Tema 2

Funciones y Objetos de JavaScript

Funciones en javascript

En esta primera parte de funciones en javascript estudiaremos como crear funciones definidas por los usurarios, como pasar parámetros y como devolver un valor.

Para crear funciones en javascript es tan sencillo como utilizar la palabra **function** e introducir entre llaves "{ }" el código que queremos que se ejecute cuando llamemos a la función.

```
function saludo()
{
        document.write("buenos días");
}
```

Para hacer referencia a la función sólo hay que teclear el nombre de la función seguido de los paréntesis:

```
saludo();
```

Las funciones hay que declararlas antes de ser utilizadas.

Si deseamos pasar valores a una función, en su declaración, hay que definir las variables separadas por coma y reciben el nombre de *argumentos*. Por cada variable que vaya a recibir la función hay que poner una variable.

```
function saludo(cadena)
{
        document.write(cadena);
}

saludo("mundo"); // mundo
```

La variable que recibe son locales a la función.

Si no pasamos un valor, la variable en la función estará "undefined".

```
saludo(); // undefined
```

Las variables van de obligatorias a opcionales.

```
function saludo(cadena,cadena2)
{
        document.write(cadena + " – " + cadena2);
}

saludo("mundo"); // mundo – undefined
saludo(,"todo"); // error
saludo('','todo'); //    – todo
```

Se puede pasar a un función valores o variables.

A diferencia de otros lenguajes en javascript no hay que indicar el número de argumentos que debe recibir una función. Cuando se invoca a una función se da un objeto de tipo matriz llamada "arguments" que contiene todos los argumentos recibidos en la función.

Los métodos y propiedades de los objetos matriz los veremos en capítulos posteriores. Ahora simplemente veremos un par de ejemplos para entender la variable "arguments".

Vemos un primer ejemplo donde muestra el número de argumentos recibidos.

```
<!DOCTYPE html>
<html>
<head>
<title>arguments</title>
<meta charset="UTF-8" />
<script>
function ejemplo()
{
  alert(arguments.length);
}
var a=b=c=0;
ejemplo(a);
ejemplo(a,b);
ejemplo(a,b,c);
</script>
</head>
<body>
Ejemplo de arguments
</body>
</html>
```

Con el siguiente ejemplo vemos todos los argumentos que hemos pasado.

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

```
<title>arguments</title>
<meta charset="UTF-8" />
<script>
function ejemplo()
{
        for(cont=0;cont<arguments.length;cont++)
                alert(arguments[cont]);
}
var a=1; var b=2; var c=3;

ejemplo(a);
ejemplo(a,b);
ejemplo(a,b,c);
</script>
</head>
<body>
Ejemplo de arguments
</body>
</html>
```

Si la función debe devolver un valor se utilizar la instrucción **return**

```
function suma(num1,num2)
{
        var sum=num1+num2;
        return sum;
}

resultado=suma(10,12);
```

En este ejemplo devuelve la suma de los números y se asigna a la variable **resultado**. Como vemos en el ejemplo anterior la función esta asignada a una variable. Si no realizamos dicha asignación, javascript, no genera ningún error, simplemente se pierde el valor.

En ocasiones podemos realizar lo mismo pero realizando la operación en la misma instrucción de **return**.

```
function suma(num1,num2)
{
        return num1+num2;
}

resultado=suma(10,12);
```

En ambos casos el resultado es el mismo.

Ámbito de las variables

Existen variables locales y globales.
Las variables *locales* son aquellas que se definen dentro de una función con la palabra reservada *var*. Dichas variables son visible sólo en dicha función. Fuera de dicha función la variable no existe.

```
<!DOCTYPE html>
<html>
<head>
<title>Ambito variables</title>
<meta charset="UTF-8" />
<script>
	function visualiza()
	{
		var mensaje="esto es un texto";
		alert(mensaje);
	}
	visualiza(); // visualiza el mensaje al acceder a la función y la variable es local
	alert(mensaje); // NO visualiza nada, la variable mensaje no existe
</script>
</head>
<body>
</body>
</html>
```

Las variables *globales* son aquellas que se definen fuera de la función y son visibles desde cualquier función.

```
<!DOCTYPE html>
<html>
<head>
<title>Ambito variables</title>
<meta charset="UTF-8" />
<script>
	var mensaje="esto es un texto";
	function visualiza()
	{
		alert(mensaje);
	}
	visualiza(); // visualiza el mensaje
	alert(mensaje); // visualiza el mensaje
</script>
</head>
<body>
</body>
```

```
</html>
```

Si utilizamos *var* para definir la variable, la variable tiene el ámbito de donde se declara. Si no definimos las variables con *var* las variables siempre son globales indistintamente de donde se definan.

```
<!DOCTYPE html>
<html>
<head>
<title>Ambito variables</title>
<meta charset="UTF-8" />
<script>
	function visualiza()
	{
		mensaje="esto es un texto";
		alert(mensaje);
	}
	visualiza(); // visualiza el mensaje
	alert(mensaje); // visualiza el mensaje
</script>
</head>
<body>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Ambito variables</title>
<meta charset="UTF-8" />
<script>
	mensaje="esto es un texto";
	function visualiza()
	{
		alert(mensaje);
	}
	visualiza(); // visualiza el contenido
	alert(mensaje); // visualiza el contenido
</script>
</head>
<body>
</body>
</html>
```

Si definimos una variable global y otra local con el mismo, la variable local tiene preferencia sobre la variable global.

```
<!DOCTYPE html>
<html>
<head>
<title>Ambito variables</title>
<meta charset="UTF-8" />
<script>
	var mensaje="global";
	function visualiza()
	{
		var mensaje="local";
		alert(mensaje);
	}
	visualiza(); // visualiza local
	alert(mensaje); // visualiza global
</script>
</head>
<body>
</body>
</html>
```

Objetos de javascript

Objetos

No es tema de este bloque la explicación de la Programación Orientada a Objetos (POO), pero, como este lenguaje utiliza objetos daremos una pequeña pincelada.

