Apuntes de Javascript

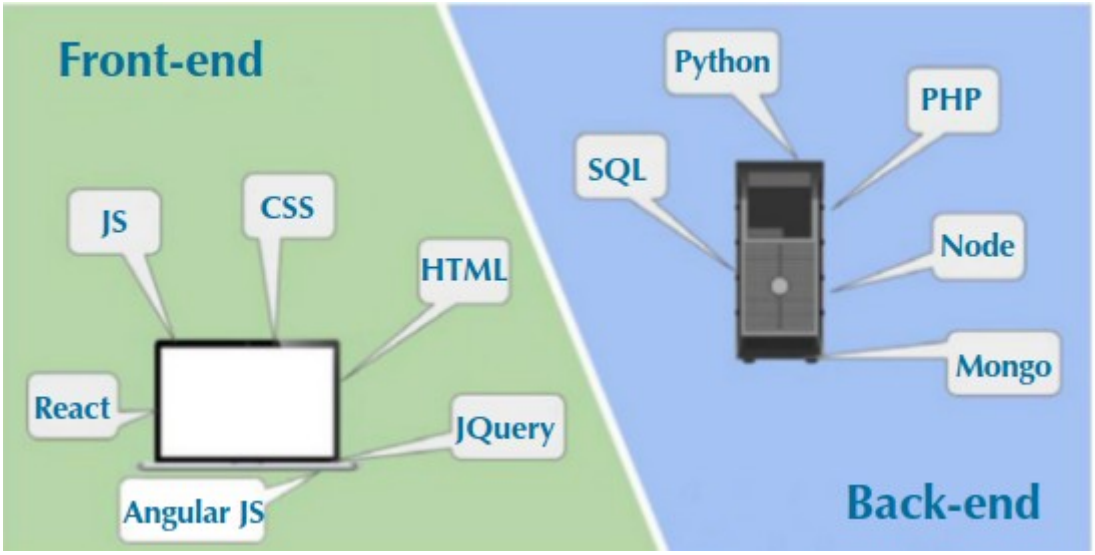
Sumario

Objetivos

- Conocer la sintaxis del lenguaje Javascript.
- Conocer cómo trabaja Javascript.
- Conocer para qué se usa Javascript.

Introducción

Javascript es un lenguaje relacionado con la programación web y se usa principalmente para programar del lado del cliente, es decir, del lado del navegador. Se podrán hacer muchas cosas con Javascript que mejorarán el comportamiento de un sitio web, como cambiar el contenido de los elementos HTML, valor de atributos, estilos, etc. es decir, JavaScript agrega dinamismo a las páginas web.



Estos y otros aspectos del lenguaje serán tratados en este documento desde un enfoque lo más práctico posible.

¿Dónde introducir el código Javascript?

- Podremos introducir código Javascript en diferentes lugares:

  - En el propio archivo HTML, entre las etiquetas `<script>` y `</script>` ya sea en la sección `<head>` o `<body>`.
  - En un fichero externo con extensión .js que habrá que enlazar desde el archivo HTML mediante `<script src="...">`.

```
<!DOCTYPE html>
<html>
<head>
    <script>
         function cambiarTexto() {
            document.getElementById("prueba").innerHTML = "Otro texto";
        }
    </script>
</head>
<body>
    <p id="prueba">Texto original</p>
    <button type="button" onclick="cambiarTexto()">Cambia el texto</button>
</body>
</html>
```

- Insertar en la parte final del `<body>` acelera la carga de la página:

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

</head>
<body>
    <p id="prueba">Texto original</p>
    <button type="button" onclick="cambiarTexto()">Cambia el texto</button>

    <script>
         function cambiarTexto() {
            document.getElementById("prueba").innerHTML = "Otro texto";
        }
    </script>
</body>
</html>
```

- Fichero externo .js:

```
// Fichero cambiarTexto.js
```

```
function cambiarTexto() {
      document.getElementById("prueba").innerHTML = "Otro texto";
}
```

```
<!DOCTYPE html>
<html>
<head>
    <!-- Tambien puede ir en el body -->
    <script src="cambiarTexto.js" ></script>
</head>
<body>
    <p id="prueba">Texto original</p>
    <button type="button" onclick="cambiarTexto()">Cambia el texto</button>
</body>
</html>
```

- Entre otras cosas, también podemos cambiar los estilos dinámicamente:

```
<html>
    <head>
        <script>
            function cambiaTamano(){
                document.getElementById("p1").style.fontSize = "10px";
            }
        </script>
    </head>
    <body>
        <p id="p1" style="font-size: 220px" onclick="cambiaTamano()">Texto</p>
    </body>
</html>
```

- Ejemplo de cambio de un atributo:

```
<!DOCTYPE html>
<html>
      <head>
            <script>
            function cambiaImagen(){
                  var imagen = document.getElementById("myImg");

                  if (imagen.src.match("green")) {
                        imagen.src =
"http://myfpschool.com/wp-content/uploads/2016/06/myblack.jpeg";
                  } else {
                        imagen.src =
"http://myfpschool.com/wp-content/uploads/2016/06/mygreen.jpeg";
                  }
            }
            </script>
      </head>
      <body>
            <img id="myImg" onclick="cambiaImagen()"
```

```
            src="http://myfpschool.com/wp-content/uploads/2016/06/myblack.jpeg"
            width="100" height="180">
    </body>
</html>
```

Salida de datos

- Javascript dispone de varias formas para realizar la salida de datos:

    - *innerHTML*: permite el acceso al contenido de los elementos HTML.
    - *document.write()*: permite generar código HTML directamente.
    - *window.alert()*: crea una ventana de alerta.
    - *console.log()*: para debbuging.
    - *window.print()*: abre la ventana de impresión.

```
<!DOCTYPE html>
<html>
<head>
    <script>
        function cambiarTexto() {
            document.getElementById("prueba").innerHTML = "Otro texto";
        }
    </script>

</head>
<body>
    <p id="prueba">Texto original</p>
    <button type="button" onclick="cambiarTexto()">Cambia el texto</button>
    <br />
    <script>document.write("<h2>Haciendo pruebas.....</h2>");</script>
    <script>window.alert("hola");</script>
    <script>window.print();</script>
</body>
</html>
```

Declaraciones y sintaxis

- Las declaraciones están compuestas por valores, operadores, expresiones, palabras clave y comentarios.
- Deben terminar en punto y coma `;`.
- Los bloques de código van entre llaves `{ }`.
- Los espacios en blanco múltiples se ignoran.
- Para declarar variables se usa la palabra clave *var* o *let*.

- Los literales numéricos decimales usan el punto '.' para separar la parte entera de la decimal.
- Las cadenas van entre comillas simples `' '` o dobles `" "`.
- `//` para comentarios de línea y `/* */` para comentarios de bloque.
- Los identificadores deben comenzar por una letra, un guion bajo `_` o el signo de dólar `$`. Los siguientes caracteres a estos pueden ser números.
- Javascript es sensible a mayúsculas.
- Se recomienda nombrar a los identificadores usando lower camel case.

Variables

- Las variables se declaran con la palabra reservada `var` o `let`, seguido del nombre de las variables, separadas por comas y terminado en `;`.
- Se permite declarar e inicializar variables en la misma sentencia.
- Si en una expresión aritmética aparece un valor entre comillas todos los números se tratarán como strings y el resultado será la concatenación.

```
<!DOCTYPE html>
<html>
<body>
    <p id="coche"></p>
    <p id="total"></p>
    <p id="concat"></p>
    <script>
        var persona = "Pepe";
        var coche = "Volvo",
        precio = 200;
        var cantidad;
        cantidad = 20;
        var x = 5 + "1";
        document.getElementById("coche").innerHTML = coche;
        document.getElementById("total").innerHTML = precio * cantidad;
        document.getElementById("concat").innerHTML = x;
    </script>
</body>
</html>
```

Volvo

4000

51

- La diferencia entre `var` y `let` radica en el alcance de la variable:
  - Una variable declarada con `var` tiene alcance global,por lo que existirá en el bloque en el que fue declarada y también en sus subbloques.
  - Una variable declarada con `let` tiene alcance local al bloque en el que fue declarada, por lo que solo existirá en dicho bloque.

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

<body>
    <script>

    var a = 5;
    var b = 10;

    console.log('a inicial: ' + a);
    console.log('b inicial: ' + b);

    if (a == 5) {
      let a = 4;
      var b = 1;

      console.log('a en el if: ' + a);
      console.log('b en el if: ' + b);
    }

    console.log('a final: ' + a);
    console.log('b final: ' + b);


    </script>
</body>
</html>
```

```
a inicial: 5
b inicial: 10
a en el if: 4
b en el if: 1
a final: 5
b final: 1
```

Hoisting

- *Hoisting* es el término que se utiliza en Javascript para referirse a la propiedad de este lenguaje que permite usar una variable antes de su declaración.
- Aunque es bueno conocer que existe esta propiedad, su uso no está recomendado.

```html
<!DOCTYPE html>
<html>
<body>
<script>
      x = 5; // Assign 5 to x

      document.write(x);

      var x; // Declare x
      document.write("<br />" + x);
</script>
</body>
</html>
```

```
5
5
```

Constantes

- Con la palabra reservada *const* podremos declarar constantes, las cuales son identificadores con un valor asignado el cual no puede ser cambiado.

```
const PI = 3.141592653589793;
```

Operadores

- Aritméticos: +, -, *, /, ++, --, % (módulo), ** (exponenciación).
- Asignación: =, +=, -=, *=, /=, %=, **=
- Cadenas: +(concatenación).
- Comparación: ==, ===(igualdad de valor y de tipo), !=, !==, <, >, <=, >=, ? (ternario)
- Lógicos: &&, ||, !

