Desarrollo Web en entorno Cliente: JAVASCRIPT

UNIDAD 4. OBJETOS DEL LENGUAJE

POO-Programación orientada a objetos (repaso conceptos generales)

OBJETOS JAVASCRIPT: creación y acceso

OBJETOS INCORPORADOS EN JAVASCRIPT

4.1 CLASE/OBJETO Array
- **Propiedades**
- **Métodos**

4.2 CLASE/OBJETO String
- **Propiedades**
- **Métodos**

4.3 CLASE/OBJETO Date
- **Propiedades**
- **Métodos**

4.4 CLASE/OBJETO Math
- **Propiedades**
- **Métodos**

4.5 CLASE/OBJETO Boolean
- **Propiedades**
- **Métodos**

4.6 CLASE/OBJETO Number
- **Propiedades**
- **Métodos**

4.7 CLASE/OBJETO RegExp
- **Propiedades**
- **Métodos**

UNIDAD 4. OBJETOS DEL LENGUAJE

POO-Programación orientada a objetos (repaso conceptos generales)

Clases en POO

Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que **la definición de un objeto es la clase.** Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase.

Propiedades en clases

Las propiedades o atributos son las características de los objetos. Cuando definimos una propiedad normalmente especificamos su nombre y su tipo. Nos podemos hacer a la idea de que las propiedades son algo así como variables donde almacenamos datos relacionados con los objetos.

Métodos en las clases

Son las funcionalidades asociadas a los objetos. Cuando estamos programando las clases las llamamos métodos. Los métodos son como funciones que están asociadas a un objeto.

Objetos en POO

Los objetos son ejemplares de una clase cualquiera. Cuando creamos un ejemplar tenemos que especificar la clase a partir de la cual se creará. Esta acción de **crear un objeto a partir de una clase se llama instanciar** (que viene de una mala traducción de la palabra instance que en inglés significa ejemplar).

Para crear un objeto se tiene que escribir una instrucción especial que puede ser distinta dependiendo el lenguaje de programación que se emplee, pero será algo parecido a esto:

miCoche = new Coche()

Con la palabra new especificamos que se tiene que crear una instancia de la clase que sigue a continuación. Dentro de los paréntesis podríamos colocar parámetros con los que inicializar el objeto de la clase coche.

Estados en objetos (valor de una propiedad)

Cuando tenemos un objeto sus propiedades toman valores. Por ejemplo, cuando tenemos un coche la propiedad color tomará un valor en concreto, como por ejemplo rojo o gris metalizado. El valor concreto de una propiedad de un objeto se llama estado.

Para acceder a un estado de un objeto para ver su valor o cambiarlo se utiliza el operador punto.

miCoche.color = “rojo”

El objeto es miCoche, luego colocamos el operador punto y por último el nombre de la propiedad a la que deseamos acceder. En este ejemplo estamos cambiando el valor del estado de la propiedad del objeto a rojo con una simple asignación.

Mensajes en objetos (es la acción de llamar a un método)

Un mensaje en un objeto es la acción de efectuar una llamada a un método. Por ejemplo, cuando le decimos a un objeto coche que se ponga en marcha estamos pasándole el mensaje "ponte en marcha".

Para mandar mensajes a los objetos utilizamos el operador punto, seguido del método que deseamos invocar.

```
miCoche.ponerseEnMarcha()
```

En este ejemplo pasamos el mensaje ponerseEnMarcha(). Hay que colocar paréntesis igual que cualquier llamada a una función, dentro irían los parámetros.

Otras cosas

Hay mucho todavía que conocer de la POO ya que sólo hemos hecho referencia a las cosas más básicas. También existen mecanismos como la **herencia** y el **polimorfismo** que son unas de las posibilidades más potentes de la POO.

La **herencia** sirve para crear objetos que incorporen propiedades y métodos de otros objetos. Así podremos construir unos objetos a partir de otros sin tener que reescribirlo todo.

El **polimorfismo** sirve para definir un código que sea compatible con objetos de varios tipos. La palabra polimorfismo se refiere al hecho de tener varios métodos con el mismo nombre y la misma implementación.

OBJETOS JAVASCRIPT: creación y acceso

JavaScript tiene **objetos, y estos objetos tienen comportamiento, estado e identidad**. Así que desde este punto de vista… Es orientado a objetos.