Un objeto es un conjunto de propiedades y métodos (datos y funciones) que están todos integrados dentro de una variable.

Esta definición no es una definición purista, pero, nos puede servir para hacernos una idea.

Realmente para hacer una introducción digna a la POO habría que hablar de *clases*, *objetos*, *herencia*, *polimorfismo*,.....

Realmente un objeto es una instancia de una clase, y una clase es la declaración de propiedades y métodos.

Para acceder a las propiedades o métodos se utiliza la siguiente sintaxis:

```
objeto.metodo

objeto.propiedad
```

Javascript no tiene clases.

Veamos a continuación los objetos propios del lenguaje.

String

Se utiliza para manipular cadenas. Cuando asignamos una cadena a una variable, javascript está creando un objeto de tipo *String*.

String indexa a partir de cero. Podemos acceder a cada uno de sus caracteres como si fuera un array.

También podemos crear un objeto *String* creando el objeto con su constructor:

```
cadena=new String("nueva cadena");
```

http://www.w3schools.com/jsref/jsref_obj_string.asp

Propiedades

length

Nos devuelve el número de caracteres que forman la cadena:

```
cadena="hola mundo";
alert(cadena.length);
```

Visulaiza --10 --

prototype

Nos permite asignar nuevas propiedades y métodos al objeto.

Esta propiedad se estudiará cuando tengamos más claro el funcionamiento de los objetos en JavaScript.

Métodos

charAt(indice)

Devuelve el carácter que ocupa la posición indicada por *índice*. El primer índice es 0 (cero).

```
cadena="hola mundo";
document.write(cadena.charAt(0));
document.write(cadena.charAt(5));
```

cadena.charAt(0) devuelve *h* el primer carácter de la cadena y *cadena.charat(5)* devuelve la letra *m*.

charCodeAt(indice)

Devuelve el código unicode del carácter que ocupa la posición marcada por *índice*. El primer índice es 0 (cero)

```
cadena=”hola mundo”;
document.write(cadena.charCodeAt(0));
```

Nos devuelve el código unicode de la letra *h* (104).

concat()

Se utiliza para agregar al objeto la cadena que se le pasa como atributo.

```
cadena="hola ";
acumula=cadena.concat("mundo");
alert(acumula); // hola mundo
document.write(cadena.concat("mundo")); // hola mundo
document.write("<br />");
document.write("buenos dias ".concat("mundo")); // buenos días mundo
```

fromCharCode(cod1,cod2,cod3.....)

Crea una cadena a partir de los códigos unicode que se le pasa como argumento. Este método se utiliza con el objeto genérico, es decir, se debe utilizar el objeto *String* no una instancia de él.

```
document.write(String.fromCharCode(35,36,37)); // #$%
```

indexOf(cadbus[,indice])

Devuelve la posición de la primera ocurrencia encontrada de *cadbus* empezando la búsqueda en la posición *índice*. La primera posición es el valor *0* y si no encuentra la ocurrencia devuelve el valor *-1*.

```
cadena="correo.electronico@servidor.com";
document.write(cadena.indexOf("a")); // -1
document.write(cadena.indexOf("id")); // 23
document.write(cadena.indexOf("id",24)); // -1
```

lastIndexOf(cadbus[,indice])

Funcionamiento parecido al método *indexOf* pero empezando la búsqueda por la derecha a partir del índice introducido o a partir del final de la cadena. El número que devuelve es contando los caracteres desde la izquierda. *índice* debe ser un número empezando a contar por la izquierda.

```
cadena="correo.electronico@servidor.com";
document.write(cadena.lastIndexOf("a")); // -1
document.write(cadena.lastIndexOf("o")); // 29
document.write(cadena.lastIndexOf("o",10)); // 5
document.write(cadena.lastIndexOf("O",10)); // -1
```

replace(buscar,reemplazar)

Este método en el parámetro “buscar”, utiliza una expresión regular. Esto se explicará en clases futuras, ahora, simplemente colocaremos el texto que queremos buscar para ser sustituido.

NO modifica la cadena, retorna la cadena modificada.

```
var resp;
var cadena=”per la carretera de relleu passa un carro carregat de torrat”;
resp=cadena.replace(“la”,”zz”);
alert(resp); // "per zz carretera de relleu passa un carro carregat de torrats"
```

search(cad_bus)

Devuelve la posición que ocupa la primera letra de “cad_bus”. Devuelve -1 si no encuentra dicha cadena.

“cad_bus” debe ser una expresión regular. Cuando se explique expresiones regulares podremos sacar el máximo rendimiento a este método. El índice empieza desde el valor '0'(cero).

```
var resp;
var cadena=”per la carretera de relleu passa un carro carregat de torrat””;
resp = cadena.search(“de”);
alert(resp); // 17
```

slice(primero,último)

Devuelve la porción de cadena comprendida entre la posición dadas por *primero* y *último*. Si no utilizamos el segundo argumento devuelve hasta el final de la cadena. Si *último* es negativo se cuentan las posiciones desde el final.

Hay que tener en cuenta que para el primer parámetro (primero) el índice empieza con el valor 0 y el segundo parámetro (último) el índice empieza con el valor 1.

Devuelve hasta la posición *último-1*.

```
cadena="Pere-Maria";
document.write(cadena.slice(1,3)); // er
document.write(cadena.slice(1,-3)); // ere-Ma
```

split(separador[,limite])

Devuelve una array de la cadena separando los elementos por el carácter indicado. Si separador es una cadena vacia ("") *split* nos devuelve un array donde cada elementos es una letra de la cadena.

Separador puede ser un carácter o una expresión regular.