Tipos de datos

- Javascript implementa múltiples tipos de datos, como cadenas, numéricos, objetos, etc.
- Si mezclamos en una expresión cadenas y números, el resultado será una cadena.
- Los tipos en Javascript son dinámicos, por lo que a una variable se le pueden asignar valores de diferentes tipos.
- Las cadenas van entre comillas simples o dobles y se pueden usar unas comillas dentro de otras,
- Con los valores numéricos se usa punto `.` para los decimales y `e` para el formato exponencial.
- Los valores booleanos son *true* y *false*.
- Las matrices se declaran con sus valores entre corchetes.
- Los objetos se declaran con sus atributos entre llaves.
- El tipo de datos de una variable no inicializada es *undefined*.
- Operadores de tipo:
  - *typeof*, devuelve el tipo de una variable.
  - *instanceof*, devuelve true si un objeto es una instancia de un tipo de objeto.

- *null* es un objeto que representa algo que no existe. Permite vaciar el contenido de un objeto.
- *null* y *undefined* son iguales en valor pero no en tipo. El tipo de *null* es *object* y el de *undefined* es *undefined*.
- Los tipos de datos primitivos en Javascript son:
  - *string*
  - *number*
  - *boolean*
  - *Array*
  - *Object*
  - *undefined*
- Datos complejos:
  - *function*
  - *object*

```
<!DOCTYPE html>
<html>
<body>
    <script>
        var x; document.write(x + " ");
        x = 5; document.write(x + " ");
        x = "hola"; document.write(x + "<br />");

        var cadena = "today it's sunny";
        document.write(cadena + "<br />");

        var x = 1e5;
        var y = 23e-5;
        document.write("x = " + x + "-" + " y = " + y
+"<br />");

        document.write((x > y) + "<br />");

        var coches = ["seat", "audi", "bmw"];
        document.write(coches[2] + "<br />");

        var personaObj = {
            nombre: "Pepe",
            apellido: "Perez",
            edad: 25
            };
        document.write(personaObj.nombre + " " +
personaObj.edad + "<br />");

        document.write(typeof(personaObj.nombre) +
"<br />");

        personaObj = null;
        document.write(typeof(personaObj));
    </script>
```

```
undefined 5 hola
today it's sunny
x = 100000- y = 0.00023
true
bmw
Pepe 25
string
object
```

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

Conversión entre tipos de datos

- Aunque JavaScript realiza conversiones entre tipos de forma transparente al programador, existen funciones para realizar de forma explícita algunas conversiones de tipos:
  - *parseInt()*, convierte una cadena a entero.
  - *parseFloat()*, convierte una cadena a decimal.
  - *toDateString()*, convierte una cadena a formato fecha.
  - *toUTCString()*, convierte la fecha y hora a formato cadena UTC.

```
<!DOCTYPE html>
<html>

<body>
    <script>
      document.write('<h2>' + "Entero: " + parseInt("5.25") + '</h2');
      document.write('<br />');

      document.write('<p style="font-size:40px">' + "Decimal: " +
parseFloat("5.25") + '</p>');
      document.write('<br />');

      document.write((new Date) + '<br />');

      var fecha = (new Date()).toDateString();
      document.write(fecha + '<br />');

      var hora = (new Date()).toUTCString();
      document.write(hora + '<br />');
    </script>
</body>
</html>
```

Funciones

- Una función es un bloque de código que ejecuta una tarea y devuelve un valor una vez invocada. Puede tener parámetros de entra da o no.
- Sintaxis:

```
function nombre(param1, param2, …) {
      …
      return valor;
}
```

```
<!DOCTYPE html>
<html>
<body>
    <script>
        function toCelsius(f) {
          return (5/9) * (f-32);
        }

        document.write(toCelsius + "<br />");
        document.write(toCelsius(60));
    </script>
</body>
</html>
```

```
function toCelsius(f) { return (5/9) * (f-32); }
15.555555555555557
```

- `toCelsius` se refiere al objeto función, mientras `toCelsius()` invoca la función.
- El paso de parámetros se realiza por valor cuando estos valores son de tipos de datos primitivos. En otro caso el paso es por referencia.

```
<!DOCTYPE html>
<html>
<body>
    <script>
        function duplicaVector(v){
            v[0] *= 2;
            v[1] *= 2;
        }
        var v = [2,3];
        duplicaVector(v);
        document.write(v[0] + " " +v[1]);
    </script>
</body>
</html>
```

```
4 6
```

Funciones anónimas

- Es posible almacenar una función en una variable, lo que se conoce como función anónima, que y no se asigna ningún nombre a la función, sino que la función se asigna a una variable.
- El uso de funciones anónimas nos permite tratar a las funciones como si fueran variables.
- También, con el uso de funciones anónimas, se pueden crear funciones autoinvocadas, las cuales no necesitan ser llamadas para ejecutarse. Su uso es común cuando se trata de funciones que solo serán ejecutadas una vez.
- El uso de funciones anónimas permite:
  - Pasar funciones como parámetro.
  - Mayor encapsulamiento evitando el uso de variables globales.

```
<!DOCTYPE html>
<html>
<body>
<script>

     // asingacion de una funcion anonima a una variable
     let f = function (a,b) {
          return a + b
     };
     document.write(f(3,1));

     // ejecucion de una funcion sin ser invocada
     (function () { a = 5; document.write("<br />" +
a**2) }) ();

     // funciones anónimas con parámetros
     x = 5;
     y = 10;
     // encapsulación de variables
     z = (function (x,y) { return x + y } (1, 2));
     document.write("<br />" + z);

     document.write("<br />" + eval(x + y));

     var cadena1 = "<br />Hola";
     var cadena2 = " y adios";
     (function (str1, str2) {document.write(str1 +
str2);}) (cadena1, cadena2);
```

```
4
25
3
15
Hola y
adios
```

```
</script>
</body>
</html>
```

Funciones flecha

- Permiten usar una sintaxis más corta para la definición de funciones.
- Sintaxis:

```
[let | var] nombreFun = (param1, param2, ... ) => expresion
```

- Lo anterior crea una función con un nombre, la cual acepta un número de parámetros, los cuales serán usados en la expresión para devolver un resultado. Es una versión abreviada de la forma tradicional:

```
[let | var] nombreFun = (param1, param2, ... ) {
   return expresion;
   }
```

- Cuando la función flecha consta solamente de la expresión que devuelve el valor, no es necesario le uso de *return* (va implícito). En otro caso habrá que usar *return* explícitamente.

```
<!DOCTYPE html>
<html>
<body>
<script>

     // Funcion tradicional
     function hola() {
          return "hola1";
     }
     document.write(hola() + "<br />");

     // Funcion Anonima
     var hola;
     hola = function() {
       return "hola2";
     };
     document.write(hola());

     // Funciones flecha
```

```
hola1
hola2
hola3
hola4
Hola Pepe
16
Resultado:
12
```

```
    hola = () => {return "hola3";}
    document.write("<br />" + hola());

    hola = () => "hola4";
    document.write("<br />" + hola());



    var nombre = "Pepe";
    saludar = saludo =>  saludo + nombre;
    document.write("<br />" +
saludar("Hola"));


    sumar = (x, y) => x + y;
    document.write("<br />" + sumar(6,
10));

    multiplicar = (x, y) => {
        document.write("<br /> Resultado:
");
        return x * y;
    }
    document.write("<br />" +
multiplicar(2, 6));

</script>
</body>
</html>
```

Strings

- Las variables cadena se declaran usando las comillas ya sean dobles o simples, como se ha podido ver en los ejemplos anteriores.
- Podemos usar los caracteres de escape `\'`, `\"` o `\\`, para usar comillas simples, dobles o backslash dentro de una cadena, respectivamente.
- Otra secuencia de escape muy común es `\n`, para salto de línea.

- Aunque string es un tipo de datos primitivo, también puede ser un objeto, aunque no se recomienda usarlo como tal.

