Funciones y métodos útiles 2

Como todo lenguaje de programación, Javascript posee una serie de métodos y funciones listas para ser usadas, facilitándonos así la programación de scripts complejos.

Existen métodos y funciones que nos permiten realizar cosas sencillas tan triviales como verificar si una variable es un numero, cadena o booleano y otras tan complejas como poder manejar fechas o poder realizar cálculos matemáticos avanzados.

Para aplicar una función o un método propio de Javascript a una variable, hay que colocar el nombre de la variable, un punto y a continuación el nombre de la función (con sus argumentos si los tuviera). Pej: `edad.toString()`[15]

2.1 Métodos para tipos numéricos

Para cualquier variable numérica podemos usar algunas de las siguientes funciones:

2.1.1 toString

Convierte el valor de esa variable a cadena. Aunque, a diferencia de otros lenguajes, Javascript hace de forma automática el cambio de tipo entre variables (casting), es muy aconsejable hacer uso de este tipo de funciones y no dejar que el lenguaje actúe por nosotros.

```
var numero = 7.456;
var mensaje = 'Tu nota media es: '+numero.toString();
```

15 Los lectores que conocen lenguajes de programación orientados a objetos se darán cuenta que esta forma es la notación punto propia del uso de objetos.

2.1.2 isNaN

Esta función recibe como argumento una variable y devuelve verdadero (`true`) si el valor de esa variable NO es un número válido (sea entero, negativo, decimal...). Devolverá falso (`false`) si el valor SI es un número.

```
var numero = 7.456;
var mensaje = 'Soy un texto';
var res;

res =isNaN(numero);
alert (res); //false

res =isNaN(mensaje);
alert (res); //true

res =isNaN(numero+mensaje);
alert (res); //true
```

2.2 Clase MATH

Como cualquier lenguaje maduro, Javascript nos proporciona una serie de clases predefinidas con sus respectivos métodos y propiedades que nos ayudarán a crear script de forma más rápida.

Soy consciente de que en este punto, muchos lectores aún no saben qué son las clases, los objetos, los métodos y las propiedades. No pasa nada, esos conceltos se explicarán más adelante en este manual. Es suficiente con pensar que una clase es como “una biblioteca de funciones”. Esas funciones ya están programadas (no nos interesa cómo) y sólo necesitamos saber qué hacen y como usarlas.

La clase Math se usa para hacer cálculos matemáticos. Esta clase posee también una serie de valores constantes que representan valores matemáticos significativos.

Para usar una clase en Javascript (y en otros lenguajes) es necesario colocar el nombre de dicha clase (respetando mayúsculas y minúsculas), a continuación un punto y para finalizar el nombre del valor o de la función que se va a usar (respetando mayúsculas y minúsculas). Pej: `Math.PI` o `Math.ceil(23,8)`

2.2.1 Constantes

Las constantes que se muestran a continuación representan valores matemáticos significativos.

Importante: al ser constantes, estos valores pueden consultarse y usarse en operaciones, pero NO pueden modificarse. Javascript lanzará un error si intentas hacerlo.

`Math.E`	2.718281828459045	Constante de Euler, base de los logaritmos naturales y también llamado número e
`Math.LN2`	0.6931471805599453	Logaritmo natural de 2
`Math.LN10`	2.302585092994046	Logaritmo natural de 10
`Math.LOG2E`	1.4426950408889634	Logaritmo en base 2 de Math.E
`Math.LOG10E`	0.4342944819032518	Logaritmo en base 10 de Math.E
`Math.PI`	3.141592653589793	Pi, relación entre la longitud de una circunferencia y su diámetro
`Math.SQRT1_2`	0.7071067811865476	Raíz cuadrada de 1/2
`Math.SQRT2`	1.4142135623730951	Raíz cuadrada de 2

```
var radio = 10.63;
var area = Math.PI * radio * radio;
aler('El area es: + area);
```

2.2.2 Métodos

A continuación se muestra una tabla con algunos métodos interesantes de la clase Math:

`abs`	Valor absoluto.	`Math.abs(-2)`
`sin, cos, tan`	Funciones trigonométricas, reciben el argumento en radianes .	`Math.cos(0)`
`asin, acos, atan`	Inversas de las funciones trigonométricas.	`Math.asin(1)`
`exp, log`	exponenciación y logaritmo en base E	`Math.log(Math.E)`
`ceil`	Devuelve el entero igual o más grande al argumento que se le pasa.	`Math.ceil(-2.3)`
`floor`	Devuelve el entero igual o más pequeño al argumento que se le pasa.	`Math.floor(7.6)`
`round`	Devuelve el entero más cercano igual al argumento que se le pasa como argumento	`Math.round(3.45)`
`max, min`	Máximo y mínimo de dos números.	`Math.max(4,7)`
`pow`	Potencia. El primer argumento es la base y el segundo el exponente	`Math.pow(3,2)`
`sqrt`	Raiz cuadrada del valor que se le pasa como argumento.	`Math.sqrt(25)`
`random`	Devuelve un número aleatorio entre 0 y 1	`Math.random()`