El argumento límite es para indicar el número de elementos de array que vamos a crear como máximo.

```
cadena="juan, andres, ruben, ricardo, francisco, luis";
cadena.split(","); // ["juan", " andres", " ruben", " ricardo", " francisco", " luis"]
cadena.split(", "); // ["juan", "andres", "ruben", "ricardo", "francisco", "luis"]
cadena.split(", ",3); // ["juan", "andres", "ruben"]
```

substr(inicio[,largo])

Extrae una cadena del objeto *String* desde el carácter que ocupa la posición que marca el argumento *inicio* y con un número de caracteres marcado por el argumento *largo*. Si el argumento *largo* se omite se extrae hasta el final de la cadena.

La cadena empieza por la posición 0.

```
cadena="Pere-Maria";
document.write(cadena.substr(2,5)); // re-Ma
```

Si el parámetro "inicio" es un valor negativo, se extrae dicho número de caracteres desde la derecha hacia la izquierda. Si "inicio" es negativo se ignora el segundo parámetro.

```
cadena="Pere-Maria";
document.write(cadena.substr(-3)); // ria
```

substring(indice1,indice2)

Extrae una cadena del objeto *String* desde el carácter que ocupa la posición del argumento de menor valor hasta la posición indicada por el argumento de mayor valor. Para calcular hasta donde extraer siempre se realiza de la siguiente forma: desde el argumento de menor valor hasta (el argumento de mayor valor menos 1)

```
cadena="Pere-Maria";
document.write(cadena.substring(1,5)); // ere-
document.write(cadena.substring(5,1)); //  ere-
```

toLowerCase()

Devuelve toda la cadena en minúsculas.

toUpperCase()

Devuelve toda la cadena en mayúsculas.

Métodos de presentación

Métodos no estándar, pueden variar entre los diferentes navegadores. Se enumeran para que se conozcan. No los usaremos. Se estudiará hacer lo mismo con DOM.

anchor(nombre)

Este método crea una etiqueta de ancla en html () El texto que asignamos al atributo *name* es el *nombre* que se pasa como argumento al método.

```
<html>
<head>
</head>
<body>
<script>
cadena="parte de arriba";
document.write(cadena.anchor("aqui"));
</script>
<br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br />
<br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br />
<br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br />
<a href="#aqui">subir</a>
</body>
</html>
```

En la instrucción *document.write(cadena.anchor("aqui"));* en el navegador aparece el contenido del objeto *cadena*.

big()

Muestra la cadena con una fuente grande. Devuelve el contenido del objeto entre etiquetas html *<big></big>*

```
document.write(cadena.big());
```

blink()

Muestra la cadena con un efecto de parpadeo. Devuelve el contenido del objeto entre las etiquetas html *<blink></blink>*.

```
document.write(cadena.blink());
```

bold()

Muestra la cadena con un efecto de negrita. Devuelve el contenido de objeto entre las etiquetas **.

```
document.write(cadena.bold());
```

fixed()

Visualiza la cadena como si estuviera entre las etiquetas *<tt></tt>*. Etiquetas de espaciado fijo, siempre hay la misma separación entre caracteres.

```
document.write(cadena.fixed());
```

fontcolor(color)

Cambia el color con el que se muestra la cadena. El valor *color* se pasa entre comillas y se puede asignar un nombre de un color o crear un color con el rojo, verde y azul (#RRGGBB). Algunos nombres de color que podemos utilizar son: “red”, ”blue” ,”yellow”, “purple”, “darkgray”, “olive”, “salmon”, “black”, “white”,....

```
otracadena=cadena.fontcolor("salmon");
document.write(cadena.fontcolor("salmon") + "<br />");
document.write(otracadena);
```

fontsize(tamaño)

Cambia el tamaño de la letra. El tamaño que le podemos mandar como argumento esta ente el 1 y el 7. El argumento se pasa entre comillas. El valor 1 es el menor y el valor 7 es el mayor.

```
document.write(cadena.fontsize("7"));
```

italics()

Muestra la cadena es cursiva. Devuelve una cadena entre las etiquetas **.

```
cadena="correo.electronico@servidor.com";
nueva=cadena.italics();
alert(nueva);
```

link(url)

Este método crea, a partir de un objeto String, un enlace. Devuelve una cadena con las etiquetas **. La cadena es el texto que muestra en el navegador y el argumento que recibe es el enlace.

```
cadena="Pere Maria";
document.write(cadena.link("http://www.iesperemaria.com"));
```

small()
Devuelve la cadena del objeto entre las etiquetas *<small></small>* (fuente pequeña).

strike()
Devuelve una cadena envuelta entre las etiquetas *<strike></strike>*. Cadena de caracteres tachada.

sub()
Devuelve una cadena envuelta entre las etiquetas **. Cadena con formato subíndice.

sup()
Devuelve la cadena entre las etiquetas ** (superíndices). Cadena con formato de superíndice.

Math

Objetos que con sus propiedades pueden ayudarnos a realizar cálculos matemáticos.

Math no es un constructor. Todas las propiedades y métodos de este objeto pueden ser llamadas con el objeto Math, no se crea un objeto.

http://www.w3schools.com/jsref/jsref_obj_math.asp

Propiedades

Sus propiedades no son modificables.

E	Número “e” base de los logaritmos naturales.(neperianos)
LN2	Logaritmo neperiano de 2
LN10	Logaritmo neperiano de 10
LOG2E	Logaritmo de base 2 del número e
LOG10E	Logaritmo de base 10 del número e
PI	Número PI
SQRT1_2	Raíz cuadrada de ½
SQRT2	Raíz cuadrada de 2

```
<!--
document.write("Math.E =>" + Math.E + "<br/>"); // 2.718281828459045
document.write("Math.LN2 =>" + Math.LN2 + "<br/>"); // 0.6931471805599453
document.write("Math.LN10 =>" + Math.LN10 + "<br/>"); // 2.602585092994046
document.write("Math.LOG2E =>" + Math.LOG2E + "<br/>"); // 1.4426950408889633
document.write("Math.LOG10E =>" + Math.LOG10E + "<br/>"); // 0.4342944819032518
document.write("Math.PI =>" + Math.PI + "<br/>"); // 3.141592653589793
document.write("Math.SQRT1_2 =>" + Math.SQRT1_2 + "<br/>"); // 0.7071067811865476
document.write("Math.SQRT2 =>" + Math.SQRT2 + "<br/>"); // 1.4142135623730951
//-->
```

Métodos

abs

Valor absoluto. Devuelve el número sin signos.