Métodos de Strings

- *lenght*: devuelve el número de caracteres que forman un string.
- *indexOf()*, *search()*: devuelve la posición de la primera ocurrencia de una cadena dentro de otra.
- *lastIndexOf()*: ídem, pero con respecto a la última ocurrencia.
- *slice()*, *substring()*, *substr()*: extraen un trozo de un string.
- *replace()*: reemplaza un valor en un string por otro valor, pero no modifica el string original.
- *toUpperCase()*, *toLowerCase()*: pasa un string a mayúsculas y minúsculas, respectivamente.
- *concat()*: concatenación de strings.
- *trim()*: quita los espacios en blanco de los extremos de un string.
- *charAt()*: devuelve el carácter dada una posición.
- *charCodeAt()*: devuelve el carácter unicode dada una posición.
- *split()*: convierte un string en un array. Hay que especificar un separador de elementos.

```
<!DOCTYPE html>
<html>
<body>
      <script>
            var txt = "Esto es una prueba de metodos de cadena";
            document.write("Longitud de txt: ");
            document.write(txt.length);

            document.write("<br />" + "Posicion de la cadena \"de\": ");
            document.write(txt.indexOf("de"));

            document.write("<br />" + "Extracion de texto entre pos 5 y 9: ");
            document.write(txt.slice(5,9));

            document.write("<br />" + "Reemplazo de 'cadena' por 'string': ");
            document.write(txt.replace("cadena","string"));

            document.write("<br />" + "Caracter en posicion 8: ");
            document.write(txt.charAt(8));
            document.write("<br />" + "Codigo UNICODE en posicion 8: ");
            document.write(txt.charCodeAt(8));

            document.write("<br />" +
                        "Convertir txt en un vector usando el espacio como
```

```
separador");
            var v = txt.split(" ");
            document.write("<br />" + "Elemento 3: " + v[3]);
      </script>
</body>
</html>
```

Longitud de txt: 39
Posicion de la cadena "de": 19
Extracion de texto entre pos 5 y 9: es u
Reemplazo de 'cadena' por 'string': Esto es una prueba de metodos de string
Caracter en posicion 8: u
Codigo UNICODE en posicion 8: 117
Convertir txt en un vector usando el espacio como separador
Elemento 3: prueba

Números

- Javascript solo tiene un tipo numérico, tanto para enteros como para reales.
- El operador suma esta sobrecargado y permite tanto realizar la suma aritmética como la concatenación de cadenas.
- Si “sumamos” un valor numérico con un número entre comillas, el valor numérico se convertirá en un *string* y se concatenarán los dos valores.
- Con el resto de operadores aritméticos Javascript convertirá los valores *string* en valores numéricos y realizará la operación.
- Las operaciones aritméticas con un resultado no válido, devuelven *NaN* (Not a Number).
- *Infinity* y *-Infinity* es el valor que devuelve Javascript si el resultado es un valor fuera de rango división entre cero.
- Los numeros hexadecimales van precedidos por *0x*.
- Los números son tipos primitivos, aunque puedan ser tratados como objetos. No se recomienda.

```
<!DOCTYPE html>
<html>
<body>
      <script>
            var x = 10;
            var y = "20";
            var z = "30";
            var res;

            res = x + y;
            document.write(res);
```

1020
2030
-10
NaN
Infinity

```
            res = y + z;
            document.write("<br />" + res);

            res = y - z;
            document.write("<br />" + res);

            res = y / "hola";
            document.write("<br />" + res);

            res = x / 0;
            document.write("<br />" + res);
      </script>
</body>
</html>
```

Métodos numéricos

- *toString()*: convierte un número en string. Si se pasa un valor numérico como parámetro, el número se cambiará a la base especificada por dicho parámetro.
- *toExponential()*: devuelve un string del número en formato exponencial.
- *toFixed()*: devuelve un string con el número escrito con tantos decimales como se especifique. Usa redondeo.
- *toPrecision()*: devuelve una cadena con tantos dígitos como se le haya especificado. Usa redondeo.
- *Number()*, *parseInt()* y *parseFloat()*: devuelven su argumento convertido a un numero, a un entero o a un flotante, respectivamente..

```
<!DOCTYPE html>
<html>
<body>
      <script>
            var entero = 7;
            var real = 7.96365;

            document.write(entero.toString(2));
            document.write("<br />" + real.toExponential(3));
            document.write("<br />" + real.toFixed(4));
            document.write("<br />" + real.toPrecision(4));
            document.write("<br />" + Number("true"));
            document.write("<br />" + Number("   154   "));
            document.write("<br />" + parseInt("22.36"));
            document.write("<br />" + parseFloat("22.36"));
      </script>
</body>
</html>
```

```
111
7.964e+0
7.9637
7.964
NaN
154
22
22.36
```

Booleanos

- Los valores que puede tomar una expresión booleana son *true* o *false*.
- *0* equivale a *false* y cualquier otro valor *true*.
- El valor que devuelve una expresión formada por operadores de comparación es un valor booleano.
- El operador ternario permite asignar un valor a una variable

```
<!DOCTYPE html>
<html>
<body>
<script>

      var x = 5;
      var y = "5";

      document.write(x == y);
      document.write("<br />" + (x === y));
      document.write("<br />" + (x !== y));

      edad = 19;
      var votar = (edad < 18) ? "No puedes
votar":"Puedes votar";
      document.write("<br />" + votar);
</script>
</body>
</html>
```

```
true
false
true
Puedes votar
```

Condicionales

- Podemos usar las siguientes expresiones para evaluar condiciones:
    - *if*
    - *if - else*
    - *if - else if - else*
    - *switch*

```
<!DOCTYPE html>
<html>
<body>
<script>

      var x = 5;
```

```
x es menor que y
fin de semana
```

```
	var y = 6;

	if (x > y) {
		document.write("x es mayor que y");
	}else if (x < y){
		document.write("x es menor que y");
	}else{
		document.write("x es igual a y");
	}

	document.write("<br />");

	dia = 6;
	switch(dia) {
	case 1: document.write("lunes");break;
	case 2: document.write("martes");break;
	case 3: document.write("miercoles");break;
	case 4: document.write("jueves");break;
	case 5: document.write("viernes");break;
	default: document.write("fin de semana");break;
	}

</script>
</body>
</html>
```

Bucles

- Los bucles que podemos usar en Javascript son:
  - *for*: itera un número determinado de veces.
  - *for/in*: itera todas las propiedades de un objeto.
  - *for/of*: itera todos los valores de un objeto iterable.
  - *while*: itera cero o más veces en función de una condición
  - *do/while*: itera una o más veces en función de una condición.

```
<!DOCTYPE html>
<html>
<body>
<script>

	for (i = 0; i < 5; i++){
		if (i == 2){ break; }
		document.write(i + " ");
	}
	document.write("<br />");

	for (i = 0; i < 5; i++){
		if (i == 2){ continue; }
		document.write(i + " ");
```

```
0 1
0 1 3 4
pepe perez 25
seat audi bmw
e s t o e s u n a c a d e n a
0 1 2 3 4
1 2 3 4 5 6
```

```
	}
	document.write("<br />");

	var persona = {n: "pepe", a: "perez", e: 25};
	for (i in persona){
		document.write(persona[i] + " ");
	}
	document.write("<br />")

	var coches = ["seat","audi","bmw"];
	for (marca of coches){
		document.write(marca + " ");
	}
	document.write("<br />")

	var cadena = "esto es una cadena";
	for (c of cadena){
		document.write(c + " ");
	}
	document.write("<br />")

	var x = 0;
	while (x < 5) {
		document.write(x++ + " ");
	}
	document.write("<br />")

	x = 0;
	do{
		if (x > 5) { break; }

		document.write(++x + " ");
	}while(true);

</script>
</body>
</html>
```

Arrays

- Se usan para almacenar múltiples valores en una única variable.
- Sintaxis:

```
var nombre_array = [elem1, elem2, …];
```

- Se pueden crear como objetos con la palabra reservada *new*, aunque no se recomienda.
- Se pueden mezclar valores de distinto tipo en un mismo array.
- Además de usar bucles convencionales, los arrays también aceptan *forEach*.

