



JavaScript



ANEXO 2.-
Orientacion a Objetos



DISTRIBUIDO POR:

CENTRO DE INFORMÁTICA PROFESIONAL S.L.

C/ URGELL, 100
08011 BARCELONA
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10
48014 BILBAO
TFNO: 94 448 31 33

www.cipsa.net

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE
REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO
CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

Objetos

Un objeto es una agrupación de propiedades y métodos que representan una información compleja, como los datos de una persona, una cuenta bancaria..., etc.

- Las propiedades representan características del elemento representado.
Sea un objeto cuenta bancaria → *nº cuenta, titular, saldo, interés...*
- Los métodos representan las operaciones que admite el elemento.
Sea un objeto cuenta bancaria → *ingresar(), retirar(), calcularBeneficios()...*

Declaración

Un objeto se declara como un conjunto de variables cada una de las cuales puede llevar asignado un valor si es una propiedad, o una función si es un método:

```
var <objeto> = {
    <atributo1> : <valor>,
    <metodo1> : function() {
        // código del método
    }
}
```

Ejemplo: El siguiente objeto define una cuenta bancaria:

```
var objCuenta = {
    // Propiedades
    ncuenta: "4345-4526-67-4354359043",
    titular: "Roger Dawson",
    saldo: 1210.45,
    interes: 0.02,
    // Métodos
    ingresar: function ( cantidad ) {
        this.saldo += cantidad;    // Añade al saldo la cantidad
    },
    retirar: function (cantidad) {
        this.saldo -= cantidad;    // Retira la cantidad del saldo
    }
}
```

Manejo

Para acceder a una propiedad de un objeto se emplea la siguiente sintaxis:

<objeto>.<propiedad> = <valor>

Para invocar a un método de un objeto se emplea la misma sintaxis que para llamar a una función pero incluyendo el objeto delante del identificador de la función:

<objeto>.<identificador_funcion>(<argumento1>, <argumento2>...)

Ejemplo:

```
// Obtiene valor propiedad saldo del objeto
console.log(objCuenta.saldo);
// Modifica el valor propiedad interes del objeto
objCuenta.interes = 0.05;
// Llama al método ingresar()
objCuenta.ingresar(100.50);
```

Declaración función constructora

Se trata de un tipo especial de funciones que crean objetos a partir del valor de los parámetros que reciben. Se emplea cuando quieren crearse múltiples objetos con las mismas propiedades y métodos sin tener que declararlos íntegramente uno por uno:

```
function <tipo_objeto>( <param1>, <param2>, ... ) {
    this.<atributo1> = <param1>;
    this.<metodo1> = function( <param1>, <param2>, ... ) {
        // código del método
    }
}
```

Ejemplo: El siguiente código define un tipo de objeto llamado *CuentaBancaria* que representa los datos y operaciones de una cuenta bancaria:

```
function CuentaBancaria( _titular, _saldo, _interes ) {
    // Propiedades
    this.titular = _titular;
    this.saldo = _saldo;
    this.interes = _interes;
    // Métodos.
    this.ingresar = function( cuantia ) {
        this.saldo = this.saldo + cuantia;
    };
    this.retirar = function( _cuantia ) {
        this.saldo = this.saldo - cuantia;
    };
    this.calcularRendimiento = function() {
        return this.saldo * this.interes;
    };
}
```

Ejemplo:

```
// Creacion del objeto 'cuenta'.
var cuenta = new CuentaBancaria("Roger", 1200.0, 0.45);
// Obtencion valores de propiedades del objeto.
console.log("Nombre del titular: " + cuenta.titular);
// Modificacion valores de propiedades
cuenta.interes = 0.5;
// Ingreso de 100.
cuenta.ingresar(100);
// Retirada de 50
cuenta.retirar(50);
var rendimiento = cuenta.calcularRendimiento();
console.log("El rendimiento anual de la cuenta es de: " + rendimiento);
```

Vinculacion de propiedades y métodos a objetos

Vincular una propiedad o método a un objeto consiste en añadirle éstos a un objeto como si formasen parte de su declaración. Para ello basta añadir un identificador y asignarle la variable o función deseada:

<objeto>.<propiedad> = <variable_a_vincular>

<objeto>.<método> = <funcion_a_vincular>

Una vez vinculada al objeto, la propiedad o método pueden invocarse como cualquier otra propiedad o método propios.

```
// Declaración del objeto
var obj = {
  a: 10,
  b: 20,
  c: 30,
}
// Declaracion de function ajena al objeto
var fn_suma = function () {
  return this.a + this.b + this.c;
}

// vinculacion de la funcion suma como metodo de obj
obj.suma = fn_suma;
// Invocación de la funcion vinculada suma()
console.log("SUMATORIO: " + obj.suma()); // Muestra 60
```

MEDIA: 20

[HtmlPage1.html:22](#)

SUMATORIO: 60

[HtmlPage1.html:26](#)