¿Herencia en JavaScript? Vale… Antes que nada: Si hemos aprendido POO con Java, C#, C++ o algún otro lenguaje basado en clases recordemos el concepto fundamental: **No hay clases en JavaScript**.

La herencia en JavaScript es *herencia entre objetos*.

Herencia sin clases. ¡Es posible! ;)

En JavaScript la herencia es una relación definida entre objetos. **Si un objeto deriva de otro, heredará todo el comportamiento y estado definido en el objeto base.**

El mecanismo de herencia que se usa en JavaScript se conoce como **herencia por prototipo** y para resumirlo diremos básicamente que *cada objeto* tiene asociado un prototipo.

¿Polimorfimo en JavaScript?

Esta pregunta implica una forma de pensar "basada en clases" pero sí es posible simular el polimorfismo en JavaScript.

Pero es que una de las claves es que no es necesario el concepto de polimorfismo en JavaScript.

Cómo instanciar objetos / Crear nuevos objetos en javascript

En los lenguajes que soportan Programación Orientada a Objetos (POO): Instanciar un objeto es la acción de crear un ejemplar de una clase para poder trabajar con él luego. Recordamos que un objeto se crea a partir de una clase y la clase es la definición de las características y funcionalidades de un objeto. Con las clases no se trabaja, estas sólo son definiciones, para trabajar con una clase debemos tener un objeto instanciado de esa clase.

En Javascript, como en cualquier lenguaje, un objeto no es más que un conjunto de **atributos** y funciones (sus **métodos**). A cada objeto podemos añadirle nuevas propiedades y funciones de diferentes maneras que iremos viendo. Sin embargo la **herencia**, tal y como la conocemos de otros lenguajes como Java, **no existe, pues no existen las clases**: **para crear un nuevo objeto que tenga los mismos atributos que otro, estos deben ser copiados**. A esta forma de trabajar se llama **herencia basada en prototipos**. Esto significa que no hay clases o interfaces como en Java o C++: **solo hay objetos que copian (heredan) sus propiedades de otros objetos, llamados prototipos**. Así que olvidémonos de la clásica definición de "objeto como instancia de una clase", porque aquí no hay clases

En esta unidad *usaré la terminología clase/objeto para tener claro los objetos base y/o predefinidos.*

En javascript para crear un objeto a partir de una clase/objeto se utiliza la instrucción new, de esta manera.

```
var miObjeto = new miClase();
```

En una variable que llamamos miObjeto asigno un nuevo (new) ejemplar de la clase/objeto miClase. Los paréntesis se rellenan con los datos que necesite la clase/objeto para inicializar el objeto, si no hay que meter ningún parámetro los paréntesis se colocan vacíos. En realidad lo que se hace cuando se crea un objeto es llamar al constructor de esa clase/objeto y el constructor es el encargado de crearlo e inicializarlo.

Cómo acceder a propiedades y métodos de los objetos

En Javascript podemos acceder a las propiedades y métodos de objetos de forma similar a como se hace en otros lenguajes de programación, con el operador punto (".").

Las propiedades se acceden colocando el nombre del objeto seguido de un punto y el nombre de la propiedad que se desea acceder. De esta manera:

```
miObjeto.miPropiedad
```

Para llamar a los métodos utilizamos una sintaxis similar pero poniendo al final entre paréntesis los parámetros que pasamos a los métodos. Del siguiente modo:

```
miObjeto.miMetodo(parametro1,parametro2)
```

Si el método no recibe parámetros colocamos los paréntesis también, pero sin nada dentro.

```
miObjeto.miMetodo()
```

OBJETOS INCORPORADOS EN JAVASCRIPT

Vamos a introducirnos en el manejo de objetos en Javascript y para ello vamos a empezar con el estudio de **las clases/objetos predefinidos** que implementan las librerías para el trabajo con strings, matemáticas, fechas etc.

Las clases/objetos que se encuentran disponibles de manera nativa en Javascript, y que vamos a ver a continuación, son las siguientes:

- **Array,** para el trabajo con grupo de elementos
- **String,** para el trabajo con cadenas de caracteres
- **Date,** para el trabajo con fechas
- **Math,** para realizar funciones matemáticas
- **Number,** para realizar algunas cosas con números
- **Boolean,** trabajo con booleanos
- **RegExp,** que sirve para construir patrones

También la clase/objeto **Function** que ya la estudiamos en la unidad anterior.

