tenemos el siguiente código HTML

```
<ul>
   <li id="miLI">Coffee</li>
   <li>Tea</li>
</ul>
```

Si ponemos la siguiente sentencia dentro del script:

```
var x = document.getElementById("miLI").parentNode.nodeName;
```

x tendrá el valor ul, que es el padre del nodo li

1.7.1 Acceso a los elementos HTML

Para el acceso a los elementos a través del DOM, el objeto document utiliza tres métodos :

- **getElementById**(): proporciona acceso a un elemento conociendo su atributo id, teniendo en cuenta que no debe de existir más de un elemento con el mismo id en HTML. Es el modo más frecuente de acceder a un nodo concreto del DOM.
- **getElementsByTagName**(): recupera, en forma de array, todos los elementos que coinciden con la etiqueta que se utiliza como parámetro. Aparecen en el mismo orden que en el código de la página web.
- **getElementsByName**(): permite recuperar sólo los elementos que tienen un atributo name especificado como argumento. El orden también será el mismo que el orden del código de la página.

Ejemplo:

Este ejemplo lo habíamos visto anteriormente que lo que hace es acceder al elemento de id demo y varia su valor a Hola Mundo

```
<button onclick="funcion()">Pulsame</button>
//HTML coloca un botón en la pagina
<p id="demo"></p>
//Espacio HTML donde se escribirá Hola Mundo cuando se pulse el
botón
<script>
function funcion() {
  document.getElementById("demo").innerHTML = "Hola Mundo";
}
</script>
```

1.7.2 Cambio de los elementos

Mediante los siguientes métodos podremos cambiar los elementos HTML:

- **innerHTML**: permite cambiar el código HTML en sí

```
<button onclick="funcion()">Pulsame</button>
//HTML coloca un botón en la pagina
<p id="demo"></p>
//Espacio HTML donde se escribirá Hola Mundo cuando se
pulse el botón
<script>
function funcion() {
  document.getElementById("demo").innerHTML = "Hola
Mundo"; //cambia demo
}
</script>
```

- **setAttribute(atributo,valor):** cambia el valor del atributo del elemento HTML

  También se puede cambiar el atributo o el estilo de un elemento utilizando:

```
elemento.atributo = valor;
elemento.style.propiedad = estilo;
```

Añadir manejadores de eventos

También podemos añadir manejadores de eventos mediante

```
document.getElementById(id).evento = function()
```

1.8 EL OBJETO FORM

Este objeto se utiliza con los formularios. Los formularios son la forma más utilizada para recoger información en un sitio web y para ello utiliza una serie de elementos como puede ser text (cuadro de texto), textarea (área de texto), radio (botones de opción), etc.

Recordemos que los formularios se construyen entre etiquetas <form> </form> y dentro de ellas podemos colocar:

- **Botones:** lo más utilizado en los formulario son los botones, básicamente tres:
  - **Botón de envío:** crea un botón para enviar el formulario al servidor.

```
<input type="submit" value="Enviar">
```

- **Botón de borrado:** se utiliza para limpiar el formulario.

```
<input type="reset" value="Restablecer">
```

- **Botón**: para definir un botón general.

```
<input type="button" value="Pulsame">
```

- **Campo de entrada de datos:** son los elementos básicos que se utilizan para recoger la información y vienen definidos por la etiqueta **input**. La etiqueta label se usa para asociar el texto que aparecerá en el formulario antes del cuadro que dará la entrada del texto. Dentro de los tipos de valores de entrada, tenemos:

  - **Campo de texto:** será un campo de tipo texto

```
<label for="nombre"> Nombre: </label>
<input type="text" id="nombre" name="nombre">
```

  - **Contraseñas**: al poner como contraseña, al introducir el valor se muestran asteriscos

```
<label for="contraseña"> Contraseña: </label>
<input type="password" id="Contraseña" name="pwd">
```

  - **Checkbox**: permite tildar/no tildar para obtener el valor booleano (sí o no)

```
<input type="checkbox" name="nombre" value="valor por
defecto"  [checked="checked"]>
```