Funciones invocadas con *function.call()*

Una función puede invocarse para que se ejecute como un método de un objeto empleando la función **Call()** e indicando como argumento el propio objeto:

```
var obj1 = {
  a: 10,
  b: 20
}
var obj2 = {
  a: 5,
  b: 10
}
// Función ajena producto()
function producto() {
  return this.a * this.b;
}
// Invocacion de funcion producto() contra objeto obj1
console.log(producto.call(obj1)); // Muestra
// Invocacion de funcion producto() contra objeto obj2
console.log(producto.call(obj2)); // Muestra
```

200

[HtmlPage1.html:23](#)

50

[HtmlPage1.html:25](#)

La referencia *this*

Esta es una palabra clave del lenguaje que hace referencia automática a un objeto y por la cual pueden invocarse sus métodos y accederse a sus propiedades:

- En el caso de un método de un objeto, **this** hace referencia al objeto en el que se ejecute dando acceso a sus métodos y propiedades:

```
var obj1 = {
  a: 10,
  b: 20,
  producto: function () {
    return this.a * this.b;
  }
}
var obj2 = {
  a: 5,
  b: 10,
  producto: function () {
    return this.a * this.b;
  }
}
console.log(obj1.producto()); // Ejecuta producto() en obj1.
console.log(obj2.producto()); // Ejecuta producto() en obj2.
```

200

[HtmlPage1.html:25](#)

50

[HtmlPage1.html:26](#)

Al ejecutarse el método *producto()*, la variable *this* hace referencia a las propiedades de *obj1* en la primera llamada (a=10, b=20), y a las de *obj2* en la siguiente (a=5 y vb=4)

- En el caso de una función vinculada a un objeto, **this** hace referencia al objeto vinculado.
- En el caso de una función llamada mediante *function.call()*, **this** hace referencia al objeto dado como argumento en *call()*.
- En el caso de una función no vinculada a ningún método, **this** hace referencia a las funciones y variables globales.

```
var a = 10;
var b = 20;

var producto = function () {
  return this.a * this.b;
}

console.log(producto()); // Muestra 200.
```

Ejercicio resuelto:

Crea una página web que define un tipo de objeto *Cuadrado*, solicite la base y la altura de tres cuadrados y a continuación muestre en una tabla sus áreas.

Solución:

Lo primero es definir el tipo de objeto *Cuadrado*:

```
function Cuadrado() {  
    this.base = 0;  
    this.altura = 0;  
  
    this.calcularArea = function() {  
        return this.base * this.altura;  
    };  
}
```

Una vez definido el tipo de objeto, ya pueden crearse objetos cuadrados indicando a cada uno su correspondiente base y altura mediante sus propiedades:

```
var cuadrado1 = new Cuadrado();  
cuadrado1.base = Number(prompt("Introduce base cuadrado 1", ""));  
cuadrado1.altura = Number(prompt("Introduce la altura de cuadrado 1", ""));  
  
var cuadrado2 = new Cuadrado();  
cuadrado2.base = Number(prompt("Introduce base cuadrado 2", ""));  
cuadrado2.altura = Number(prompt("Introduce la altura de cuadrado 2", ""));  
  
var cuadrado3 = new Cuadrado();  
cuadrado3.base = Number(prompt("Introduce base cuadrado 3", ""));  
cuadrado3.altura = Number(prompt("Introduce la altura de cuadrado 3", ""));
```

Para mostrar al área de cada cuadrado sólo es necesario llamar al método *calcularArea()* de cada uno de ellos:

```
document.write("<table>");  
document.write("<tr><td>Area del cuadrado 1: " + cuadrado1.calcularArea() +  
"</td></tr>");  
document.write("<tr><td>Area del cuadrado 2: " + cuadrado2.calcularArea() +  
"</td></tr>");  
document.write("<tr><td>Area del cuadrado 3: " + cuadrado3.calcularArea() +  
"</td></tr>");  
document.write("</table>");
```

Ejercicios propuestos:

1. Sea un tipo de objeto *Usuario* provisto de dos propiedades: *usuario* y *clave* que define además un método *comprobarClave()* que recibe dos parámetros (*usuario* y *clave*) y devuelve un valor cierto si ambos coinciden con el nombre del usuario y su contraseña:

```
function Usuario( _usuario, _clave ) {
  this.usuario = _usuario;
  this.clave = _clave;
  this.comprobarUsuario = function( _usu, _cla) {
    if ( this.usuario == _usu && this.clave == _cla) {
      return true;
    } else {
      return false;
    }
  };
}
```

Crea una página que cree a su vez un objeto *Usuario* con nombre “*roger*” y clave: *1234*. La página debe solicitar entonces un nombre de usuario y una contraseña y comprobar si son correctos empleando el método *comprobarUsuario()* del objeto usuario. Si el usuario se identifica correctamente debe mostrársele un mensaje de bienvenida. En caso contrario debe mostrársele un mensaje de error y volvérselo a solicitar el usuario y la contraseña.

2. Crea un tipo de objeto *ApuestaPrimitiva* provisto de seis propiedades que representan los valores de una primitiva: *v1*, *v2*, *v3*, *v4*, *v5*, y *v6*. Define además un método *contarAciertos()* que recibe como parámetros seis valores de la combinación ganadora y devuelve el número de aciertos.