2.3 Métodos para cadenas

Javascript posee una buena cantidad de funciones[16] para el manejo de variables de tipo cadena. A continuación se van a comentar las más relevantes:

2.3.1 length

Indica el número de caracteres que componen la cadena. En realidad `length` no es una función, es una variable (propiedad) que se asocia a la cadena. Por ese motivo no se colocan paréntesis en su llamada.

```
var texto ='No se como paliar este dolor';
var total = texto.length;
aler(total); //28
```

2.3.2 toUpperCase()

Transforma todos los caracteres de la cadena a caracteres en mayúscula.

```
var texto ='No se como paliar este dolor';
var mayus = texto.toUpperCase();
aler(mayus);
```

2.3.3 toLowerCase()

Transforma todos los caracteres de la cadena a caracteres en minúscula.

```
var texto ='No se como paliar este dolor';
var minus = texto.toLowerCase();
aler(minus);
```

16 En realidad son métodos ya que las cadenas son objetos en JavaScript. Esto se verá más adelante.

2.3.4 charAt()

Esta función necesita como argumento un número entero (el cual indica la una posición) y devuelve el carácter de la cadena que se encuentra en la posición indicada. Es importante destacar que las posiciones empiezan a contarse a partir del cero (no del 1).

```
var texto ='Te quiero';
var letra = texto.charAt(0); //T
var otra = texto.charAt(4); //u
```

2.3.5 indexOf()

Esta función necesita como argumento un carácter y devuelve la posición de la primera aparición de ese carácter en la cadena. Es importante destacar que las posiciones empiezan a contarse a partir del cero (no del 1).

Si el carácter buscado no se encuentra en la cadena, se devolverá el valor -1

```
var texto ='Te quiero';
var pos1 = texto.indexOf('e'); //1 (devuelve la primera
aparición)
var pos2 = texto.charAt('q'); //3
```

2.3.6 lastIndexOf()

Esta función es totalmente análoga a la anterior pero esta vez nos va a devolver la posición de la última aparición del carácter en la cadena.

```
var texto ='Te quiero';


//devuelve la última aparición
var pos1 = texto.lastIndexOf('e'); //6
var pos2 = texto.charAt('q'); //3
```

2.3.7 substring()

Con esta función vamos a obtener subcadenas de la cadena original. Para ello debemos indicar como argumento uno o dos números enteros.

Si se indica solo un argumento (un número entero) la función devuelve la subcadena desde esa posición hasta el final:

```
var texto ='Las cadenas son muy simples';
var trozo = texto.substring(5);
alert(trozo); //'adenas son muy simples'
```

Si se le pasan dos argumentos a la función (dos números enteros separados por coma), se va a devolver la subcadena comprendida entre la posición inicial (menor de los dos números) y el carácter inmediatamente anterior a la posición final (mayor de los dos números).

```
var texto ='Las cadenas son muy simples';
var trozo = texto.substring(0,5); //da igual si pongo (5,0)
alert(trozo); //'Las c'
```

2.3.8 split()

Esta función necesita como argumento una carácter, el cual va a actuar de separador. La función va a 'partir en trozos' la cadena a partir del separador indicado y va a devolver un array cuyas celdas van a tener los trozos resultantes

```
var texto ='Las cadenas son muy simples';


var palabras = texto.split(" "); //caracter espacio en blanco
//En la variable palabras voy a tener el siguiente array:
// Las | cadenas | son | muy | simples
```

```
var letras =  texto.split(""); //No hay caracter separador
//Letras a ser un array con las letras y los espacios
//en blanco que forman el texto


var otro = texto.split("a");
//Otro sera el siguiente array:
// L | s c |den |s son muy simples
```

2.4 Temporizadores

Los temporizadores son funciones de Javascript que ejecutan acciones cada cierto tiempo. Este tipo de funciones son muy útiles para programar scripts donde haga falta la ejecución de funciones sin la supervisión del usuario (Pej: bucle de un juego, animaciones, medición de rendimiento...)

Hay que tener un poco de cuidado porque estas funciones no paran de repetir las acciones a no ser que nosotros se lo indiquemos expresamente.

2.4.1 setInterval

La función temporizador más famosa de Javascript. Necesita dos argumentos: una lista con las acciones a repetir (entre comillas) y el número de milisegundos que debe esperar entre cada repetición.