- **Radio**: permite elegir entre una serie de opciones, definidas por el grupo name.

```
<input type="radio" name="color" value="rojo">
<input type="radio" name="color" value="verde">
<input type="radio" name="color" value="amarillo">
```

- **Color**: para especificar un color

```
<label for="Color"> Elige el color: </label>
<input type="color" id="Color" name="Color">
```

- **Fecha**: especifica una fecha. Dependiendo del navegador, puede aparecer una interfaz para su selección.

```
<input type="date" id="comienzo" name="Fecha"
value="2020-05-12" min="2020-01-01" max="2020-12-31">
```

- **E-mail**: para campos que van a contener una dirección de correo. Dependiendo del navegador, puede ser validado automáticamente.

```
<label for="mail"> Correo electrónico: </label>
<input type="email" id="mail" name="Correo">
```

- **Fichero:** para adjuntar un archivo mediante el botón Examinar…

```
<label for="fichero"> Seleccionar archivo: </label>
<input type="file" id="fichero" name="Fichero">
```

- **Número**: define un campo para entrar un número

```
<label for="cantidad"> Cantidad (entre 1 y 5): </label>
<input type="number" id="cantidad" name="cantidad"
min="1" max="5">
```

- **Tel**: define un campo para entrar un número de teléfono

```
<label for="telefono"> Entre un número de teléfono: </
label>
<input type="tel" id="telefono" name="telefono"
pattern="[0-9]{2}-[0-9]{2}-[0-9]{3}" required>
```

- **Áreas de texto**: es para introducir gran cantidad de texto.

```
<textarea rows="numerofilas" cols="numerocolumnas"
name="nombre"> Texto que se colocará en el área de texto
</textarea>
```

- **Grupos de selección:** para elegir entre una serie de opciones mediante desplegable, utilizamos la etiqueta <select>

```
<select name="transporte">
   <option value="coche">Coche</option>
   <option value="avión">Avión</option>
   <option value="tren">Tren</option>
</select>
```

1.8.1 Validar la información de un formulario

Aunque HTML5 ya permite definir patrones de validación para los diferentes campos de formularios, aún podemos utilizar JavaScript para personalizar nuestras validaciones, siempre del lado del cliente, antes de mandarlo al servidor, que también puede constar de sus propias validaciones de seguridad. También se utilizará JavaScript cuando tenemos un navegador que no admite la validación HTML.

En JavaScript, el funcionamiento de la validación se basa en el comportamiento del evento **onsubmit**. Si este evento devuelve el valor falso, el

formulario no se envía. Por tanto, en cuanto se encuentra un elemento incorrecto dentro de la validación se cambia el valor de este evento.

Por ejemplo, si tenemos un campo de formulario vacío, podríamos crear la siguiente función:

```
function validar() {
   var x=document.forms["formulario"]["Campo"].value;
   if (x==""){
      alert("El campo no puede quedar vacío");
      return false;
   }
}
```

Y en el formulario del archivo HTML, colocamos el evento submit

```
<form name="Formulario" action="/action_page.php"
onsubmit="return validar()" method="post">
   Nombre: <input type="text" name="Campo">
   <input type="submit" value="Submit">
</form>
```

La mayoría de los navegadores admiten una API para la validación de restricciones, que son una serie de métodos y propiedades disponibles en el DOM. La API de validación dispone de dos métodos:

- **checkValidity()**: Si un elemento no es válido, devuelve false

```
<p> Entre un número del 100 al 300 </p>
<input id="num" type="number" min="100" max="300"
required>
<button onclick="funcion()">OK</button>
<p id="mensaje"> </p>
<script>
   function función(){
```

```
        var numero=Document.getElementById("num");
        if (!numero.checkValidity()) {
        document.getElementById("mensaje").innerHTML =
        numero.validationMessage;
      } else {
        document.getElementById("mensaje").innerHTML =
        "Número correcto";
        }
      }
  </script>
```

- **setCustomValidity(mensaje):** añade un mensaje de error personalizado que se muestra cuando el elemento no es validado.

```
  <form>
     <label for="mail"> Dirección de correo: <label>
     <input type="email" id="mail" name="mail">
     <input type="submit" value="Submit">
  </form>

  <script>
     var mail=Document.getElementById("mail");
     mail.addEventListener("input",function(event){
        if (mail.validity.typeMismatch){
           mail.setCustomValidity("El mail no es correcto");
        }
     })
  </script>
```

1.9 OTROS OBJETOS

Existen otros objetos predefinidos que tienen relación directa con el propio JavaScript, que representan funcionalidades ampliadas o incorporadas al lenguaje, algunos de los más utilizados.