Math.abs(número)

acos

Arco coseno. Los valores deben estar entre 1 y -1. Acepta decimales. Devuelve una unidad en radianes o NaN.

Math.acos(número)

```
alert("Math.abs(-23.45)=>" + Math.abs(-23.45)); // 23.45
alert("Math.abs(23.55)=>" + Math.abs(23.55)); // 23.55
alert("Math.acos(1)=>" + Math.acos(1)); // 0
alert("Math.acos(-1)=>" + Math.acos(-1)); // 3.141592653589793
alert("Math.acos(0.89)=>" + Math.acos(0.89)); // 0.4734511572720662
alert("Math.acos(-0.89)=>" + Math.acos(-0.89)); // 2.6681414963177272
```

asin

Arco seno. Sirve para calcular el ángulo cuyo seno es el valor dado por el argumento, es decir, el llamado arcosen. Este argumento deberá ser una expresión numérica, o transformable en numérica, comprendida entre -1 y +1 y el ángulo devuelto viene dado en radianes.

Math.asin(número)

```
alert("asin(1)=>" + Math.asin(1)); // 1.5707963267948966
alert("asin(4.5)=>" + Math.asin(4.5)); // NaN
```

atan

Arco tangente

Función arcotangente. Devuelve un valor cuyas unidades son radianes o NaN.

Math.atan(número)

```
alert("atan(4.5)=>" + Math.atan(4.5)); // 1.3521273809209546
alert("atan(-4.5)=>" + Math.atan(-4.5)); // -1.3521273809209546
```

atan2

Arco tangente. Devuelve el ángulo desde el eje x hasta el punto (x,y) (en radianes). El valor devuelto está en PI y -PI

Math.atan2(x,y)

ceil

Redondeo superior. Devuelve el número redondeado por exceso, es decir, 4'25 --> 5

Math.ceil(número)

```
alert("ceil(1.2)=>" + Math.ceil(1.2)); // 8
alert("ceil(-1.2)=>" + Math.ceil(-1.2)); // -1
alert("ceil(4.5)=>" + Math.ceil(4.5)); // 5
alert("ceil(-4.5)=>" + Math.ceil(-4.5)); // -4
alert("ceil(8.7)=>" + Math.ceil(8.7)); // 9
alert("ceil(-8.7)=>" + Math.ceil(-8.7)); // -8
```

cos

Coseno. Devuelve el coseno de un número o NaN. El número que se le pasa como argumento debe estar en radianes.

Math.cos(número)

```
alert(“cos(360)=>" + Math.cos(360) ); // -0.2836910914865273
alert("cos(-360.20)=>" + Math.cos(-360.20) ); // -0.46854330225681967
```

exp

Exponencial. Devuelve el número **e** elevado al exponente (número recibido como argumento).

Math.exp(número)

```
alert(“exp(4.5)=>" + Math.exp(4.5)); // 90.01713130052181
alert("exp(-4.5)=>" + Math.exp(-4.5)); // 0.011108996538242306
```

floor

Redondeo inferior. Devuelve el valor del argumento redondeado por defecto, es decir, el mayor número entero menor o igual al argumento. Si el argumento fuera no numérico será convertido a numérico siguiendo las reglas de la función parseInt() o parseFloat().

Math.floor(número)

```
alert("floor(1.2)=>" + Math.floor(1.2)); // 1
alert("floor(-1.2)=>" + Math.floor(-1.2)); // -2
alert("floor(4.5)=>" + Math.floor(4.5)); // 4
alert("floor(-4.5)=>" + Math.floor(-4.5)); // -5
alert("floor(8.7)=>" + Math.floor(8.7)); // 8
alert("floor(-8.7)=>" + Math.floor(-8.7)); // -9
```

log

Logaritmo natural. Devuelve el logaritmo natural o neperiano de un número.

Math.log(número)

max

Máximo.

Devuelve el valor máximo de los números o expresiones pasadas como argumentos.

Math.max(número1,número2)

min

Mínimo. Devuelve el menor número de los valores o expresiones pasadas como argumento.

Math.min(numero1,numero2)

pow

Potencia de un número elevado a un exponente.

Math.pow(base,exponente)

random

Número al azar entre el valor 0 y 1.

Math.random()

round

Redondear

Redondea al número entero más cercano.

Math.round(número)

```
alert("round(1.2)=>" + Math.round (1.2)); // 1
alert("round(-1.2)=>" + Math.round (-1.2)); // -1
```

```
alert("round(4.5)=>" + Math.round (4.5)); // 5
alert("round(-4.5)=>" + Math.round (-4.5)); // -4
alert("round(8.7)=>" + Math.round (8.7)); // 9
alert("round(-8.7)=>" + Math.round (-8.7)); // 9
```

sin

Seno. Devuelve el seno de un número (que debe estar en radianes) o NaN.

Math.sin(número)

sqrt

Raíz cuadrada. Devuelve la raíz cuadrada del número pasado como argumento.

Math.sqrt(número)

tan

Tangente. Sirve para calcular la tangente del ángulo pasado como argumento en radianes. Este argumento deberá ser una expresión numérica o transformable en numérica.

Math.tan(número)

Date

Con este objeto podremos manipulas las fechas. JavaScript maneja fechas en milisegundos. Los meses vienen representados en rangos entre 0 y 11. Los días de la semana vienen representados con el 0 para domingo, 1 para lunes....., 6 para sábado.