Métodos de arrays

- *toString()*: convierte un array en un string con los elementos separados por comas.
- *join()*: se comporta como *toString()*, pero especificando el separador.
- *push()* y *pop()*: añadir y retirar elementos de un array por el final, respectivamente.
- *Shift()* y *unshift()*: eliminar y añadir elementos por el principio del array, respectivamente.
- *delete*: permite borrar elementos. No varía el tamaño del array, solo elimina el elemento.
- *splice()*: permite añadir elementos a un array especificando tanto el punto de inserción como si queremos eliminar elementos.
- *concat()*: para unir arrays.
- *slice()*: crea un nuevo array a partir de una sección de otro.
- *split()*: permite crear un array a través de un objeto *String*, indicando un carácter que permita separar los elementos del *String*.

```
<!DOCTYPE html>
<html>
<body>
      <script>
            var colores = ["rojo","azul","verde"];
            var vector = ["hola",343,7.656,true];

            document.write(colores[0]);
            document.write("<br />" +
colores[colores.length-1]);
            document.write("<br />" + vector +
"<br />");

            document.write("<ul>");
            for (i = 0; i< vector.length; i++){
                  document.write("<li>" +
vector[i] + "</li>");
            }
            document.write("</ul>");

            colores.forEach(valor =>
document.write(valor.toUpperCase()+ " "));

            vector.push("otro elemento");
            document.write("<br />" + vector);
            vector.pop();
            document.write("<br />" + vector);

            vector.unshift("otro elemento");
            document.write("<br />" + vector);
```

rojo
verde
hola,343,7.656,true

- hola
- 343
- 7.656
- true

ROJO AZUL VERDE
hola,343,7.656,true,otro elemento
hola,343,7.656,true
otro elemento,hola,343,7.656,true
hola,343,7.656,true
hola,343,,true
4
rojo,azul,amarillo,naranja,violeta,verde
rojo,azul,gris,naranja,violeta,verde
rojo,azul,naranja,violeta,verde
rojo,azul,naranja,violeta,verde,hola,343,,true
azul,naranja,violeta,verde,hola,343

```
            vector.shift();
            document.write("<br />" + vector);

            delete vector[2];
            document.write("<br />" + vector);
            document.write("<br />" +
vector.length);



colores.splice(2,0,"amarillo","naranja","violeta")
;
            document.write("<br />" + colores);
            colores.splice(2,1,"gris");
            document.write("<br />" + colores);
            colores.splice(2,1);
            document.write("<br />" + colores);

            var union = colores.concat(vector);
            document.write("<br />" + union);

            var trozo = union.slice(1,7);
            document.write("<br />" + trozo);


      </script>
</body>
</html>
```

Ordenación de arrays

- *sort()*: ordena el array alfabéticamente.
- *reverse()*: ordena el array alfabéticamente de manera inversa.
- *Math.max.apply()* y *Math.min.apply()*: se pueden usar para calcular el máximo y el mínimo elemento de una array, respectivamente.

```
<!DOCTYPE html>
<html>
<body>
      <script>
            var cadenas = ["aabc","abc","zdc","ew"];
            var numeros = [4,40,2,220,1,10];

            document.write("<br />" + cadenas.sort());
            document.write("<br />" + numeros.sort() + "
<--!!!");

            /*
            Sort pasa dos valores a la funcion de comparacion y
            ordena los valores segun el valor que le venga
devuelto
            Si la funcion devuelve un positivo, b va antes que a,
            si devuelve un negativo a va antes que b
            */
            document.write("<br />" + numeros.sort(
```

```
aabc,abc,ew,zdc
1,10,2,220,4,40
<--!!!
1,2,4,10,40,220
1,2,4,220,40,10
220
```

```
                        function(a,b){return a - b}));

            /*
            Coloca los valores de forma aleatoria
            */
            document.write("<br />" + numeros.sort(
                        function(a,b){return 0.5 -
Math.random()}));

            /*
            Devolver el maximo de un array
            */
            function maximo(numeros){return Math.max.apply(null,
numeros)};
            document.write("<br />" + maximo(numeros));

      </script>
</body>
</html>
```

Iteración sobre arrays

- *forEach()*: realiza una llamada a una función para cada elemento. La función recibe los parámetros valor actual, índice y el propio array.
- *map()*: crea un nuevo array realizando una función sobre otro. La función recibe los parámetros valor actual, índice y el propio array.
- *filter()*: crea un nuevo array con los elementos que cumplen una condición. La función recibe los parámetros valor actual, índice y el propio array.
- *reduce()*: permite obtener un único valor calculado sobre todos los elementos del array. La función recibe, valor actual, índice, el propio array y un parámetro para almacenar el resultado.
- *every()*: comprueba si todos los elementos de un array pasan un test. La función recibe los parámetros valor actual, índice y el propio array.
- *some()*: comprueba si al menos un elemento de un array pasa un test. La método recibe la función y la aplica al array, devolviendo true si algún elemento la cumple y false en otro caso.
- *find()* y *findIndex()*: *find()* devuelve el valor del primer elemento que cumple una condición y *findIndex()* su posición. Las funciones reciben los parámetros valor actual, índice y el propio array.

```
<!DOCTYPE html>
<html>
<body>
```

```
Pos 0 = 45
Pos 1 = 4
Pos 2 = 9
Pos 3 = 16
```

```
	<script>

	var numeros = [45, 4, 9, 16, 25];

	function mostarVector(valor, indice, vector) {
		document.write("Pos " + indice +" = " + valor + "<br
/>");
	}
	numeros.forEach(mostarVector);

	function porDos(valor){
		return valor * 2;
	}
	var doble = numeros.map(porDos);
	document.write(doble);

	function filtrar(valor){
		return valor > 10;

	}
	var filtro = numeros.filter(filtrar);
	document.write("<br />" + filtro);

	function sumar(total, valor, indice, vector){
		return total + valor;
	}
	var sumatorio = numeros.reduce(sumar);
	document.write("<br />" + sumatorio);

	function comprobarTodos(valor){
		return valor > 10;
	}
	document.write("<br />" + numeros.every(comprobarTodos));

	function comprobarAlgunos(valor){
		return valor > 10;
	}
	document.write("<br />" + numeros.some(comprobarAlgunos));


	function encontrar(valor){
		return valor == 9;
	}
	document.write("<br />" + numeros.find(encontrar) + " pos= " +
			numeros.findIndex(encontrar));

	</script>
</body>
</html>
```

```
Pos 4 = 25
90,8,18,32,5
0
45,16,25
99
false
true
9 pos= 2
```

Arrays de dos dimensiones

- Para arrays de más de una dimensión usaremos el constructor *`new Array()`*.

```html
<!DOCTYPE html>
<html>
<body>


<script>
	// array de 4 x 4
	var matriz = new Array(4);
	for (i = 0; i < matriz.length; i++){
		matriz[i] = new Array(4);
	}

	// insercion de valores
	for(i = 0; i < matriz.length; i++){
		for (j = 0; j < matriz[i].length; j++){
			matriz[i][j] = Math.ceil(Math.random() * 9);
		}
	}

	// recorrido de la matriz
	for(i = 0; i < matriz.length; i++){
		for(j = 0; j < matriz[i].length; j++){
			document.write(matriz[i][j] + " ");
		}
		document.write("<br />");
	}
</script>

</body>
</html>
```

```
4 3 3 5
7 9 5 4
7 3 7 1
1 5 3 6
```

Fecha/hora y manejo del tiempo

- Podemos obtener la fecha y la hora a través del objeto *`Date()`*.

```
var d = new Date();
```

Sat Apr 11 2020 13:39:11 GMT+0200 (CEST)

- El formato estándar en el que se usan las fechas en Javascript es

YYYY-MM-DDTHH:MM:SS

- Javascript proporciona varios métodos a traveś del objeto Date() para acceder a los campos que forman la fecha y la hora, de los cuales destacamos los siguientes:
  - *`getFullYear()`*: YYYY

- `getMonth()`: MM (0 - 11)
- `getDate()`: DD (1 – 31)
- `getHours()`: HH (0 – 23)
- `getMinutes()`: MM (0 – 59)
- `getTime()`: millisegundos desde 01/01/1970

- Existen métodos `set` para configurar un valor de fecha en una variable de tipo Date(), así como varios constructores:
-
  - `new Date()`
  - `new Date(milisegundos)`
  - `new Date(fechaString)`
  - `new Date (año, mes, día, horas, minutos, segundos, milisegundos)`

```
<!DOCTYPE html>
<html>
<body>
<script>

      var d = new Date();
      document.write(d.getTime());

      document.write("<br />" + d.getFullYear());
      document.write("<br />" + d.getMonth());
      document.write("<br />" + d.getMinutes());

      d.setFullYear(2030);
      document.write("<br />" + d.getFullYear());

</script>
</body>
</html>
```

```
1586606068396
2020
3
54
2030
```

- El objeto `window` proporciona los siguientes métodos que, junto con los anteriores, permiten manejar el tiempo en el navegador a través de Javascript:

  - `setTimeout (funcion, milisecs)`: ejecuta la función una vez que ha transcurrido el tiempo indicado.
  - `setInterval (funcion, milisecs)`: ejecuta la función periódicamente, en función del tiempo indicado.
  - `clearInterval()`: para la ejecución de `setInterval()`.
  - `clearTimeout()`: para la ejecución de `setTimeout()`.