*** **Las clases/objetos se escriben con la primera letra en mayúsculas.** Tiene que quedar claro que una clase/objeto es una especie de "declaración de características y funcionalidades" de los objetos. Dicho de otro modo, las clases/objetos son descripciones de la forma y funcionamiento de los objetos. Con las clases/objetos generalmente no se realiza ningún trabajo, sino que se hace con los objetos, que creamos a partir de las clases/objetos.

String es una clase/objeto, lo mismo que Date. Si queremos trabajar con cadenas de caracteres o fechas necesitamos crear objetos de las clases/objetos String y Date respectivamente. Como sabemos, Javascript es un lenguaje sensible a las mayúsculas y las minúsculas y en este caso, **las clases/objetos, por convención, siempre se escriben con la primera letra en mayúscula y también la primera letra de las siguientes palabras**, si es que el nombre de la clase/objeto está compuesto de varias palabras. Por ejemplo si tuviéramos la clase/objeto de "Alumnos universitarios" se escribiría con la -A- de alumnos y la -U- de universitarios en mayúscula. AlumnosUniversitarios sería el nombre de nuestra supuesta clase.

***** Uso de mayúsculas y minúsculas.** Ya que nos hemos parado anteriormente a hablar sobre el uso de mayúsculas en ciertas letras de los nombre de las clases/objetos, vamos a terminar de explicar la convención que se lleva a cabo en Javascript para nombrar a los elementos.

Para empezar, cualquier variable se suele escribir en minúsculas, aunque si este nombre de variable se compone de varias palabras, se suele escribir en mayúscula la primera letra de las siguientes palabras a la primera. Esto se hace así en cualquier tipo de variable o nombre menos en los nombres de las clases/objetos, donde se escribe también en mayúscula el primer caracter de la primera palabra.

Ejemplos:

Number, que es una clase/objeto se escribe con la primera en mayúscula.
RegExp, que es el nombre de otra clase/objeto compuesto por dos palabras, tiene la primera letras de las dos palabras en mayúscula.
miCadena, que supongamos que es un objeto de la clase/objeto String, se escribe con la primera letra en minúscula y la primera letra de la segunda palabra en mayúscula.
fecha, que supongamos que es un objeto de la clase/objeto Date, se escribe en minúsculas por ser un objeto.
miFuncion(), que es una función definida por nosotros, se acostumbra a escribir con minúscula la primera.

Como decimos, esta es la norma general para dar nombres a las variables, clases/objetos, objetos, funciones, etc. en Javascript. Si la seguimos así, no tendremos problemas a la hora de saber si un nombre en Javascript tiene o no alguna mayúscula y además todo nuestro trabajo será más claro y fácil de seguir por otros programadores o nosotros mismos en caso de retomar un código una vez pasado un tiempo.

4.1 CLASE/OBJETO Array

Esta clase/objeto nos va a dar la facilidad de construir objetos tipo arrays cuyos elementos pueden contener cualquier tipo básico, y cuya longitud se modificará de forma dinámica siempre que añadamos un nuevo elemento (y, por tanto, no tendremos que preocuparnos de esa tarea). Para poder usar un objeto array, tendremos que crear una instancia de la clase/objeto Array, por ejemplo, si escribimos:

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

tendremos creada una objeto `a` que contendrá 15 elementos (este número puede aumentar durante la ejecución de la aplicación), enumerados del 0 al 14. Para acceder a cada elemento individual usaremos la notación `a[i]`, donde i variará entre 0 y N-1, siendo N el número de elementos que le pasamos al constructor:

```
a[0]=”Primera”;
a[4]=”Ultima”;
```

Si no sabemos el número de elementos, podemos definir el array:

```
a=new Array();
```

luego introduciremos los valores. Pero es buena práctica de programación especificar el número de elementos, es más eficaz, ya que hacemos un uso más eficiente de la memoria.

También podemos inicializar el array a la vez que lo declaramos, pasando los valores que queramos directamente al constructor, por ejemplo:

```
a=new Array(21,"cadena",true);
```

que nos muestra, además, que los elementos del array no tienen por qué ser del mismo tipo.

Por tanto: si ponemos un argumento al llamar al constructor, este será el número de elementos del array (y habrá que asignarles valores posteriormente), y si ponemos más de uno, será la forma de inicializar el array con tantos elementos como argumentos reciba el constructor.