Estos rangos hay que tenerlos en cuenta cuando obtengamos los valores de los métodos del objeto. Para asignarle valores hay que utilizar fechas válidas.

Para crea un objeto fecha los podemos hacer de cualquiera de las siguientes formas:

```
var fecha= new Date(); \\ fecha del día de hoy
\\ número en milisegundos desde la media noche del 1/1/70
var fecha= new Date(número);
var fecha= new Date(cadena);
var fecha= new Date(año, mes, día[, hora[, minutos[, seg[,ms]]]]);
```

```
var evento = new Date("November 10 1990");
var otro = new Date("10 Nov 1990");
var otro = new Date("10/02/2000"); //Oct, 2, 2000
// el mes se pasa 0 para enero, 11 par diciembre
var instante = new Date(1990, 10, 10, 20,00);
```

Donde se usen cadenas para indicar una fecha podemos añadir al final las siglas GMT (o UTC) para indicar que la hora se refiere a hora del meridiano Greenwich, si no se toma como hora local, o sea, según la zona horaria configurada en el ordenador donde se ejecute el script.

http://www.w3schools.com/jsref/jsref_obj_date.asp

Métodos

getTimezoneOffset()

Nos da la diferencia horaria en minutos del ordenador con respecto al meridiano de Greenwich

getTime()

Devuelve el tiempo transcurrido en milisegundo desde el 1/1/70 hasta la fecha.

```
fecha= new Date(2006,2,5,18,57);
alert("fecha.getTimezoneOffset() => " + fecha.getTimezoneOffset()); // -60
alert("fecha.getTime() => " + fecha.getTime());
```

getDate()

Devuelve el día actual del mes. Devuelve un valor entre 1 y 31.

getDay()

Devuelve el día de la semana. Devuelve un valor entre 0 (domingo) y 6 (sabado).

getMonth()

Devuelve el número de mes de la fecha actual. Devuelve un valor entre 0 (enero) y 11 (diciembre).

getYear()

Devuelve el año actual como un entero. **Obsoleto**, se mantiene por compatibilidad con versiones anteriores. Se aconseja utilizar *getFullYear()*. El valor que devuelve es el año menos 1900.

fecha = new Date("10/10/2009");
fecha.getYear() ;--> 109.

getFullYear()

Nos devuelve el año actual con cuatro dígitos.

getHours()
Nos devuelve la hora de la fecha. Devuelve un valor entre 0 y 23.

getMinutes()
Nos devuelve los minutos de la fecha. Devuelve un valor entre 0 y 59.

getSeconds()
Nos devuelve los segundo de la fecha. Devuelve un valor entre 0 y 59.

getMilliseconds()
Nos devuelve los milisegundos de la fecha. Devuelve un valor entre 0 y 999

```
fecha= new Date(2006,2,5,18,57,30,360); // 5/3/2066 a las 18:57:30,360
alert("fecha.getDate() => " + fecha.getDate()); // 5
alert("fecha.getMonth() => " + fecha.getMonth()); // 2
alert("fecha.getYear() => " + fecha.getYear()); // 106
alert("fecha.getFullYear() => " + fecha.getFullYear()); // 2006
alert("fecha.getHours() => " + fecha.getHours()); // 18
alert("fecha.getMinutes() => " + fecha.getMinutes()); // 57
alert("fecha.getSeconds() => " + fecha.getSeconds()); // 30
alert("fecha.getMilliseconds() => " + fecha.getMilliseconds()); // 360
```

setTime(milisegundos)
Asigna la fecha que se recibe en milisegundos.

setDate(díames)
Nos permite cambiar el día de mes de la fecha. Debe recibir un valor entre 1 y 31. Si damos un valor fuera de este rango intenta ajustarlo a una fecha válida a partir del 1 del mes de la fecha y cambiando de mes.

setMonth(nºmes)
Nos permite cambiar el mes de la fecha. Debe recibir un valor entre 0 (enero) y 11 (diciembre). Si colocamos valores fuera de este rango intenta ajustar a una fecha válida a partir del primer mes del año y cambiando de año.

setYear
Nos permite cambiar el año de la fecha. Si pasamos sólo el valor “00” y utilizamos este método asignará el año 1900 y no el 2000. Está **obsoleto** y se mantiene por compatibilidad con versiones anteriores. Se aconseja utilizar *setFullYear*.

setFullYear(anyo)
Nos permite cambiar el año de la fecha. Debemos pasarle los cuatro dígitos del año.

setHours(hora)
Nos permite asignar una nueva hora al objeto fecha.

setMinutes(minuto)

Nos permite cambiar los minutos a la fecha del objeto.

setSeconds(segundo)

Nos permite cambiar los segundos a la fecha del objeto.

setMilliSeconds(milisegundo)

Nos permite cambiar los milisegundos a la fecha del objeto.

```
fecha= new Date();
fecha1=new Date(2006,1,5);
fecha.setTime(fecha1.getTime()) // fecha => 5/2/2006 0:0:0,0
fecha.setDate(15); // => día 15
alert("fecha.getDate() => " + fecha.getDate()); // 15
fecha.setMonth(0); // => mes Enero
alert("fecha.getMonth() => " + fecha.getMonth()); // 0
fecha.setYear(0); // => año 1900
alert("fecha.getYear() => " + fecha.getYear()); // 0
alert("fecha.getFullYear() => " + fecha.getFullYear(); // 1900
fecha.setFullYear(2000); // => año 2000
alert("fecha.getYear() => " + fecha.getYear()); // 100
alert("fecha.getFullYear() => " + fecha.getFullYear()); //2000
fecha.setHours(15); // => hora 3 de la tarde
alert("fecha.getHours() => " + fecha.getHours()); // 15
fecha.setMinutes(15); // => minuto 15
alert("fecha.getMinutes() => " + fecha.getMinutes()); // 15
fecha.setSeconds(30); // => segundos 30
alert("fecha.getSeconds() => " + fecha.getSeconds()); // 30
fecha.setMilliseconds(100); // => Milisegundo 100
alert("fecha.getMilliseconds() => " + fecha.getMilliseconds()); // 100
```

toGMTString()

Este método nos devuelve el objeto en cadena de caracteres según el estándar GMT. **Obsoleto**. Se recomiendo *toUTCString()*.