3. Crea un tipo de objeto *Triángulo* provisto de tres propiedades: *lado1*, *lado2* y *lado3*. Define además tres métodos:

- a. *esEquilatero()* → devuelve un valor cierto si todos los lados son iguales.
- b. *esIsosceles()* → devuelve un valor cierto si dos de los lados son iguales.
- c. *esEscaleno()* → devuelve un valor cierto si todos los lados son distintos.

Crea una página que solicite los tres lados de un triángulo, cree un objeto triángulo, y después indique si es equilátero, isósceles o escaleno llamando a sus métodos.

Objetos de Javascript

Vectores

Una variable normal contiene un único valor. Un vector es un tipo de objeto definido por Javascript que almacena múltiples valores identificados por un *índice* o *clave*:

0	1	2	3	4	5	6	7	8	9

Representación de un vector de 10 elementos.

Un vector se declara como un objeto del tipo *Array*:

```
var v = new Array();           // Sin valores.
```

```
var v = ["a", "b", "c"];      // Con valores.
```

Para acceder a un valor del vector se emplea el nombre del vector y su índice:

```
<identificador_vector>[ <índice> ]
```

Ejemplo:

El siguiente código define un vector en el que se almacenan cuatro valores de diferentes tipos:

```
// Declaracion del vector
var v = new Array();

// Asignacion de valores
v[0] = "Roger";
v[1] = 120;
v[2] = 0.5;
v[3] = false;

// Obtencion de valores
document.write( v[0] + "<br>");
document.write( v[1] + "<br>");
document.write( v[2] + "<br>");
document.write( v[3] + "<br>");
```

Longitud de un vector

La longitud de un vector puede obtenerse invocando el método **length**.

```
var vector = ['hola', 'mundo'];
console.log(vector.length); // muestra en la consola 2
```

Recorrido de vectores

Recorrer una vector consiste en obtener uno por uno todos los valores que contiene. Esta operación puede implementarse de múltiples maneras:

Recorrido con bucle **for**:

```
var valores = ["Lunes",  
              "Martes",  
              "Miercoles",  
              "Jueves",  
              "Viernes",  
              "Sabado",  
              "Domingo"];
```

El siguiente código muestra por pantalla todos los valores contenidos en el vector empleando una variable `indx` que recorre todos sus índices del 0 al último = longitud-1.

```
for( var indx = 0; indx < valores.length; indx++) {  
    document.write("En posicion "+indx+"->" +valores[indx]+"<br>");  
}
```

El resultado mostrado es:

```
En posicion 0 -> Lunes  
En posicion 1 -> Martes  
En posicion 2 -> Miercoles  
En posicion 3 -> Jueves  
En posicion 4 -> Viernes  
En posicion 5 -> Sabado  
En posicion 6 -> Domingo
```

Recorrido con **for (in)**

El siguiente código emplea un bucle **for** especial que se ejecuta tantas veces como valores haya en el vector devolviendo cada índice en la variable *indice*.

```
var indice;  
for(indice in valores) {  
    document.write("valor es:"+ indice +"=>"+valores[indice]+"<br >");  
}
```

El resultado mostrado es:

```
valor es: 0 => Lunes  
valor es: 1 => Martes  
valor es: 2 => Miercoles  
valor es: 3 => Jueves  
valor es: 4 => Viernes  
valor es: 5 => Sabado  
valor es: 6 => Domingo
```

Vectores asociativos

Un vector asociativo es aquel en el que los valores se almacenan asociados a un clave como una cadena en vez de a un índice numérico:

```
var extensiones = new Array();
extensiones["españa"] = "es";
extensiones["inglaterra"] = "en";
extensiones["usa"] = "us";
extensiones["rusia"] = "ru";
```

Para acceder a cada elemento se emplea el valor utilizado como *clave*.

```
document.write( extensiones["españa"] + "<br>");
document.write( extensiones["inglaterra"] + "<br>");
document.write( extensiones["usa"] + "<br>");
document.write( extensiones["rusia"] + "<br>");
```

Las matrices asociativas son equivalentes a los objetos en los que cada valor se correspondería con una propiedad:

```
var extensiones = { españa:"es", inglaterra:"en", usa:"us", rusia:"ru"};
```

Para acceder a los valores puede emplearse la sintaxis propia de un objeto, o la de un vector:

```
document.write(extensiones.españa); // Devuelve valor de la propiedad españa
document.write(extensiones["españa"]); // Devuelve valor con la clave "españa"
```

Recorrido de vectores asociativos

Las matrices asociativas pueden recorrerse empleando una estructura **for..in**. El bucle siguiente muestra todas las propiedades y sus valores del objeto *extensiones*.

```
var clave;
for(clave in extensiones) {
    document.write("valor es:"+clave+"=>" + extensiones[clave] + "<br>");
}
```

El resultado mostrado por pantalla es:

```
valor es:españa=>es
valor es:inglaterra=>en
valor es:usa=>us
valor es:rusia=>ru
```

Vectores de objetos

Una vector también puede almacenar objetos.