- Podemos utilizar el evento `window.onload` para inicializar `setTimeout()` y `setInterval()`. Este evento sirve para ejecutar funciones una vez que la página ha cargado completamente.

```
<html>
<script>

let intervalo = setInterval(funCB,500);

setTimeout(parar,5000);

function funCB() {
document.write("holaaaaa");
}

function parar() {
clearInterval(intervalo);
}

</script>
</html>
```

```
<html>
<script>
let intervalo = setInterval(() =>
document.write("holaaaaa"),500);

setTimeout(() => clearInterval(intervalo),5000);
</script>
</html>
```

Math

- Entre las funciones matemáticas que proporciona Javascript a través del objeto `Math`, destacamos las siguientes:

    - `round()`: redondeo al entero más cercano.
    - `pow()`: potencia.
    - `sqrt()`: raíz cuadrada.
    - `abs()`: valor absoluto.
    - `ceil()`: entero más cercano por exceso.
    - `floor()`: entero más cercano por defecto.
    - `min()`: mínimo valor de una lista.
    - `max()`: máximo valor de una lista.
    - `random()`: numero aleatorio en el rango [0 -1).
    - Constantes `PI` y `E`.

```html
<!DOCTYPE html>
<html>
<body>
<script>
	document.write(Math.max(2,5,2,4,5,7,4));
	document.write("<br />" + Math.pow(7,5));
	document.write("<br />" + Math.ceil(Math.random() * 10));
</script>
</body>
</html>
```

```
7
16807
9
```

Expresiones regulares

- Una expresión regular es una secuencia de caracteres que forman un patrón de búsqueda.
- La sintaxis en Javascript es la siguiente: `/patron/modificadores`
- Modificadores:
  - `i`: no sensible a mayúsculas
  - `g`: búsqueda global. Encuentra todas las coincidencias en lugar de parar en la primera.
  - `m`: búsqueda multilínea.
- Patrones:
  - `[abc]`: encuentra todos los caracteres entre corchetes.
  - `[0-9]`: encuentra todos los dígitos en el rango.
  - `(x|y)`: encuentra cada una de las alternativas.
- Cuantificadores:
  - `n+`: encuentra cualquier cadena que contenga, al menos, una n.
  - `n*`: encuentra cualquier cadena que contenga cero o más n.
  - `n?`: encuentra cualquier cadena que contenga cero o una n.
- Métodos del objeto `String`:

- `search()`: busca una cadena en otra y devuelve la posición.
- `replace()`: remplaza una cadena con otra.
- `Match()`: busca un string contra un patrón de búsqueda y devuelve el elemento encontrado como una array.

- Métodos del objeto `RegExp`:

  - `test()`: busca un `string` que cumpla un patrón y devuelve `true` o `false`.
  - `exec()`: busca un `string` que cumpla un patrón y devuelve el texto encontrado.

```
<!DOCTYPE html>
<html>
<body>
<script>

      var txt = "En un lugar de La Mancha...";
      document.write(txt.search("de"));
      document.write("<br />" + txt.replace("de",
"DE"));

      document.write("<br />" + txt.search(/e la/i));
      document.write("<br />" + txt.replace(/e la/i, "E
LA"));

      document.write("<br />" + txt.match(/lu+/i));

      document.write("<br />" + (/lu+/i).test(txt));

      var patron = /lu+/i;
      document.write("<br />" + patron.exec(txt));

</script>
</body>
</html>
```

```
12
En un lugar DE La
Mancha...
13
En un lugar dE LA
Mancha...
lu
true
lu
```

Control de errores y excepciones

- Las excepciones que se puedan generar durante la ejecución del código pueden ser manejadas mediante mediante bloques `try - catch`:

  - `try`: permite probar un bloque de código en busca de posibles excepciones.

- *Catch*: permite especificar un bloque de código que se ejecutará en caso de que haya un error o excepción.
- *throw*: permite crear un manejador a medida.
- *finally*: permite ejecutar código por defecto después del bloque *try – catch*, independientemente del resultado.

- El objeto Error nos dará información referente al tipo de error que se ha producido, volcando dicho error en la variable que le pasemos a *catch*.

```
<!DOCTYPE html>
<html>
<body>
      <script>
      try {
        muestra("holaaa!");
      }
      catch(err) {
        document.write(err.message);
      }
      </script>

</body>
</html>
```

Can't find variable: muestra

Validación de entrada de datos

```
<!DOCTYPE html>
<html>
<body>

<p>Introduzca un numero entre 50 y 100:</p>

<input id="dato" type="text">
<button type="button"
onclick="miFuncion()">Comprobar valor</button>

<p id="err_txt"></p>

<script>
function miFuncion() {
  var err_msg, x;
  err_msg = document.getElementById("err_txt");
  err_msg.innerHTML = "";
  x = document.getElementById("dato").value;
  try {
    if(x == "")  throw "No hay dato";
    if(isNaN(x)) throw "El dato no es un numero";
    x = Number(x);
    if(x > 100)   throw "Numero muy grande";
```

Introduzca un numero entre 50 y 100:

144 Comprobar valor

Error: Numero muy grande

```
    if(x < 50)  throw "Numero muy pequeno";
  }
  catch(err) {
    err_msg.innerHTML = "Error: " + err;
  }
  finally {
    document.getElementById("dato").value = "";
  }
}
</script>

</body>
</html>
```

HTML DOM

- A través del HTML DOM (Document Object Model), Javascript puede acceder y cambiar todos los elementos de un documento HTML.
- En el DOM todos los elementos HTML son objetos. Una propiedad es un valor que se puede cambiar y un método es una acción que se puede realizar.

```
<!DOCTYPE html>
<html>
<body>

<p id="prueba"></p>

<script>
      document.getElementById("prueba").innerHTML = "Hola";
</script>

</body>
</html>
```

Hola

- En el ejemplo anterior observamos los siguientes elementos:
    - *document*: es el elemento raíz del documento HTML. Es el objeto que representa toda la página web.
    - *getElementById*: es un método que busca elementos HTML por su ID.

- *innerHTML*: es una propiedad de los elementos HTML que se refiere al contenido que hay entre las etiquetas.
- *p*: elemento HTML con el que se está trabajando.

- Lo que se hecho es localizar en el documento un elemento con *ID="prueba"*, que ha resultado ser un párrafo. Se ha accedido a su contenido, inicialmente vacío, y se ha insertado entre las etiquetaras * *, el texto "Hola".

Manejo de elementos a través del DOM

- Encontrar elementos HTML:

  - *document.getElementById(id)*
  - *document.getElementsByTagName(nombre)*
  - *document.getElementsByClassName(nombre)*

- Cambiar el valor de elementos HTML:

  - *element.innerHTML = nuevo_contendio*
  - *element.attribute = nuevo_valor*
  - *element.style.propiedad = nuevo_estilo*
  - *element.setAttribute(atributo, valor)*

- Añadir y borrar elementos:

  - *document.createElement(elemento)*
  - *document.removeChild(element)*
  - *document.appendChild(element)*
  - *document.write(texto)*

```
<!DOCTYPE html>
<html>
<body>

<p>Hola</p>
<p>Adios</p>

<p class="mi_parrafo">Un parrafo cualquiera</p>
<p class="mi_parrafo">Otro parrafo cualquiera</p>
```

Hola

Adios

Un parrafo cualquiera

Otro parrafo cualquiera

Texto en pos 0 : Hola
Texto en pos 1 : Adios
Texto en pos 2 : Un parrafo

```
<script>
      var parrafos = document.getElementsByTagName("p");
      var i = 0;
      while(parrafos[i]){
            document.write("<br />");
            document.write("Texto en pos  " + i + " : "
                        + parrafos[i].innerHTML);
            i++;
      }

      document.getElementsByClassName("mi_parrafo")[0].
            style.fontWeight="900";
      document.getElementsByClassName("mi_parrafo")[1].
            style.fontWeight="900";

</script>

</body>
</html>
```

cualquiera
Texto en pos 3 : Otro parrafo
cualquiera

- Podemos acceder a elementos a través de selectores

```
<!DOCTYPE html>
<html>
<body>

<p class="c1">Hola</p>
<p class="c2">Adios</p>

<script>
      var x = document.querySelectorAll("p.c1");
      document.write(x[0].innerHTML);
</script>

</body>
</html>
```

Hola

Adios

Hola

- Cambio del valor de un atributo:

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif" width="160" height="120">

<script>
      document.getElementById("image").src = "landscape.jpg";
</script>

</body>
```

```
</html>
```

- Cambio de estilos CSS:

```
<!DOCTYPE html>
<html>
<body>

<p id="p1">Hola 1</p>
<p id="p2">Hola 2</p>

<script>
      document.getElementById("p1").style.visibility="hidden";
      document.getElementById("p2").style.color = "blue";
      document.getElementById("p2").style.fontFamily = "Arial";
      document.getElementById("p2").style.fontSize = "larger";
</script>

</body>
</html>
```

Hola 2

- Se pueden animar objetos con Javascript cambiando propiedades CSS. Hay que tener en cuenta que todas la animaciones deberán estar incluidas en un contenedor.