Podríamos poner como mención especial de esto lo siguiente. Las inicializaciones que vemos a continuación:

```
a = new Array("cadena");
a = new Array(false);
```

Inicializan el objeto array `a`, en el primer caso, con un elemento cuyo contenido es la cadena 'cadena', y en el segundo caso con un elemento cuyo contenido es false.

Lo comentado anteriormente sobre inicialización de arrays con varios valores, significa que si escribimos

```
a=new Array(2,3);
```

NO vamos a tener un array con 2 filas y 3 columnas, sino un array cuyo primer elemento será el 2 y cuyo segundo elemento será el 3. Entonces, ¿cómo creamos un array bidimensional? (un array bidimensional es una construcción bastante frecuente). Creando un array con las filas deseadas y, después, cada elemento del array se inicializará con un array con las columnas deseadas. Por ejemplo, si queremos crear un array con 4 filas y 7 columnas, bastará escribir:

```
a = new Array(4);
for(i = 0; i < 4; i++){
        a[i] = new Array(7);
}
```

y para referenciar al elemento que ocupa la posición (i,j), escribiremos a[i][j];

```
a[1][3]=”Hola”;
```

Ejemplo: **4.23-ArrayMultidimensional.HTML**

Modernamente también se usa un formato de inicialización de arrays que usa el corchete, como ya vimos en el capítulo anterior.

```
var valores = [5, 2, 11, -7, 1];
```

Ejemplo: **4.12-arrays_bucles.HTML** (recorrer un array)

Vistas las peculiaridades de la creación de arrays, vamos ahora a estudiar sus propiedades y sus métodos.

Propiedades de Array

Propiedad	Significado
length	Devuelve el número de elementos del array.
prototype	Permite añadir nuevos atributos al objeto array, o sea, nuevas propiedades y métodos

1. **lenght:**

```
for (var i=0; i<codigos_productos.length; i++){
     document.write(codigos_productos[i] + “<br>”);
}
```

2. **prototype:**

```
Array.prototype.nueva_propiedad = valor;
Array.prototype.nuevo_metodo= nombre_de_la_funcion;
```

EJ.: si queremos crear una propiedad llamada *dominio* y establecerle el valor *.com*. Luego podemos cambiar el valor de esta propiedad en el momento que creemos nuevos arrays:

```
var paginas_comerciales=new Array();
Array.prototype.dominio=”.com”

var paginas_gubernamentales=new Array();
```

```
paginas_gubernamentales.dominio=".gov"

document.write("Extensión de las páginas comerciales: ",
paginas_comerciales.dominio);
document.write("Extensión de las páginas gubernamentales: ",
paginas_gubernamentales.dominio);
```

Métodos de Array

Método	Significado
Nombre_array.concat(array) Nombre_array.concat(valor1, valor2, …)	Devuelve el array resultado de la concatenación del valor del objeto array con el array que se le pasa como parámetro, o con los elementos que se le pasan como parámetros
Nombre_array.join([separador])	Genera un string que contiene los substrings mantenidos por cada elemento del array unidos por el ó los caracteres indicados en el parámetro **separador**. Si no se utiliza el parámetro, este método utiliza la coma (,) como separador. Concatena los elementos de un array en una sola cadena separada por un carácter opcional
Nombre_array.pop()	Elimina y devuelve el último elemento del array. Este método modifica la longitud del array (atributo **length**).
Nombre_array.push(elem1, ... elemN)	Añade al final del array los elementos que se pasan como parámetro y devuelve la nueva longitud de éste.
Nombre_array.reverse()	Devuelve un array igual que el original en el que se ha invertido el orden de los elementos.
Nombre_array.shift()	Elimina y devuelve el primer elemento del array. Este método modifica la longitud del array (atributo **length**).
Nombre_array.slice(índice_inicio,[índice_fin])	Devuelve un array formado por los elementos del array que se encuentran entre las posiciones marcadas por **índice_inicio** y **índice_fin - 1**.
Nombre_array.splice(inicio, [número_elementos_a_borrar], [elem1, ... elemN])	Elimina elementos de un array y, si es necesario, inserta nuevos elementos en su lugar, devolviendo los elementos eliminados
Nombre_array.sort([función])	Ordena los elementos del array. Si no se utiliza el parámetro, este método ordena los elementos del array de menor a mayor comparando cada elemento como si se tratase de una string. Si se pasa como parámetro una función de ordenación, este método ordena los elementos del array de acuerdo con el criterio de ordenación de la función.
Nombre_array.toString()	Devuelve un string que representa el objeto array.
Nombre_array.unshift(elem1, ... elemN)	Añade uno o más elementos al comienzo del array. Netscape devuelve la nueva longitud del array y Explorer el array modificado