En el ejemplo es emplear un vector para almacenar una serie de objetos, cada uno de los cuales contiene a su vez los datos de una persona (nombre, apellido, edad).

```
// Declaracion del tipo de objeto Persona
function Persona( _nombre, _apellido, _edad ) {
    this.nombre = _nombre;
    this.apellido = _apellido;
    this.edad = _edad;
}
```

Para registrar un objeto en el vector se le asigna indicando un índice.

```
// Declaracion y relleno de matriz de objetos Persona.
var v = new Array();
v[0] = new Persona("Roger", "Petrov", 23);
v[1] = new Persona("Yuri", "Estebanov", 29);
v[2] = new Persona("Sasha", "kurchitev", 26);
```

Para recuperar cada objeto se puede emplear una variable 'obj' a la que se asigna el objeto *Persona* en cada posición del vector.

```
var i;
var obj;
document.write("<table>");
// Bucle de recorrido del vector.
for (i = 0; i < v.length; i++) {
    // Obtencion del objeto Persona en la posicion 'i' del vector
    obj = v[i];
    // Visualizacion de sus propiedades .
    document.write("<tr>");
    document.write("<td>" + obj.nombre + "</td>");
    document.write("<td>" + obj.apellido + "</td>");
    document.write("<td>" + obj.edad + "</td>");
    document.write("</tr>");
}
document.write("</table>");
```

El resultado mostrado es:

NOMBRE	APELLIDO	EDAD
Roger	Petrov	23
Yuri	Estebanov	29
Sasha	kurchitev	26

Métodos de manipulación de vectores

Los vectores son objetos y como tales disponen de una serie de propiedades y métodos que permiten su manipulación:

Métodos	Descripción
<u>concat()</u>	Une dos matrices devolviendo una tercera con los valores de ambas.
<u>indexOf()</u>	Devuelve la posición de un valor en un vector. Si el valor no existe retorna -1.
<u>join()</u>	Devuelve una cadena con todos los valores de un vector separados por comas.
<u>lastIndexOf()</u>	Devuelve la posición de la última aparición de un valor en un vector. Si el valor no existe retorna -1.
<u>pop()</u>	Devuelve el elemento en la última posición de un vector eliminándolo.
<u>push()</u>	Añade un elemento al final de la matriz devolviendo la longitud final del vector
<u>reverse()</u>	Invierte el orden de los valores de un vector
<u>shift()</u>	Devuelve el valor situado en la primera posición de un vector eliminándolo.
<u>slice()</u>	Devuelve una sección de un vector.
<u>sort()</u>	Ordena los valores de un vector.
<u>splice()</u>	Añade y elimina valores de un vector a partir de una posición dada.
<u>toString()</u>	Devuelve una cadena de caracteres representando un vector.
<u>unshift()</u>	Añade un valor al principio de un vector devolviendo la nueva longitud.

Ejemplo: *reverse()* / *join()* → Estos métodos invierten el orden de los valores del vector y devuelve una cadena con todos los valores separados por el carácter indicado:

```
var v = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
// Método que devuelve la matriz invertida
var alreves = v.reverse();
// Método que devuelve una cadena con los valores del vector separados por ,
document.write(alreves.join(",")); // Muestra 9,8,7,6,5,4,3,2,1,0
```

Ejemplo: *indexOf()* → Método que devuelve la posición en el vector del valor dado. En caso de no encontrarse el valor retorna -1. Se solicita al usuario el nombre de una fruta y si está presente entre los valores del vector frutas se muestra su posición. Si la fruta no está se muestra el mensaje "La fruta no existe".

```
var frutas = ["naranja", "limon", "manzana", "platano"];
var nombre = prompt("Indica el nombre de una fruta", "");
// Método que devuelve la posición del valor indicado en el vector
var posicion = frutas.indexOf(nombre);
if ( posicion >= 0 ) {
    document.write("La fruta esta en la posición: " + posicion );
} else {
    document.write("La fruta no existe");
}
```

Ejemplo: *slice()* / *join()* → El método *slice()* retorna un vector con el número de valores indicados a partir de la posición dada de otro vector.

```
var m = new Array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
// Devuelve un vector con la cantidad de valores indicados a partir del índice dado.
var sub = m.slice(2, 6);
document.write(sub.join("-"));           // Muestra 2-3-4-5
```

Ejemplo: *push()* / *pop()* → Estos métodos insertan y extraen valores de un vector por el final.

```
// Declaracion de la clase Persona
var datos = new Array();
datos.push("a");           // Inserta valores al final
datos.push("b");
datos.push("c");
alert(datos.length);       // Longitud = 3 elementos.
alert(datos.pop());        // Extrae 'c'
alert(datos.pop());        // Extrae 'b'
alert(datos.pop());        // Extrae 'a'
alert(datos.length);       // Longitud = 0 elementos.
```

Ejemplo: *unshift()* / *shift()* → Estos métodos insertan y extraen valores de un vector por el principio.

```
// Declaracion de la clase Persona
var datos = new Array();
datos.unshift("a");        // Inserta valores al principio
datos.unshift("b");
datos.unshift("c");
alert(datos.length);       // Longitud = 3 elementos.
alert(datos.shift());       // Extrae 'c'
alert(datos.shift());       // Extrae 'b'
alert(datos.shift());       // Extrae 'a'
alert(datos.length);       // Longitud = 0 elementos.
```

(*) Los métodos *unshift()/shift()* y *push()/pop()* se emplean tanto para insertar como para extraer valores de un vector. Los métodos *unshift()/shift()* insertan y extraen valores al principio del vector, mientras que *push()/pop()* lo hacen al final.