GMT *Greenwich Mean Time*: tiempo promedio del Observatorio de Greenwich, en Londres

toUTCString()

Igual que *toGMTString* pero según el estándar UTC. **Tiempo Universal Coordinado**, también conocido como "*tiempo civil"*

toLocaleString() – toLocalDateString() – toLocalTimeString()

Devuelve la fecha y la hora convertida en una cadena utilizando la configuración regional.

valueOf

Devuelve el valor primitivo del objeto *Date*. Devuelve la hora en milisegundos contando a partir de medianoche del 1/1/70

toString

Devuelve una representación en formato cadena del objeto *Date*.

```
fecha= new Date();
fecha1=new Date(2006,1,5);
fecha.setTime(fecha1.getTime()); // fecha => 5/2/2006 0:0:0,0
alert("fecha.toGMTString() => " + fecha.toGMTString());
                                        // Sat, 04 Feb 2006 23:00:00 GMT
alert("fecha.toUTCString() => " + fecha.toUTCString());
                                        // Sat, 04 Feb 2006 23:00:00 GMT"
alert("fecha.toLocaleString() => " + fecha.toLocaleString());
                                        // 5/2/2006 00:00:00
alert("fecha.valueOf() => " + fecha.valueOf()); // 1139094000000
alert("fecha.toString() => " + fecha.toString());
                                        // Sun Feb 05 2006 00:00:00 GMT+0100 (CET)
```

Number

Objeto destinado al manejo de datos y constantes numéricas. No es habitual crear objetos de este tipo ya que JavaScript lo crea automáticamente.

La sintaxis de creación es la misma que con el resto de objetos.

var numero= new Number(valor_inicial)

http://www.w3schools.com/jsref/jsref_obj_number.asp

Propiedades

Las propiedades de este objeto se deben utilizar con el objeto “Number”. No las podemos utilizar con variables que contengan este tipo de objeto. Es decir, sólo son accesibles como Number.MAX_VALUE

MAX_VALUE

Indica el valor máximo finito positivo utilizable por JavaScript. Según ECMAScript-262 aproximadamente $1.7976931348623157 \times 10^{308}$.

MIN_VALUE

Indica el valor mínimo finito positivo utilizable por JavaScript. Según ECMAScript-262 aproximadamente 5×10^{-324} .

NaN

Una constante usada para indicar que una expresión ha devuelto un valor no numérico. NaN no puede compararse usando los operadores lógicos habituales, para ver si un valor es igual a NaN se debe usar la función incorporada isNaN.

```
var num=25;
alert(num.NaN) // undefined
```

NEGATIVE_INFINITY

Una constante para indicar infinito negativo.

POSITIVE_INFINITY

Una constante para indicar infinito positivo.

Métodos

toString([x])

Devuelve el número en una cadena. Si no pasamos parámetro o le pasamos el valor "10" nos devuelve el número en cadena en base 10. Si parámetro es un valor entre 2 y 36 nos devuelve el número en formato cadena pero convertido a la base que hemos pasado como parámetro.

```
var numero=new Number(234.125);
alert(numero); // 234.125
alert(numero.toString()); // 234.125
alert(numero.toString(2)); // 11101010.001
alert(numero.toString(8)); // 352.1
alert(numero.toString(16)); // ea.2
```

toLocaleString()

Devuelve el número en cadena colocando el carácter decimal en . (punto) o en , (coma) según la localización del cliente.

toFixed(x)

Indicamos que el objeto Number tiene x números decimales, redondeando hacia arriba. Si faltan decimales se rellenan con ceros (a la derecha de los decimales)

```
var numero=new Number(234.1527364);
alert(numero);              //  234.1527364
alert(numero.toFixed(5)); // 234.15274
alert(numero.toFixed(4)); // 234.1527
alert(numero.toFixed(9)); // 234.152736400
```

toPrecision(x)

Indicamos que el objeto Number está compuesto por x números contando los enteros y los decimales. Si Faltan decimales se rellenan con ceros (a la derecha de los decimales).

```
var numero=new Number(234.1527364);
alert(numero);                    // 234.1527364
alert(numero.toPrecision(5)); // 234.15
alert(numero.toPrecision(4)); // 234.2
alert(numero.toPrecision(11)); // 234.15273640
```

toExponential(x)

Devuelve una cadena que representa a un número en forma exponencial. Devuelve un dígito por delante del punto decimal y la parte decimal redondeada a x dígitos.

```
var numero=new Number(234.1527364);
alert(numero);                          // 234.1527364
alert(numero.toExponential(5)); // 2.34153e+2
alert(numero.toExponential(4)); // 2.3415e+2
alert(numero.toExponential(11)); //  2.34152736400e+2
```

Boolean

Este objeto nos permite crear variables boleanas, es decir, variables que sólo pueden contener el valor verdadero o falso (true, false).

```
a = new Boolean(); => Asigna el valor false a la variable a
a = new Boolean(0); => Asigna el valor false a la variable a
a = new Boolean(“”); => Asigna el valor false a la variable a
a = new Boolean(false); => Asigna el valor false a la variable a
a = new Boolean(núm_distinto_cero); => Asigna el valor true a la variable a
a = new Boolean(true); => Asigna el valor true a la variable a
a = new Boolean(“ “) => Asigna el valor true a la variable a
```

http://www.w3schools.com/jsref/jsref_obj_number.asp

Métodos

valueOf()

Devuelve el valor boleano.

toString()

Devuelve el valor boleano en una cadena.

```
var boleano = new Boolean();
alert(typeof(boleano.valueOf()); --> boolean
alert(typeof(boleano.toString()); --> string
```

Array

Esto objeto nos va a permitir crear arrays. Los elementos que componen un array pueden ser cualquier tipo.