Ejemplos:

- **Método concat():** **4.14-concat.html**
- **Método join():** **4.15-join.html**
- **Método pop():** **4.19-pop.html**
- **Método push():** **4.13-push.html**
- **Método reverse():** **4.16-reverse.html**
- **Método shift():** **4.18-shift.html**
- **Método slice():** **4.20-slice.html**
- **Método splice():** **4.22-splice.html**
- **Método sort():** **4.21-sort.html**
- **Método toString():** **toString.html**
- **Método unshift():** **4.17-unshift.html**

Ejercicios para alumnado: Ejecutar los siguientes programas y ver lo que hacen

concat.html join.html push-pop.html slice.html sort.html

splice.html unshift-shift.html

4.2 CLASE/OBJETO String

En javascript las variables de tipo texto son objetos de la clase/objeto String. Esto quiere decir que cada una de las variables que creamos de tipo texto tienen una serie de propiedades y métodos. Recordamos que las propiedades son las características, como por ejemplo longitud en caracteres del string y los métodos son funcionalidades, como pueden ser extraer un substring o poner el texto en mayúsculas.

Para crear un objeto de la clase/objeto String lo único que hay que hacer es asignar un texto a una variable. El texto va entre comillas, como ya hemos visto en los capítulos de sintaxis. También se puede crear un objeto string con el operador new, que veremos más adelante. La única diferencia es que en versiones de Javascript 1.0 no funcionará new para crear los Strings.

Propiedades de String

Propiedad	Significado
length	Nos indica la longitud del string.
prototype	Permite añadir nuevos atributos al objeto string.

Métodos de String

Método	Significado
anchor(nombre_ancla)	Crea un enlace (local) donde el atributo **name** toma el valor del parámetro y se asigna como texto del enlace el que tenga el objeto string. Genera el código HTML: **valor del objeto** Devuelve una cadena convertida en un ancla de HTML.
big()	Devuelve el valor del objeto string al que se le ha aplicado una fuente grande. Genera el código HTML: **<big>valor del objeto</big>** Aumenta tamaño cadena.
blink()	Devuelve el valor del objeto string al que se le ha aplicado el estilo de texto **parpadeo**. Genera el código HTML: **<blink>valor del objeto</blink>**. La etiqueta <blink> está obsoleta en **HTML 4.01** Crea efecto de una cadena parpadeando.
bold()	Devuelve el valor del objeto string al que se le ha aplicado el **estilo negrita**. Genera el código HTML: **valor del objeto**.
charAt(indice)	Devuelve el carácter de string que se encuentra en la posición indicada en el parámetro (el primer carácter del string ocupa la posición 0). Permite acceder a un carácter en concreto de una cadena.
charCodeAt(indice)	Devuelve un valor entero que representa la codificación ISO-Latin-1 del carácter que se encuentra en la posición indicada en el parámetro (el primer carácter del string ocupa la posición 0).
concat(string)	Concatena el valor del objeto string con el string que se pasa como parámetro. El