Si se emplean correctamente puede manejarse el vector como una *pila* o una *cola*.

Una pila es un vector en el que se añaden valores para extraerlos en orden inverso al que se han insertado. La inserción de valores se hacen con *push()* y la extracción con *pop()*.

Una cola es un vector en el que se añaden valores para extraerlos en el mismo orden en el que se han insertado. La inserción de valores se hace con *unshift()* y la extracción con *pop()*.

Ejercicio Resuelto:

Crea una página que solicite al usuario un número de mes y muestre el nombre del mes correspondiente.

Solución:

El nombre del mes se obtiene empleando su número como índice pero restando 1, ya que los nombres de los meses se almacenan en el vector empezando por el índice 0.

```
// Vector de meses
var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo",
             "Junio", "Julio", "Agosto", "Septiembre",
             "Octubre", "Noviembre", "Diciembre"];

var num_mes = Number(prompt("Indica el numero de mes", ""));
if ( num_mes >= 1 && num_mes <= 12) {
    document.write("El mes indicado es: " + meses[num_mes-1]);
} else {
    document.write("Mes no válido");
}
```

Ejercicio Resuelto:

Crea una página que solicite el nombre de un color y muestre a continuación el código HTML correspondiente.

Solución.

Se empieza por crear dos matrices, una con los nombres de los colores (*nombres*) y otra con sus correspondientes códigos HTML (*colores*) *manteniendo el mismo orden*.

```
// Declaracion de nombres de colores
var nombres = new ["Negro",
                  "Azul",
                  "Rojo",
                  "Magenta",
                  "Verde",
                  "Cyan",
                  "Amarillo",
                  "Blanco"];

// Declaracion de codigos de colores
var colores = ["&H000000",
              "&H0000FF",
              "&HFF0000",
              "&HFF00FF",
              "&H00FF00",
              "&H00FFFF",
              "&HFFFF00",
              "&HFFFFFF"];
```

Cuando el usuario indica el nombre del color deseado, se localiza su posición en el vector de nombres empleando el método **indexOf()**. El método recibe como argumento el valor a buscar y se invoca contra el vector en el que buscarlo.

```
// Solicitud nombre de color
var nombre = prompt("Indica nombre del color", "");
// Obtencion del índice del color indicado
var indice = nombres.indexOf( nombre );
```

Si el valor devuelto es -1 significa que el valor buscado no está presente en el vector. En caso contrario retorna la posición de la primera aparición del valor.

```
if ( indice != -1 ) {
    alert("El código HTML del color es " + colores[indice]);
} else {
    alert("Color desconocido.");
}
```

Ejercicios Propuestos:

1. Crea una página que solicite el nombre de los socios de una peña. Para ello, la página solicitará cada nombre y a continuación mostrará un cuadro de diálogo preguntando si se desea continuar. Una vez que el usuario haya finalizado de introducir todos los nombres la página debe mostrar los siguientes datos:
 - a. Una tabla con los nombres de los socios en el orden en que se han introducido.
 - b. Una tabla con los nombres de los socios ordenados alfabéticamente.
 - c. El total de socios introducidos.
2. Crea una página que solicite al usuario el número de alumnos presentes en un aula. A continuación la página debe solicitar el nombre y calificación final de cada alumno.

Una vez introducidos los nombres y notas de todos los alumnos la página debe mostrar el nombre del alumno con mejor calificación y las media de las notas.

3. Crea una página para registrar las precipitaciones registradas por meses. Para ello la página debe solicitar el nombre de un mes y unas precipitaciones en litros. A continuación la página debe mostrar un cuadro de diálogo preguntando si el usuario desea registrar más precipitaciones. Las precipitaciones asociadas a los mismos meses deben acumularse.

Una vez que el usuario termine la página debe mostrar una tabla con todos los meses y el acumulado de precipitaciones correspondiente a cada uno de ellos.

Cadenas de caracteres

Las cadenas de caracteres son realmente objetos de tipo **String**.

```
var nombre = "pepe";
var nombre = new String("pepe");
```

El tipo **String** define las siguientes propiedades y métodos que pueden utilizarse con cualquier cadena:

Propiedades	Descripción
<u>length</u>	Devuelve la longitud de una cadena en caracteres.