Devuelve un número único llamado manejador (handler) el cual será necesario cuando queramos 'parar' las repeticiones.

```
//cada segundo (1000 milisegundos) va a aparecer el mensaje por
consola: 'Soy un pesado'


var num = setInterval("console.log('soy un pesado');",
1000);
```

```
function mostrar(texto){
   alert(texto);
}
//cada medio segundo (500 milisegundos) va a ejecutar la
//funcion 'mostrar'
var num = setInterval("mostrar('soy muy listo')", 500);
```

2.4.2 clearInterval

Esta función se encarga de parar el flujo de repeticiones iniciado por un `setInterval`. Para ello es necesario pasarle como parámetro el manejador que devuelve la función `setInterval` que queremos 'parar':

Si queremos parar el setInterval del último ejemplo, haríamos:

```
clearInterval(num);
```

2.4.3 setTimeout

Esta función sirve para retrasar la ejecución de un bloque de acciones durante un tiempo indicado. Es decir, las acciones se realizarán una sola vez pasado el tiempo indicado en el segundo parámetro.

Tiene los mismos argumentos que `setInterval`: la lista de acciones entre comillas y el tiempo en milisegundos.

También devuelve un número único llamado manejador (handler) el cual será necesario cuando queramos 'evitar' esa cuenta atrás.

```
function irse(){
   location.href = "http://google.es";
}
console.log("Nos vamos a google en 3 segundos...");
var tiempo = setTimeout("irse()", 3000);
```

2.4.4 clearTimeout

Esta función se encarga de parar la cuenta atrás iniciada por un `setTimeout`. Para ello es necesario pasarle como parámetro el manejador que devuelve la función `setTimeout` que queremos 'parar':

Si queremos parar el setTimeout del último ejemplo, haríamos:

```
clearTimeout(tiempo);
```

2.5 Manejo de Fechas

Javascript posee una clase propia para el manejo de fechas y horas: la clase `Date`

Para trabajar con fechas/horas, lo primero que debemos hacer es crear una variable que soporte ese tipo de datos. Pej:

```
var fecha = new Date()
```

Con esta sentencias indicamos que fecha va a ser una variable de tipo `Date` y en ella se va a guardar la fecha/hora actual o partes de ella, según le indiquemos a través de parámetros.

De esta forma, al no usar parámetros cuando creamos la variable tipo `Date` (como en el ejemplo anterior), vamos a obtener la fecha y la hora actual en el siguiente formato:

```
Mon Nov 25 2013 13:40:11 GMT+0200
```

Nosotros también podemos crear fechas/horas diferentes a la actual usando parámetros:

```
Date (año, mes, día)
Date (año, mes, día, hora, minuto, segundo)
```

```
//10 de Mayo de 1999
var fecha1 = new Date(1999,4,10);


//20 de Agosto de 2008 (17:29:10)
var fecha2 = new Date(2008,7,20,17,29,10);
```

Es importante destacar que los meses se indican mediante un número que comienza en 0 (Enero) y acaba en 11 (Diciembre).

Otra forma de crear fechas (típica de otros lenguajes de programación) es colocar el número de milisegundos transcurridos desde el 1 de Enero de 1970:

```
Date (milisegundos)
```

```
//Thu Jun 01 1970 01:00:00
var fecha1 = new Date(0);


//Sat Nov 20 2286 18:46:40
Var fecha2 = new Date(10000000000000);
```

2.5.1 Funciones

Con la clase `Date` no solo podemos crear fechas. También podemos trabajar con fechas/horas ya creadas obteniendo partes de esas fechas/horas para mostrarlas o alterarlas.

Los métodos más usados de la clase `Date` son:

`getTime()`	Devuelve el número de milisegundos transcurridos desde el 1 de Junio de 1970 hasta la fecha sobre la que se está operando.
`getMonth()`	Devuelve el número de mes de la fecha (recuerda que el 0 es Enero).
`getFullYear()`	Devuelve el año de la fecha en formato de 4 cifras.
`getYear()`	Devuelve el año de la fecha en formato de 2 cifras.
`getDate()`	Devuelve el número del día del mes de la fecha.

`getDay()`	Devuelve el día de la semana indicado como número (0 para domingo, 1 para lunes... 6 para sábado)
`getHours()`, `getMinutes()`, `getSeconds()`, `getMilliseconds()`	Devuelven la hora, minutos, segundos y milisegundos (respectivamente) de la hora de la fecha.

Análogamente a estos métodos `get` existen sus homónimos `set` (`setDate`, `setHours`, `setMonth...`)

Se ha dado una visión muy reducida de la clase Date. Estos son sólo algunos de los métodos de dicha clase (los más usados). Sin embargo, la clase posee varios métodos más. Se puede obtener una visión más amplia de ella en la pagina oficial del W3C: http://www.w3schools.com/jsref/jsref_obj_date.asp