	string resultado de la concatenación es equivalente a la aplicación del operador + sobre dos objetos de tipo string.
fixed()	Devuelve el valor del objeto string al que se le ha aplicado una fuente con formato monoespacio. Genera el código HTML: **<tt>valor del objeto</tt>**.
fontcolor(color)	Cambia el color con el que se muestra el string.
fontsize(tamaño)	Cambia el tamaño con el que se muestra el string. Modifica el tamaño de la fuente de la cadena.
fromCharCode(cod1, ... ,codN)	Devuelve el string compuesto por los caracteres asociados a los código ISO-Latin-1 que se pasan como parámetro.
indexOf(patron,indice)	Devuelve la posición dentro del string en la que se encuentra la primera ocurrencia del patrón que se pasa como primer parámetro, a partir de la posición indicada en el segundo parámetro. Si no se especifica el segundo parámetro, este método comienza la búsqueda a partir de la posición 0 del string. Si no se encuentra el patrón devuelve -1.
italics()	Devuelve el valor del objeto string al que se le ha aplicado el **estilo cursiva**. Genera el código HTML: **valor del objeto**.
lastindexOf(patron,indice)	Devuelve la posición dentro del string en la que se encuentra la última ocurrencia del patrón que se pasa como primer parámetro, buscando hacia el principio del string a partir de la posición indicada en el segundo parámetro. Si no se especifica el segundo parámetro, este método comienza la búsqueda a partir de la posición última del string (String.length-1). Si no se encuentra el patrón devuelve -1. Devuelve la posición de la última ocurrencia del carácter pasado como parámetro.
link(url)	Crea un enlace donde el atributo **href** toma el valor del parámetro y se asigna como texto del enlace el que tenga el objeto string. Genera el código HTML: **valor del objeto**
match(expr_reg)	Busca en la string el patrón indicado en la expresión regular que se pasa como parámetro. Devuelve un array que contiene el resultado de la búsqueda. Busca una coincidencia en una cadena y devuelve todas las coincidencias encontradas.
replace(expr_reg,nuevo_texto)	Devuelve un string cuyo contenido es el del string al que se le aplica este método, en el que se han sustituido las cadenas de caracteres que coinciden con la expresión regular que se pasa como primer parámetro por el substring indicado en el segundo parámetro.
search(expr_reg)	Devuelve la posición dentro del string en la que se encuentra el primer substring que coincida con la expresión regular que se pasa como parámetro. si no se encuentra nungún substring devuelve -1.
slice(inicio,fin)	Devuelve un string formado por los caracteres del string que se encuentra entre las posiciones marcadas por **inicio** y **fin - 1**.
small()	Devuelve el valor del objeto string al que se le ha aplicado una fuente pequeña. Genera el código HTML: **valor del objeto**.
split(separador)	Divide el string en un array de substrings, de forma que, el valor que se usa como parámetro determina el carácter a utilizar para dividir el string original. Si no se encuentra el carácter separador, este método devuelve un array con un único elemento que contiene la cadena original.

	Corta un cadena en base a un separador que pasamos como parámetro.
strike()	Devuelve el valor del objeto string al que se le ha aplicado el **estilo tachado**. Genera el código HTML: **<strike>valor del objeto</strike>**. La etiqueta <strike> está obsoleta en **HTML 4.01**
sub()	Devuelve el valor del objeto string al que se le ha aplicado una **fuente de subíndice**. Genera el código HTML: **_{valor del objeto}**.
substr(inicio,longitud)	Devuelve un string formado por los caracteres del string que se encuentra desde la posición marcada por **inicio** hasta la posición marcada por **inicio + longitud -1**.
substring(indice1,indice2)	Devuelve la cadena de caracteres formada por los caracteres del string que se encuentran entre la posición indicada en el primer parámetro y una menos de la indicada en el segundo parámetro. Si el primer parámetro es menor que 0, éste se trata como si fuese 0. si el segundo parámetro es mayor que la longitud del string éste se trata como si fuese la longitud del mismo. Si el primer parámetro es igual al segundo, este método devuelve el string vacío. Si no se utiliza el segundo parámetro, se extrae el substring
sup()	Devuelve el valor del objeto string al que se le ha aplicado una **fuente de superíndice**. Genera el código HTML: **^{valor del objeto}**.
toLowerCase()	Devuelve el valor del objeto string con todos los caracteres en **minúscula**
toUpperCase()	Devuelve el valor del objeto string con todos los caracteres en **mayúscula**

Ejemplo (Manipular una cadena de caracteres): String.html

Ejemplo (script donde escribimos el contenido de un string con un carácter separador ("-") entre cada uno de los caracteres del string)

String1.html

Ejemplo (script que rompe un string en dos mitades y las imprima por pantalla. Las mitades serán iguales, siempre que el string tenga un número de caracteres par. En caso de que el número de caracteres sea impar no se podrá hacer la mitad exacta, pero partiremos el string lo más aproximado a la mitad)

String2.html

Ejercicio alumnado: Realizar uno o varios programa/s en el/los que se pruebe los siguientes métodos del objeto string:

concat(string)	indexOf(patron,indice)	match(expr_reg)
replace(expr_reg,nuevo_texto)	search(expr_reg)	slice(inicio,fin)
split(separador)	substr(inicio,longitud)	substring(indice1,indice2)

4.3 CLASE/OBJETO Date

Esta clase/objeto nos va a permitir hacer manipulaciones con fechas: poner fechas, consultarlas... para ello, debemos saber lo siguiente: JS maneja fechas en milisegundos.