Ejemplo: El siguiente código declara una cadena y muestra su número de caracteres.

```
var nombre = "pepe";
document.write(nombre.length);
```



Métodos	Descripción
<u>charAt()</u>	Devuelve el carácter en la posición indicada.
<u>concat()</u>	Devuelve una cadena resultante de la concatenación de otras dos.
<u>indexOf()</u>	Retorna la posición de la primera aparición de una cadena en otra. Si no aparece retorna -1.
<u>lastIndexOf()</u>	Retorna la posición de la última aparición de una cadena en otra. Si no aparece retorna -1.
<u>replace()</u>	Reemplaza en una cadena las apariciones de una subcadena por otra.
<u>search()</u>	Retorna la posición en la que una cadena verifica una <i>expresión regular</i> .
<u>split()</u>	Parte una cadena en partes en función de un carácter separados y devuelve las partes en un vector.
<u>substr()</u>	Devuelve una parte de una cadena indicando la posición del primer carácter y la longitud.
<u>substring()</u>	Devuelve una parte de una cadena indicando la posición del primer y del último carácter.
<u>toLowerCase()</u>	Convierte una cadena a minúsculas
<u>toUpperCase()</u>	Convierte una cadena a mayúsculas
<u>trim()</u>	Elimina los espacios en blanco presentes al principio y final de una cadena.

Ejemplos:

- **charAt(posicion)** → Este método retorna el carácter presente en la posición indicada de la cadena. El primer carácter se considera que es el 0.

```
var cadena = "Hola mundo";
// Devuelve la letra en el índice 4 de la cadena
var letra = cadena.charAt(3);
alert(letra);           // Muestra 'a'
```

- ***indexOf(cadena, pos_inicial)***: Este método retorna la posición del primer carácter en el que aparece la cadena pasada como parámetro. Si la cadena se repite, sólo se retorna la posición correspondiente a la primera aparición. Si la cadena indicada no está presente, retorna el valor -1. El parámetro pos_inicial indica la posición del primer carácter a partir del que se busca la cadena indicada. Si se omite se busca desde el principio de la cadena.

```
var cadena = "Hola mundo";
var pos = cadena.indexOf("mundo", 0);
alert(pos); // Muestra 5
var pos1 = cadena.indexOf("pepe", 0);
alert(pos1); // Muestra -1
```

- ***lastIndexOf(cadena)***: Este método es equivalente al anterior pero con la excepción de que retorna la posición de la última aparición de la cadena. Si la cadena pasada como parámetro no está presente, retorna el valor -1.

```
var cadena = "Bien Hola mundo Hola";
var pos_primera = cadena.indexOf("Hola");
var pos_final = cadena.lastIndexOf("Hola");
alert(pos_primera); // Muestra 5
alert(pos_final); // Muestra 16
```

- ***substring(pos_inicio, pos_final)***: Devuelve una nueva cadena con los caracteres comprendidos entre la posición inicial y la final excluida:

```
var str="Hello world!";
document.write(str.substring(3)); // "lo world!"
document.write(str.substring(3,7)); // "lo w"
```

- ***slice(pos_inicio, pos_final)***: Devuelve una nueva cadena con los caracteres incluidos entre pos_inicio y pos_final. Si no se indica pos_final, se devuelven todos los caracteres restantes. Si se indica pos_inicio menor que 0, se retornan los caracteres con respecto al final de la cadena.

```
var str="Hello happy world!";
document.write(str.slice(0)); // "Hello happy world!"
document.write(str.slice(6)); // "happy world!"
document.write(str.slice(-6)); // "world!"
document.write(str.slice(0,1)); // "H"
document.write(str.slice(6,11)); // "happy"
```

- ***substr(pos_inicio, longitud)***: Devuelve una cadena con la longitud indicada en el parámetro longitud, recortada a partir del carácter pos_inicio.

```
var str="Hello world!";
document.write(str.substr(3)); // "lo world!"
document.write(str.substr(3,4)); // "lo w"
```

- ***split(caracter)***: Este método permite segmentar una cadena de texto en subcadenas en función de un carácter pasado como parámetro. Las subcadenas son retornadas en un vector:

```
var cadena = "19029,230943,34093,4930059,3049320";
var partes = new Array();
partes = cadena.split(",");
// Bucle que recorre el vector de cadenas 'partes'
for (var i = 0; i < partes.length; i++) {
    document.write(partes[i] + "<br>");
}
```

El código muestra:

```
19029
230943
34093
4930059
3049320
```

- ***toUpperCase()/toLowerCase()***: Estos métodos retornan la cadena pasada a mayúsculas (*toUpperCase*), o a minúsculas (*toLowerCase*).

```
var cadena = "aBcDeF";
var mayusculas = cadena.toUpperCase();
var minusculas = cadena.toLowerCase();
alert(mayusculas);           // Muestra 'ABCDEF'
alert(minusculas);          // Muestra 'abcdef'
```

- ***concat(cadena)***: Este método retorna una cadena de texto resultado de la concatenación de la cadena sobre la que se invoca el método y la dada como parámetro.

```
var nombre = "roger";
var apellido = " petrov";
var nombreCompleto = nombre.concat(apellido);
alert(nombreCompleto);      // Muestra 'roger petrov'
```

- ***replace(expresión, sustituto)***: Este método sustituye en una cadena aquellas partes que cumplan la expresión regular indicada en el valor '*expresión*' y las sustituye por la cadena del valor '*sustituto*'.

```
var str="Welcome to Microsoft! ";
str=str + "We are proud to announce that Microsoft has ";
str=str + "one of the largest Web Developers sites in the world.";
document.write(str.replace(/microsoft/gi, "W3Schools"));
```

Este código sustituye en la cadena str las apariciones de "*microsoft*" por "*W3Schools*"

- ***isNaN()***: Este método retorna un valor lógico cierto si y solo si la cadena contiene un valor que no es interpretable como un número. En caso contrario retorna un valor lógico *false*.