Para construir un objeto array habrá que crearlo con el constructor.

```
vector=new Array(15);
```

Los índices del array anterior van desde el valor 0 hasta el valor 14.

También podemos definir un array sin indicar un número determinado de elementos.

```
vector = [];
```

Para acceder a un elemento del array se utilizan la siguiente nomenclatura *vector[i]*. El índice debe ir entre corchetes.

También podemos definir un array y asignar valores al mismo tiempo:

```
var vector=new Array(1,"uno",2,"dos");
```

Esta opción no es muy aconsejable. Si colocamos sólo un valor estamos indicando el número de elementos de dicho array, por tanto, si queremos crear un array con un único valor no podemos utilizar este constructor.

```
var vector=[1,"uno",2,"dos];
```

Para definir un array bidimensional tenemos que declarar un array y en cada elemento declarar otro array.

```
vector=new Array(4);
for(i=0;i<vector.length;i++)
  {
    vector[i]=new Array(3);
  }
```

Para acceder a cada elemento de un array bidimensional utilizamos la siguiente sintaxis:

```
vector[x][y];
```

http://www.w3schools.com/jsref/jsref_obj_array.asp

Propiedades

length

Devuelve la longitud del array, es decir, el número de elementos del array.

```
var vector = new Array(10);
alert(vector.length);
```

prototype

Nos permite asignar nuevas propiedades al objeto.

Esta propiedad se estudiará cuando tengamos más claro el funcionamiento de los objetos en JavaScript.

Métodos

concat(objArray)

Se utiliza para concatenar dos arrays. Une el objeto *Array* con el array que se le pasa como argumento. El resultado lo devuelve en un nuevo array. Los array que se concatenan no se modifican.

indexOf(searchelement [,inicio])

Busca el elemento en el array y devuelve su posición. Si no encuentra el elemento en el array devuelve -1. *inicio* indica la posición desde donde empieza a buscar.

join(separador)

Forma una cadena de los elementos del array separando los elementos por el argumento que se le indicada.

Si no indicamos ningún separador utiliza la coma (,).

```
vector=new Array(1,"uno",2,"dos");
alert(vector.join()); // 1,uno,2,dos
alert(vector.join(",")); //  1,uno,2,dos
alert(vector.join("-")); // 1-uno-2-dos
```

lastIndexOf(searchelement [,inicio])

Funciona exactamente igual de *indexOf()*. Se diferencia es que empieza a buscar por el último elemento del array.

pop()

Borra y devuelve el último elemento del array.

push(elemento)

Añade el elemento que recibe como argumento y devuelve el número de elementos que forman el array.

reverse()

Invierte el orden de los elementos de un Array en el propio array, sin crear uno nuevo.

shift()

Borra y devuelve el primer elemento del array.

slice(inicio, fin)

Extrae parte de un Array devolviéndolo en un nuevo objeto Array desde el elemento *inicio* hasta el elemento *fin-1*. Si se omite el segundo argumento *fin* será el último elemento del array y si ponemos un valor negativo sin el segundo argumento se extraen los elementos desde el final.

Si el valor negativo se lo pasamos al argumento *fin* tiene el siguiente efecto. Si el argumento *inicio* es positivo se colocará en el elemento marcado por el primer argumento y luego irá quitando por el final el número de elementos que hayamos pasado en negativo.

Si el primer argumento es negativo y el segundo también empieza a contar desde el final del array.

```
vector=new Array(1,"uno",2,"dos",3,"tres",4,"cuatro");
otrov=vector.slice(4,6);
alert(otrov.join()); // 3,tres
otrov=vector.slice(4);
alert(otrov.join()); // 3,tres,4,cuatro
otrov=vector.slice(-5);
alert(otrov.join()); // “dos”,3,”tres”,4,”cuatro”
otrov=vector.slice(3,-1);
alert(otrov.join()); // “dos”,3, "tres", 4
otrov=vector.slice(-3,-1);
alert(otrov.join()); // "tres", 4
```

sort()

Ordena alfabéticamente los elementos de un objeto Array.

Puede recibir como argumento una función que marque los criterios de ordenación de la tabla. Esta función posee dos argumentos y devolverá un valor negativo si el primer argumento es menor que el segundo, cero si son iguales y un valor positivo si el primer argumento es mayor que el segundo.

En castellano esto es necesario si queremos que la ñ y vocales acentuadas figuren en su lugar.

```
function compareNumbers(a, b) {
  return b – a
}
numberArray.sort(compareNumbers); // ordena en orden inverso
```

splice(index,nelementos,item1,item2,..... ,itemx)

Añade/borra elementos de un array. ,

index. Indica a partir de que elemento se añade/borra elementos. Si es un valor negativo empieza desde el último elemento del array. El indice impieza desde 0

nlementos. Número de elementos que vamos a borrar desde la posición *index*. Si ponemos un valor 0 no borraremos ningún elemento. La posición empieza desde 1.

item1,.....,itemx. Elementos que se añadirán a la tabla desde la posición indicada en *index*.

```
var array1=["luis","andres","juan"];
alert(array1); // luis,andres,juan
array1.splice(0,1,"ricardo");
alert(array1); // ricardo,andres,juan
array1.splice(0,1,"pepe","gema");
alert(array1); // pepe,gema,andres,juan
```

unshift(elemento,elemento, ...)