```
<!DOCTYPE html>
<html>
<style>
#contenedor {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#objeto {
  width: 50px;
  height: 50px;
  position: absolute;
  background-color: red;
}
</style>
<body>

<p><button onclick="mover()">Mover objeto</button></p>
```

```
<div id ="contenedor">
  <div id ="objeto"></div>
</div>

<script>
function mover() {
  var elem = document.getElementById("objeto");
  var pos = 0;
  var id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + "px";
      elem.style.left = pos + "px";
    }
  }
}
</script>

</body>
</html>
```

Eventos HTML

- Un evento HTML es “algo” que sucede en un elemento HTML, como puede ser un click o que la página termine de cargar, por ejemplo.
- Con Javascript podemos programar las reacciones a dichos eventos creando manejadores de eventos.
- Sintaxis:

```
<elemento evento=”manejador”>
```

```
<!DOCTYPE html>
<html>
<body>

      <button onclick="mostarHora()">Mostar hora</button>

      <script>
            function mostarHora() {
              document.getElementById("hora").innerHTML = Date();
            }
      </script>

      <p id="hora"></p>

</body>
```

```
</html>
```

- Eventos más comunes:
  - *onchange*: un elemento HTML ha cambiado.
  - *onclick*: se ha hecho click en un elemento HTML.
  - *onmousedown*: se ha presionado un botón del ratón, pero no se ha soltado.
  - *onmouseup*: se ha soltado el botón del ratón
  - *onmouseover*: el ratón ha pasado por encima de un elemento HTML.
  - *onmousemove*: se ha movido el ratón.
  - *onmouseout*: el ratón se salido de un elemento HTML.
  - *onkeydown*: el usuario ha presionado una tecla.
  - *onload*: el navegador ha terminado de cargar la página.

```
<!DOCTYPE html>
<html>
<body onload="iniciar()">

	<script>
	function setTexto01() {
	  document.getElementById("prueba").innerHTML = "Quita el raton!!!";
	}

	function setTextoInic() {
		document.getElementById("prueba").innerHTML = "Pasa el raton...";
	}

	function iniciar(){
		document.getElementById("prueba").innerHTML = "Pasa el raton...";
	}
	</script>

<div id="prueba" onmouseover="setTexto01()" onmouseout="setTextoInic()"></div>

</body>
</html>
```

- En el ejemplo se crean dos manejadores de eventos sobre * *, uno para cuando el ratón está encima del elemento y otro para cuando deja de

estarlo. También se crea un manejador para inicializar el contenido del dicho elemento * *, el cual se ejecuta cuando termina de cargar la sección *<body>*.

- El siguiente ejemplo muestra como responder al evento *onclick* de tres formas diferentes:

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.innerHTML='Adios!'">Haz click aqui</h1>
<h1 onclick="cambiarTexto01(this)">Haz click aquí</h1>
<h1 id="prueba" onclick="cambiarTexto02()">Haz click aquí</h1>

<script>
      function cambiarTexto01(id){
            id.innerHTML = "Adios!";
      }

      function cambiarTexto02(){
            document.getElementById("prueba").innerHTML = "Adios";
      }
</script>
</body>
</html>
```

- La primera forma realiza la acción directamente, la segunda lo hace a través del manejador de eventos y la tercera, accediendo a través de un atributo.

- Los eventos *onload* y *onunload* se ejecutan cuando al terminar de cargar la página y al salir de ella, respectivamente. El método *onunload* puede ser problemático para sacar mensajes por pantalla o consola. Se recomienda usarlo para tareas de mantenimiento como pueden ser el borrado de archivos temporales al salir de la página que los ha creado

```
<!DOCTYPE html>
<html>
<body onload="comprobarCookies()" onunload="salir()"
>

<p id="prueba"></p>

<script>
function comprobarCookies() {
  var text = "";
  if (navigator.cookieEnabled == true) {
    text = "Cookies activadas.";
  } else {
    text = "Cookies no activadas.";
```

```
  }
  document.getElementById("prueba").innerHTML = text;
}

function salir() {
alert("adios");
}


</script>

</body>
</html>
```

- El evento *onchange* se ejecuta cuando un campo cambia su contenido.

```
<!DOCTYPE html>
<html>
<head>
<script>
function miFuncion() {
  var x = document.getElementById("txt");
  x.value = x.value.toUpperCase();
}
</script>
</head>
<body>

Introduzca texto: <input type="text" id="txt" onchange="miFuncion()">

</body>
</html>
```

- Mediante *onmousedown* y *onmouseup* podremos controlar las pulsaciones de los botones del ratón. Junto con *onclick*, estos eventos nos dan un gran control sobre las interacciones con con los clicks del ratón por parte del usuario.

```
<!DOCTYPE html>
<html>
<body>

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

<script>
function mDown(obj) {
  obj.style.backgroundColor = "#1ec5e5";
  obj.innerHTML = "Suelta...";
}

function mUp(obj) {
  obj.style.backgroundColor="#D94A38";
```

```
  obj.innerHTML="Gracias...";
}
</script>

</body>
</html>
```

EventListener

- El método *addEventListener()* permite asociar manejadores de eventos a un elemento específico sin tener que sobreescribir los que ya existan.
- Esto permite asociar más de un acción a un mismo evento, como puede ser, dos manejadores al evento click, por ejemplo.
- El método *removeEventListener()* permite eliminar un manejador de eventos de un evento.
- Sintaxis:

```
elemento.addEventListener(evento, funcion, capturaUsuario);
```

  - El primer parámetro es el tipo de evento
  - El segundo parámetro es la función que manejará el evento
  - El tercer parámetro es un valor booleano para especificar cómo se capturará el evento en caso de dos elementos HTML estén anidados y los dos tengan eventos asociados. Por defecto se ejecutará primero el evento más interno (bubbling). Si queremos cambiar esto, lo haremos a través de este parámetro, poniendo true al elemento más externo. Es opcional.

```
<!DOCTYPE html>
<html>
<body>

<button id="boton01">Haz click</button>
<button id="boton02">Haz click</button>

<script>
document.getElementById("boton01").addEventListener("click", function() {
  alert("Hola 01");
});

document.getElementById("boton02").addEventListener("click", miFuncion);

function miFuncion() {
```

```
  alert ("Hola 02");
}
</script>

</body>
</html>
```

- Dos formas de insertar el manejador de eventos.
- Nótese que el nombre del evento es "*click*" no "*onclick*".

- Como se ha comentado, además de que un mismo elemento tenga asociados varios manejadores, podemos añadir más de un manejador al mismo evento.

```
<!DOCTYPE html>
<html>
<body>

<button id="boton">Haz click</button>

<script>
var b = document.getElementById("boton");
b.addEventListener("click", fHola);
b.addEventListener("click", fAdios);
b.addEventListener("mouseover", fOver);
b.addEventListener("mouseout", fOut);

function fHola() {
  alert ("Hola");
}

function fAdios() {
        alert ("Adios");
}

function fOver(){
      b.style.color = "red";
      b.style.fontWeight = "bold";
}

function fOut(){
      b.style.color = "black";
      b.style.fontWeight = "normal";
}

</script>

</body>
</html>
```

- Podemos añadir manejadores a otros elementos, además que a los del DOM, como puede ser al objeto *window*.

```html
<!DOCTYPE html>
<html>
<body>

<p id="prueba"></p>

<script>
window.addEventListener("resize", tam);

function tam(){
      var p = document.getElementById("prueba");
      var res = screen.width + " x " + screen.height;
      var tam = window.innerWidth + " x " + window.innerHeight;
      p.innerHTML = "Res: " + res + " Tam: " + tam;

}
</script>

</body>
</html>
```

Res: 1920 x 1080
Tam: 679 x 380

- Para el paso de parámetros a los manejadores se usarán funciones anónimas.

```html
<!DOCTYPE html>
<html>
<body>


<button id="boton">Haz click</button>

<p id="prueba"></p>

<script>
var p1 = 35;
var p2 = 7;

document.getElementById("boton").addEventListener("click", function() {
  fun(p1, p2);
});

function fun(a, b) {
 document.getElementById("prueba").innerHTML = a * b;
}
</script>

</body>
</html>
```

- El método *removeEventListener()* desvincula un manejador de eventos de un elemento HTML.

```
<!DOCTYPE html>
<html>
<body>

<button id="boton">Haz click</button>

<p id="prueba"></p>

<script>
var cont = 0;
document.getElementById("boton").addEventListener("click", fun);


function fun() {
 document.getElementById("prueba").innerHTML =  5 * Math.random();
 cont++;

 if (cont == 3){
            document.getElementById("boton").removeEventListener("click",fun);
        }
}

</script>

</body>
</html>
```