Manejo de expresiones regulares

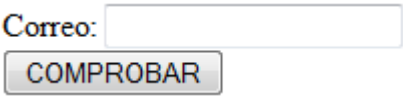
Las expresiones regulares son cadenas de texto que definen un *formato*. Estas se emplean para verificar si una cadena de caracteres introducida por el usuario verifica un determinado formato como puede ser el de una fecha, un número de teléfono, una dirección de correo electrónico..., etc.

Las expresiones regulares pueden declararse empleando dos sintaxis distintas:

- `var <expr_reg> = new RegExp(<expresión_regular>)`
- `var <expr_reg> = /<expresión_regular> /`

Una vez creado el objeto que representa la expresión regular pueden invocarse su método **test()** pasando una cadena como argumento. Si la cadena cumple con el formato descrito por la expresión regular el método retorna un valor lógico *cierto*. En caso contrario retorna *falso*.

Ejemplo: La siguiente página muestra una caja de texto en la que el usuario debe introducir una dirección de correo electrónico y un botón "COMPROBAR". Cuando el usuario pulse este botón debe mostrarse un mensaje indicando si la dirección introducida es correcta o no.



```

<!DOCTYPE html>
<html>
<head>
  <script>
    var correo;

    function validar(){
      var exprReg = /^(.+@.+.+)$ /;
      if (exprReg.test( correo.value)) {
        alert("Direccion correcta");           // DIRECCION CORRECTA
      } else {
        alert("La direccion no es valida");     // DIRECCION NO VALIDA
      }
    }

    function startup(){
      correo = document.getElementById("correo");
      var boton = document.getElementById("boton");
      boton.addEventListener("click", validar, false);
    }
    window.addEventListener("load", startup, false);
  </script>
</head>
<body>
  Correo: <input type="text" id="correo"><br >
  <input type="button" id="boton" value="COMPROBAR">
</body>
</html>

```

Fechas

Javascript define un tipo de objetos **Date** que permite la representación y manejo de fechas y horas en una página. Para obtener la fecha y hora actuales del equipo del usuario desde javascript sólo es necesario crear un objeto **Date**:

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

También es posible crear objetos **Date** que representan otras fechas:

- A partir de una cadena de caracteres:

```
var d0 = new Date("79/9/24"); // 24 septiembre 1974
```

- A partir de valores año, mes, día, hora, minutos, segundos:

```
var d2 = new Date(79,5,24,11,33,0); // 24 septiembre'74 11:33
```

Los objetos Date una vez creados disponen de múltiples métodos que permiten tanto obtener como modificar partes de una fecha:

Método	Descripción
<u>getDate()</u>	Retorna el día del mes (1-31)
<u>getDay()</u>	Retorna el día de la semana (0-6)
<u>getFullYear()</u>	Retorna el año (con 4 dígitos)
<u>getHours()</u>	Retorna la hora (0-23)
<u>getMilliseconds()</u>	Retorna los milisegundos (0-999)
<u>getMinutes()</u>	Retorna los minutos (0-59)
<u>getMonth()</u>	Retorna el mes del año (0-11)
<u>getSeconds()</u>	Retorna los segundos (0-59)
<u>getTime()</u>	Retorna el número de milisegundos transcurridos desde el 1 de Enero de 1970 (tiempo base de referencia).
<u>getTimezoneOffset()</u>	Retorna la diferencia entre la hora actual y la hora internacional UTC.
<u>parse()</u>	Devuelve el número de milisegundos transcurridos desde el 1 de Enero de 1970 de la cadena indicada como argumento.
<u>setDate()</u>	Fija el día del mes
<u>setFullYear()</u>	Fija el año (con 4 dígitos)
<u>setHours()</u>	Fija la hora.
<u>setMilliseconds()</u>	Fija los milisegundos.

<u>setMinutes()</u>	Fija los minutos.
<u>setMonth()</u>	Fija el mes del año.
<u>setSeconds()</u>	Fija los segundos.
<u>setTime()</u>	Fija la fecha a partir de un número de milisegundos transcurridos desde el 1 de Enero de 1970.
<u>toLocaleDateString()</u>	Retorna la fecha local como cadena de caracteres.
<u>toLocaleTimeString()</u>	Retorna la hora local como una cadena de caracteres.
<u>toUTCString()</u>	Retorna la fecha y hora universal UTC en forma de cadena de caracteres.