Añade uno o más elementos al inicio del array.

```
vector=new Array(1,"uno",2,"dos",3,"tres",4,"cuatro");
alert(vector.toString()); // 1,uno,2,dos,3,tres,4,cuatro
vector.unshift("añade","vuelve","retorna")
alert(vector.toString()); // añade,vuelve,retorna,1,uno,2,dos,3,tres,4,cuatro
```

forEach()

Por cada elemento del array ejecuta una función que recibe como parámetro. Esta función puedes definir tres parámetros. El primero será el elemento, el segundo es el indice y el tercero es el propio array. NO devuelve ningún valor.

```
var array1=[1,2,3,4,5];
var sum=0;
array1.forEach(function(item,index,vector){
      sum+=item;
  });
alert(sum); // suma todos los elementos del array
```

map()

Recorre cada elemento del array y retorna un array con todos los elementos retornados por la función que recibe como parámetro. La función que pasamos como parámetro también recibe tres valores igual que en forEach();

```
a = [1, 2, 3];
b = a.map(function(x) { return x*x; }); // b es [1, 4, 9]
```

filter()

Recorre cada elemento del array y retorna los elementos que cumplan una condición. La función que recibe como parámetro también podemos definir tres parámetros.

```
a=[22,14,56,32,14,58,74,95,12];
b=a.filter(funcion(item){
      return item>30
   });
alert(b); // [56,32,58,74,95]
```

every()

Retorna verdadero o falso. Devuelve verdadero si y solo si todos los elementos del array cumplen un condición

```
a = [1,2,3,4,5];
a.every(function(x) { return x < 10; })      // => true: Todos los valores son < 10.
a.every(function(x) { return x % 2 === 0; }) // => false: no todos los valores son par.
```

same()

Parecido al anterior. Devuelve verdadero si existe al menos un elemento que cumpla la condición. La función que pasamos puede tener tres parámetros.

```
a = [1,2,3,4,5];
a.some(function(x) { return x%2===0; }) // => true hay algún numero par
a.some(isNaN)                           // => false: no hay alfanuméricos
```

reduce() y **reduceRight()**

Estos métodos combinan los elementos del array y producen un único resultado. Estos métodos reciben dos argumentos: el primero es la función y el segundo es la inicialización del primer parámetro de la función. La función que pasamos a estos métodos tienen cuatro parámetros: una variable, ítem, index y el array.

```
var a = [1,2,3,4,5]
var sum = a.reduce(function(x,y) { return x+y }, 0);      // suma los valores
var product = a.reduce(function(x,y) { return x*y }, 1); // multiplica los valores
var max = a.reduce(function(x,y) { return (x>y)?x:y; }); // valor más alto
```

reduceRight() funciona igual que el reduce pero empezando a recorrer el array desde el índice más alto hacia el más pequeño. Es decir, recorre el array al revés.

https://www.inkling.com/read/javascript-definitive-guide-david-flanagan-6th/chapter-7/ecmascript-5-array-methods

https://coderwall.com/p/_ggh2w

Object

Existe un objeto llamado *Object* del que derivan todos los objetos de JavaScript, los predefinidos y los definidos por el usuario.

Esto significa que los objetos usados en JavaScript heredan las propiedades y métodos de Object.

Métodos

toString

Devuelve una cadena dependiendo del objeto en que se use

Objeto	Cadena devuelta por el método
Array	Los elementos del array separados por coma
Boolean	Si el valor es false devuelve "false" si no devuelve "true"
Function	la cadena "function nombre_de_función(argumentos){ [código]}"
Number	Representación textual del número
String	El valor de la cadena
Default	"[object nombre_del_objeto]"

valueOf

Devuelve el valor del objeto dependiendo del objeto en que se use

Objeto	Valor que devuelve el método
Array	Una cadena formada por los elementos separados por coma
Boolean	El valor booleano (true o false)
Date	La fecha como el número de milisegundos desde el 1/1/1970, 00:00
Function	La propia función
Number	El valor numérico
String	La cadena
Default	El propio objeto

Propiedades

constructor

Esta propiedad contiene una referencia a la función que crea las instancias del objeto en particular. Por ejemplo:

```
x = new String("Hola");
//En este caso x.constructor contendrá
//alert(x.constructor);
//function String() { [native code] }
```

prototype

Es una propiedad utilizada para asignar nuevos métodos o propiedades a un objeto, elementos estos que serán heredados por las diferentes instancias de ese objeto. Ejemplo:

```
Array.prototype.nombTipo = "matriz";
lista = new Array(9);
alert(lista.nombTipo);
//Escribirá la palabra matriz que es el nombTipo
//que hemos dado para el objeto Array
```

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

Number.prototype.valorbase=15;
Number.prototype.quita = resta;

var x = new Number;
var y = x.quita(20,50 );
alert(y); // 30
alert(x.valorbase); // 15
y=x.quita(x.valorbase,100);
alert(y); // 85
```

```
function ver(){ return this.valueOf(); }
Number.prototype.mira=ver;
a=10;
alert(a.mira()); // 10
```

El operador *instanceof* nos sirve para determinar la clase concreta del objeto.

variable1 instanceof xxxxxxx;

Esta instrucción devuelve true si el objeto variable1 es del tipo xxxxxxx. Sustituiremos xxxxxxx por cualquiera de los objetos de javascript que conocemos: String, Array, Number.....

http://www.w3schools.com/jsref/default.asp

Con el ECMAScript 6 (junio 2015), han añadido nuevos métodos y propiedades en los objetos descritos en este documento. No todos son operativos y varían de un navegador a otro.

Para poder depurar correctamente los diferentes scripts que creemos podemos apoyarnos en la consola de depuracion yen las API de la consola de depuración.

En esta dirección https://getfirebug.com/wiki/index.php/Console_API tu puede encontrar todas las propiedades del objeto console. Para empezar podemos utilizar los siguientes métodos:

console.log("cadena")
Muestra cadena en la ventana de consola del depurador.

console.table(tabla)
En la consola del depurador dibuja una tabla con filas y columnas.

console.dir(objeto)
Muestra las propiedades del objeto.