1.9.1 Date

Permite manejar fechas y horas. Para crear una objeto con el dia y hora actuales pondríamos:

```
miFecha = new Date()
```

Algunos de los métodos son:

- **getDate()/setDate()**: devuelven o establece e día
- **getDay()/setDay():** devuelve/establece el día de la semana
- **getHours()/setHours():** devuelve/establece la hora
- **getMinutes()/setMinutes():** devuelve/establece los minutos
- **getMonth()/setMonth():** devuelve/establece el mes
- **getSeconds()/setSeconds():** devuelve/establece los segundos
- **getTime():** devuelve los milisegundos transcurridos entre el día 1/1/1970 y la fecha correspondiente.
- **setTime():** actualiza la fecha completa a partir de un número de milisegundos.
- **getYear()/setYear():** devuelve/establece el año

1.9.2 Math

Proporciona funciones matemáticas estándar. Las constantes se definen como propiedades mientras que las funciones se definen como métodos.

Algunas de las propiedades son:

- **E**: constante de Euler
- **LN2**: logaritmo neperiano de 2
- **LOG2N**: logaritmo en base 2 de N
- **LOG10N**: logaritmo decimal de N
- **SQRTN**: raíz cuadrada de N

Y como métodos:

- **abs(x):** valor absoluto de x
- **acos(x):** arcocoseno de x
- **asin(x):** arcoseno de x
- **atan(x):** arcotangente de x
- **cell():** entero igual o inmediatamente siguiente al valor dado.
- **cos(x):** coseno de x
- **exp():** resultado de elevar E por un número
- **floor(x):** mayor entero que es menor o igual que x
- **log():** logaritmo neperiano
- **max(x,y):** mayor valor entre x e y
- **min(x,y):** menor valor entre x e y
- **pow(x,y):** devuelve x elevado a y
- **random():** muestra un número aleatorio entre 0 y 1
- **round(x):** redondea al entero más próximo
- **sin():** seno del número
- **sqrt(x):** raíz cuadrada de x
- **tan():** tangente del número

1.9.3 String

Para trabajar con cadenas de caracteres. Como propiedades tiene:

- **length**: longitud en caracteres de la cadena.

Y como métodos:

- **charAt(indice):** devuelve el carácter que hay en la posición indicada como índice.
- **fromCharCode():** crea una cadena a partir de una secuencia de caracteres separados por comas.
- **indexOf(x, c):** muestra la posición de la primera vez que aparece el carácter x a partir de la posición c que se desea empiece la búsqueda
- **lastIndexOf(x, c)**: igual que indexOf con la diferencia que comienza desde el final de la cadena en lugar del principio.
- **replace(subs, nuevostr)**: sirve para reemplazar porciones de texto
- **Split(separador):** separa una cadena en subcadenas según el separador
- **Substring(inicio, fin):** muestra la subcadena que empieza con el carácter inicio y termina en el carácter fin.
- **toLowerCase():** convierte todos los caracteres de la cadena a minúsculas
- **toString():** devuelve un objeto en forma de cadena
- **toUpperCase():** convierte todos los caracteres de la cadena a mayúsculas
- **valueOf():** muestra el valor de un objeto en forma de cadena

PROYECTO

- Crear una página web adicional a nuestra página en la que crearemos un formulario de registro de usuario.
- Validar las entradas de ese formulario para que los datos que se solicitan sean correctos.
- Cambiar el archivo anterior para que el nombre de la persona lo obtenga a través del formulario.