Todos los métodos *getXXX()* / *setXXX()* expuestos en la anterior table tienen su equivalente referido a la fecha/hora internacional UTC con la forma *getUTCXXX()* y *setUTCXXX()*. De este modo; el método que devuelve la hora local es ***getHours()***, y el método equivalente que devuelve la hora universal UTC es ***getUTCHours()***.

(*) La diferenciación entre hora local y hora universal UTC es debido a que en internet la hora válida es la universal ya que es la misma en todas las partes del mundo.

Ejemplo: El siguiente código muestra la hora local del equipo del usuario y la hora universal UTC equivalente en función de la parte del mundo en que se encuentre.

```
var ahora = new Date();
document.write("HORA LOCAL: " + ahora.toLocaleString());
document.write("<br />");
document.write("HORA UNIVERSAL: " + ahora.toUTCString());
```

El resultado mostrado es:

HORA LOCAL: miércoles, 09 de abril de 2014 20:37:02
HORA UNIVERSAL: Wed, 09 Apr 2014 18:37:02 GMT

Temporizadores

Los temporizadores y los intervalos son unos objetos especiales de Javascript que permiten establecer la ejecución de una función tras un intervalo de tiempo, o cada cierto intervalo de tiempo:

Intervalos

Un intervalo es un objeto que ejecuta una función de Javascript transcurrido un cierto intervalo de tiempo expresado en milisegundos. Para crear un intervalo se emplea la función **setTimeout()**:

```
var <objeto_intervalo> = setTimeout( <funcion_manejadora>, <intervalo> )
```

Esta función crea un objeto intervalo que ejecutará la función indicada por el parámetro **funcion_manejadora** transcurrido el lapso de tiempo en milisegundos indicado por el parámetro **intervalo**. Una vez ejecutada la función indicada el objeto intervalo deja de existir.

Un objeto intervalo puede ser anulado antes de ejecutar la función manejadora mediante la función **clearTimeout()**:

```
clearTimeout(<objeto_intervalo>);
```

Esta función recibe como parámetro la variable que contiene el objeto *intervalo* y lo elimina evitando que llegue a ejecutar la función indicada.

Temporizadores

Un temporizador es un objeto que ejecuta una función cada cierto intervalo de tiempo indefinidamente. Para crear un temporizador debe llamarse a la función **setInterval()**.

```
var <obj_temporizador> = setInterval( <funcion_manejadora>, <intervalo> )
```

Esta función crea un objeto *temporizador* que ejecutará la *funcion_manejadora* una y otra vez a intervalos de tiempo equivalentes a los milisegundos indicados en el parámetro *intervalo*.

Para detener el temporizador debe llamarse a la función **clearInterval()**:

```
clearInterval(<obj_temporizado >);
```

La función elimina el objeto temporizador indicado como valor evitando que la *funcion_manejadora* vuelva a ser ejecutada.

Ejemplo: El siguiente código muestra una página provista de dos botones en los que se muestra la hora actual del sistema actualizada una vez por segundo. El botón “Iniciar” inicia la visualización de la hora creando el objeto temporizador, y el botón “Pausar” lo detiene destruyendo el objeto temporizador.

```
<!DOCTYPE html>
<html>
<head>
  <script>
    var timer;
    var capa;

    // Funcion de inicializa el temporizador
    function iniciar() {
      timer = setInterval(mostrarHora, 1000);
    }

    // Funcion que finaliza el temporizador
    function detener() {
      clearInterval(timer);
    }

    // Funcion ejecutada por temporizador
    function mostrarHora() {
      var momento = new Date();
      var h = momento.getHours();
      var m = momento.getMinutes();
      var s = momento.getSeconds();
      cadena = "";
      cadena += (h < 9) ? "0" + h.toString() : h.toString();
      cadena += ":";
      cadena += (m < 9) ? "0" + m.toString() : m.toString();
      cadena += ":";
      cadena += (s < 9) ? "0" + s.toString() : s.toString();
      capa.innerHTML = cadena;
    }

    // Funcion de inicializacion
    function startup() {
      capa = document.getElementById("capa");
      var btn_inicio = document.getElementById("btn_inicio");
      btn_inicio.addEventListener("click", iniciar, false);
      var btn_fin = document.getElementById("btn_fin");
      btn_fin.addEventListener("click", detener, false);
    }
    window.addEventListener("load", startup, false);
  </script>
</head>
<body>
  <div id="capa">
    ESTO ES UNA CAPA
  </div>
  <br />
  <input id="btn_inicio" type="button" value="INICIAR RELOJ">
  <input id="btn_fin" type="button" value="PAUSAR RELOJ">
</body>
</html>
```

La función **iniciar()** crea un objeto temporizador llamado **timer** que ejecuta a intervalos regulares de 1000 milisegundos la función **mostrarHora()**. Esta obtiene a su vez la hora actual del sistema y genera una cadena de texto con el formato “hh:mm:ss” que es insertada como contenido HTML en la capa “**capa**” para su visualización. La función **detener()** elimina el objeto temporizador **timer** deteniendo así la ejecución de la función **mostrarHora()**.