Navegación por nodos HTML DOM

- Con HTML DOM se puede navegar a través de la estructura de árbol formada por los nodos que componen el documento HTML.
- De acuerdo con el estándar W3C HTML DOM, todo en un documento HTML es un nodo:

    - El documento es un nodo.
    - Cada elemento HTML es un nodo.
    - El texto dentro de las etiquetas es un nodo.
    - Los comentarios son nodos

- Ejemplo

```html
<html>

  <head>
    <title>Ejemplo</title>
  </head>

  <body>
    <h1>Nodos</h1>
    <p>Parrafo</p>
  </body>

</html>
```

- *<html>* es el nodo raíz.
- *<html>* es padre de *<head>* y *<body>*
- *<head>* tiene un hijo, *<title>*
- *<title>* tiene un hijo , *"Ejemplo"*
- *<body>* tiene dos hijos, *<h1>* y *<p>*
- *<head>* y *<body>* son hermanos.
- *<h1>* tiene un hijo, "Nodos"
- *<p>* tiene un hijo, "Parrafo"
- *<h1>* y *<p>* son hermanos

- Se pueden usar las siguientes propiedades para navegar entre nodos:
  - *parentNode*
  - *childNodes{numero}*
  - *children*
  - *firstChild*
  - *lastChild*
  - *nextSibling*
  - *previousSibling*

```html
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Ejemplo</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").firstChild.nodeValue;

var txt =
document.getElementById("id01").childNodes[0].nodeValue;
document.write(txt);
</script>

</body>
</html>
```

Ejemplo

Ejemplo

Ejemplo

- Hay dos propiedades especiales que permiten acceder a todo el documento:
  - *document.body*: el cuerpo del documento HTML.
  - *document.documentElement*: todo el documento HTML.

```html
<!DOCTYPE html>
<html>
<body>

<div>
<p>Ejemplo</p>
<p>Este ejemplo muestra el uso de <b>document.documentBody</b>
y document.documentElement</p>
</div>

<script>
      alert(document.body.innerHTML);
      alert(document.documentElement.innerHTML);
</script>

</body>
</html>
```

- La propiedad *nodeName* tiene las siguiente características:
    - Es de solo lectura.
    - El *nodeName* de un elemento HTML es el mismo que el de su etiqueta.
    - El *nodeName* de un atributo es el nombre del atributo.
    - El *nodeName* de un elemento texto es siempre *#text*.
    - El *nodeName* del documento es *#document*.
    - El *nodeName* devuelve el valor de las etiquetas en mayúscula.
- La propiedad *nodeValue* tiene las siguiente características:
    - El *nodeValue* de un nodo HTML es *null*.
    - El *nodeValue* de un nodo texto es el propio texto.
    - El *nodeValue* de un atributo es el valor del atributo.
- La propiedad *nodeType* devuelve el tipo de nodo:
    - *1*: *ELEMENT_NODE.*
    - *2*: *ATTRIBUTE_NODE*
    - *3*: *TEXT_NODE*
    - *8*: *COMMENT_NODE*
    - *9*: *DOCUMENT_NODE*

```html
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Ejemplo</h1>


<script>
 var x = document.getElementById("id01").nodeName;
 document.write("Nombre del nodo: " + x);

 x = document.getElementById("id01").firstChild.nodeValue;
      document.write("<br />Valor del nodo: " + x);

 x = document.getElementById("id01").nodeType;
      document.write("<br />Tipo del nodo: " + x);

</script>

</body>
</html>
```

Ejemplo
Nombre del nodo: H1
Valor del nodo: Ejemplo
Tipo del nodo: 1

- Ejemplo de acceso a los atributos de un nodo

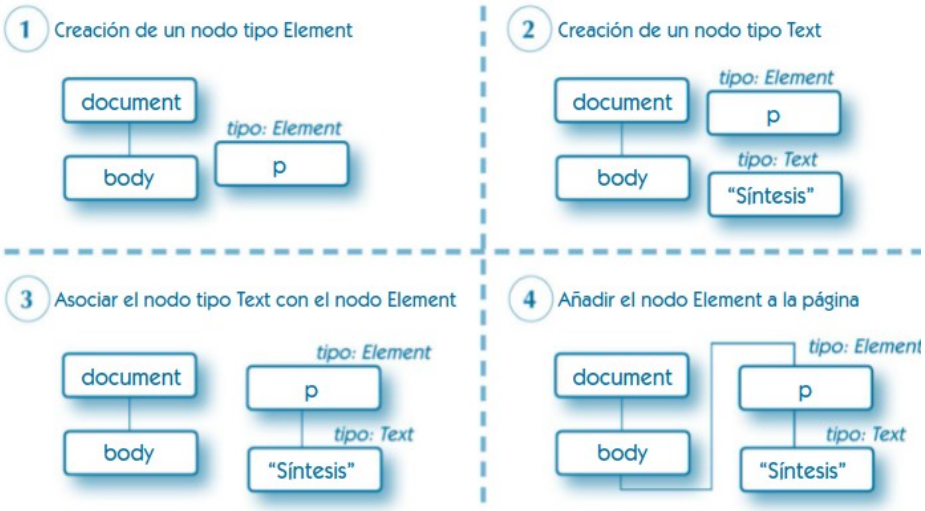
```html
<html>
<head>
<title>Ejemplo Atributos</title>
<script>
function listadoAttributos() {
var parrafo = document.getElementById("parrafo");
var resultado = document.getElementById("result");

if (parrafo.hasAttributes()) {
var attrs = parrafo.attributes;
var salida = "";
for (var i = 0; i < attrs.length; i++) {
salida += attrs[i].name + "->" + attrs[i].value + ";";
}
resultado.value = salida;
} else {
resultado.value = "No hay atributos a mostrar";
}
}
</script>
</head>
<body>
<p id="parrafo" style="color: lightgreen;">Parrafo de ejemplo</p>
<input type="button" value="Muestra el nombre cada atributo y su valor"
onclick="listadoAttributos()">
<input id="result" type="text" value="">
```

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

Nodos HTML DOM

- Podemos añadir y eliminar nodos dinámicamente con Javascript.




```
<!DOCTYPE html>
<html>
<body>

<div id="div1">
<p id="p1">Esto es un parrafo</p>
<p id="p2">Esto es otro parrafo</p>
</div>

<script>
var parrafo = document.createElement("p");
var nodo = document.createTextNode("Parrafo nuevo");
parrafo.appendChild(nodo);
var elemento = document.getElementById("div1");
elemento.appendChild(parrafo);
</script>

</body>
</html>
```

Esto es un parrafo

Esto es otro parrafo

Parrafo nuevo

- *document.Create()*, crea un elemento *<p>* y *document.createTextNode()* crea un elemento de texto. Mediante *appendChild()* se asocia el elemento de texto con *<p>*. Finalmente, el elemento *<p>* con su texto hijo, se asocia al elemento * * y se muestra en la página.

```
<!DOCTYPE html>
<html>
<body>

<div id="div1">
<p id="p1">Esto es un parrafo</p>
<p id="p2">Esto es otro parrafo</p>
</div>

<script>
var parrafo = document.createElement("p");
var nodo = document.createTextNode("Parrafo nuevo");
parrafo.appendChild(nodo);

var elemento = document.getElementById("div1");
var hijo = document.getElementById("p1");
elemento.insertBefore(parrafo, hijo);

</script>

</body>
</html>
```

Parrafo nuevo

Esto es un parrafo

Esto es otro parrafo

- En este caso, *insertBefore()*, se inserta el nodo *parrafo* antes del nodo "*p1*".

```
<!DOCTYPE html>
<html>
<body>

<div id="div1">
<p id="p1">Esto es un parrafo</p>
<p id="p2">Esto es otro parrafo</p>
</div>

<button onclick="miFuncion()">Eliminar segundo parrafo</button>

<script>

function miFuncion(){
      document.getElementById("p2").remove();
}

</script>

</body>
</html>
```

- Mediante el método *remove()*, eliminamos el nodo seleccionado.

```
<!DOCTYPE html>
<html>
<body>

<div id="div1">
<p id="p1">Esto es un parrafo</p>
<p id="p2">Esto es otro parrafo</p>
</div>

<script>
var padre = document.getElementById("div1");
var hijo = document.getElementById("p1");

var parrafo = document.createElement("p");
var nodo = document.createTextNode("Parrafo nuevo");
parrafo.appendChild(nodo);

padre.replaceChild(parrafo,hijo);
</script>

</body>
</html>
```

Parrafo nuevo

Esto es otro parrafo

- *replaceChild()*, permite reemplaza un elemento hijo por otro, en este caso, un párrafo por otro.

Colecciones

- Con el método *getElementsByTagName()* podemos obtener una lista de todos los objetos de un tipo que haya en el documento.
- Hay que tener en cuenta que este método devuelve una lista, no un array por lo que no se podrán usar los métodos tipos de array como *valueOf()*, *push()*, *pop()*, etc.

```
<!DOCTYPE html>
<html>
<body>

<p>Parrafo 1</p>
```

Parrafo 1

Parrafo 2

Haz click en el boton para cambiar color

Cambiar color

```
<p>Parrafo 2</p>

<p>Haz click en el boton para cambiar color</p>

<button onclick="cambiarColor()">Cambiar</button>

<script>
function cambiarColor() {
  var coleccion = document.getElementsByTagName("p");

  for (i = 0; i < coleccion.length; i++) {
    coleccion[i].style.color = "red";
  }
}
</script>

</body>
</html>
```

- En la variable *coleccion* se genera una lista con todos los elementos *<p>* del documento, la cual se itera posteriormente, accediendo a la propiedad color de cada elemento de la lista y cambiándolo a rojo.
- Podemos cambiar atributos accediendo a ellos a través del operador *'.'* o con los métodos :
  - *hasAttribute():* devuelve true o false
  - *getAttribute():* devuelve el valor del atributo
  - *setAttribute():* actualiza o añade un atributo
  - *removeAttribute():* elimina un atributo

elemento.hasAttribute("src");

elemento.getAttribute("src");

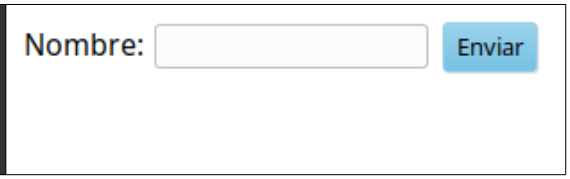
elemento.setAttribute("src");

elemento.removeAttribute("src");

Formularios

- Podemos validar información de formularios con Javascript.

```
<!DOCTYPE html>
<html>
<head>
<script>
function validar() {
```



```
  var x = document.forms["myForm"]
["nombre"].value;
  if (x == "") {
    alert("Cuadro de texto vacio");
    return false;
  }
}
</script>
</head>
<body>

<form name="myForm"onsubmit="return
validar()">
  Nombre: <input type="text"
name="nombre_txt">
  <input type="submit" value="Enviar">
</form>

</body>
</html>
```

- Este ejemplo podría realizarse mediante validación HTML sin más que poner "*required*" al cuadro de texto.

- *checkValidity()*: comprueba la validez de los datos y permite tomar una decisión al respecto.
- *rangeOverflow()* y *rangeUnderflow()*: si los valores se pasan por arriba o por abajo permite tomar una decisión al respecto.

```
<!DOCTYPE html>
<html>
<body>

<p>Introduce un numro entre 100 y 300</p>

<input id="numero_txt" type="number" min="100"
max="300" required>
<button onclick="comprobar()">OK</button>


<p id="p01"></p>

<script>
function comprobar() {
  var numero =
document.getElementById("numero_txt");
  if (!numero.checkValidity()) {
    document.getElementById("p01").innerHTML =
"Entrada incorrecta";
  } else {
    document.getElementById("p01").innerHTML =
"Entrada correcta";
```

Introduce un numro entre 100 y 300

101 − + OK

Entrada correcta

```
  }
}
</script>

</body>
</html>
```

Javascript BOM

- El Browser Object Model nos permite comunicarnos con los navegadores.
- Hay que tener en cuenta que no es una interfaz estándar por lo que ciertos métodos pueden no funcionar correctamente en algunos navegadores.
- Algunos objetos interesantes del BOM son:

  - *window*: representa la ventana del navegador.
  - *screen*: información de la pantalla del usuario.
  - *location*: información sobre la URL actual y redirección.
  - *history*: permite navegar a la página anterior y siguiente, con respecto a las páginas visitadas.
  - *navigator*: información sobre el navegador del usuario. La información que se obtienen sobre el tipo de navegador no es fiable.
  - *window.alert*: diferentes modelos de ventanas pop-up.

```
<!DOCTYPE html>
<html>
<body>

<button onclick="loc()">Google</button>
<button
onclick="confirmacion()">Confirmar...</button>
<button onclick="pedirInfo()">Pedir
info...</button>

<script>
var w = window.innerWidth
var h = window.innerHeight


document.write("<br />Ventana interna
navegador: " + w + " x " + h);
```

```
var res = screen.width + " x " +
screen.height;
document.write("<br />Res: " + res);

var so = navigator.platform;
document.write("<br />Sistema operativo del
usuario: " + so)

function loc(){

window.location.assign("http://www.google.es")
;
}

function confirmacion() {
        var txt;
        if (confirm("Presiona un boton")) {
          txt = "Has aceptado";
        } else {
          txt = "Has cancelado";
        }
        document.write("<br />" + txt);
      }

function pedirInfo() {
        var txt;
        var persona = prompt("Nombre?:",
"Marcos");
        if (persona == null || persona == "")
{
          txt = "El usuario ha cancelado ";
        } else {
          txt = "Hola " + persona + " que
tal?";
        }
        document.write("<br />" + txt);
      }


</script>

</body>
</html>
```

POO en Javascript

- Javascript permite programar mediante orientación a objetos.

```
<!DOCTYPE html>
<html>
<body>

<script>
```

```
Pepe
Perez
123
Pepe Perez
Pepe Perez 123 function ()
{ return this.nombre + " " +
```

```
var persona = {
  nombre   : "Pepe",
  apellido : "Perez",
  id       : 123,
  nombreCompleto : function() {
    return this.nombre + " " + this.apellido;
  }
};

document.write(persona.nombre);
document.write("<br />" + persona["apellido"]);

document.write("<br />" + persona["id"]);

document.write("<br />" + persona.nombreCompleto());

document.write("<br />");
for (x in persona){
      document.write(persona[x] + " ");
}
</script>

</body>
</html>
```

```
this.apellido; }
```

- Se muestran en el ejemplo varias formas de visualizar los elementos de un objeto. La palabra clave *this* representa al propio objeto.

- Los elementos que forman un objeto se pueden pueden variar, ya sea añadiendo o quitando elementos. Se añaden simplemente dándoles un valor y se quitan mediante la palabra reservada *delete*.

```
<!DOCTYPE html>
<html>
<body>

<script>

var persona = {
  nombre   : "Pepe",
  apellido : "Perez",
  id       : 123,
  nombreCompleto : function() {
    return this.nombre + " " +
this.apellido;
  }
};

persona.nacionalidad = "espanola";
persona.sueldo = function(){
      return 3.5 * 500;
```

```
Pepe Perez espanola function (){ return 3.5 *
500; }
```

```
};
delete persona.id;
delete persona.nombreCompleto;


for (x in persona){
      document.write(persona[x] + " ");
}

</script>

</body>
</html>
```

- Los elementos de un objeto se pueden convertir en un array mediante el método *`Object.values()`*.

```
<!DOCTYPE html>
<html>
<body>

<script>

var persona = {
  nombre   : "Pepe",
  apellido : "Perez",
  id       : 123
};

var vector = Object.values(persona);

for (x in vector){
      document.write(vector[x] + " ");
}

</script>

</body>
</html>
```

Pepe Perez 123

- Podemos crear métodos *`getter`* y *`setter`* para acceder a los elementos del objeto.

```
<!DOCTYPE html>
<html>
<body>

<script>

var persona = {
```

123

```
  nombre: "Pepe",
  apellido: "Perez",
  id: "",
  set identificador(id){
        this.id = id;
  },
  get identificador(){
        return this.id;
  }
};

persona.identificador = 123;
document.write(persona.identificador);

</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<p id="p01"></p>

<script>

var obj = {counter : 0};


Object.defineProperty(obj, "reset", {
  get : function () {this.counter = 0;}
});
Object.defineProperty(obj, "incremetar", {
  get : function () {this.counter++;}
});
Object.defineProperty(obj, "decrementar", {
  get : function () {this.counter--;}
});
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
});
Object.defineProperty(obj, "restar", {
  set : function (value) {this.counter -= value;}
});


obj.reset;
obj.add = 5;
obj.restar = 1;
obj.incremetar;
obj.decrementar;
document.getElementById("p01").innerHTML = obj.counter;
</script>

</body>
</html>
```

- Se usará una función para crear los constructores de los objetos

```
<!DOCTYPE html>
<html>
<body>

<script>


function Persona(nombre, apellido, edad){
      this.nombre = nombre;
      this.apellido = apellido;
      this.edad = edad;
      this.sueldo = 0;
      this.setSueldo = function (factor){
            this.sueldo = 800 * factor;
      };
      this.datos = function() {
            return this.nombre + " " +
            this.apellido + " " +
            this.edad + " " +
            this.sueldo;
      };
}

var empleado = new Persona("Pepe", "Perez", 25);
empleado.setSueldo(3);

document.write(empleado.datos());

</script>

</body>
</html>
```

Pepe Perez 25 2400