Todos los métodos de la clase/objeto Date dependerán de la fecha y hora que tenga el equipo del visitante.

Los meses de Enero a Diciembre vienen dados por un entero cuyo rango varía entre el 0 y el 11 (es decir, el mes 0 es Enero, el mes 1 es Febrero, y así sucesivamente), los días de la semana de Domingo a Sábado vienen dados por un entero cuyo rango varía entre 0 y 6 (el día 0 es el Domingo, el día 1 es el Lunes, ...), los años se ponen tal cual, y las horas se especifican con el formato HH:MM:SS.

Podemos crear un objeto Date vacío, o podemos crearlo dándole una fecha concreta. Si no le damos una fecha concreta, se creará con la fecha correspondiente al momento actual en el que se crea.

Para crearlo dándole un valor, tenemos estas posibilidades:

```
var Mi_Fecha = new Date(año, mes);
var Mi_Fecha = new Date(año, mes, día);
var Mi_Fecha = new Date(año, mes, día, horas);
var Mi_Fecha = new Date(año, mes, día, horas, minutos);
var Mi_Fecha = new Date(año, mes, día, horas, minutos, segundos);
```

Tiene una serie de métodos que podemos dividirlos en tres subconjuntos:

Métodos de lectura: comienzan con el prefijo ***get***. Principal función **consultar** las diferentes parte de una instancia del objeto Date.
Métodos de escritura: comienzan con el prefijo ***set***. Sirven para **inicializar** las diferentes parte de una instancia del objeto Date.
Métodos de conversión: comienzan con el prefijo ***to***. Convierten objetos del tipo Date en cadenas de texto o en milisegundos según los diferentes criterios que veremos a continuación.

Propiedades de Date

Propiedad	Significado
prototype	Permite añadir nuevos atributos al objeto Date.

Métodos de Date

Método	Significado
getDate()	Devuelve el día del mes del objeto Date (valor entero entre 1 y 31)
getDay()	Devuelve el día del mes de la semana del objeto Date (valor entero entre 0 y 6). al domingo le corresponde el valor 0, al lunes el 1, etc.
getHours()	Devuelve la hora del objeto Date (valor entero entre 0 y 23 que indica el número de

	horas transcurridas desde la media noche).
getMilliSeconds()	Devuelve los milisegundos del objeto Date (valor entero entre 0 y 999).
getMinutes()	Devuelve los minutos del objeto Date (valor entero entre 0 y 59).
getMonth()	Devuelve el mes del objeto Date (valor entero entre 0 y 11). A enero le corresponde el valor 0, ... etc.
getSeconds()	Devuelve el número de segundos del objeto Date (valor entero entre 0 y 59).
getTime()	Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 horas hasta la hora del objeto Date.
getTimezoneOffset()	Devuelve los minutos de diferencia entre la hora del objeto Date y la hora Universal coordinada (UTC), antes llamada la hora en el meridiano de Greenwich (GMT).
getYear()	Devuelve el año del objeto Date. Si el año se encuentra entre 1900 y 1999, este método devuelve un entero de dos dígitos (la diferencia entre el año y el valor 1900). Si el año está fuera de este rango, Explorer devuelve el valor del año expresado en cuatro dígitos, mientras que Netscape sigue devolviendo el año expresado como la diferencia entre este y el valor 1900. Está considerado **obsoleto**
getFullYear()	Devuelve el año del objeto Date con cuatro dígitos.
parse(fecha)	Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 horas hasta la fecha indicada en el parámetro (de tipo string).
getUTCDate()	Devuelve el día del mes del objeto Date de acuerdo al horario universal (valor entero entre 1 y 31)
getUTCDay()	Devuelve el día del mes de la semana del objeto Date de acuerdo al horario universal (valor entero entre 0 y 6). al domingo le corresponde el valor 0, al lunes el 1, etc.
getUTCFullYear()	Devuelve el año del objeto Date de acuerdo al horario universal con cuatro dígitos.
getUTCHours()	Devuelve la hora del objeto Date de acuerdo al horario universal (valor entero entre 0 y 23 que indica el número de horas transcurridas desde la media noche).
getUTCMilliSeconds()	Devuelve los milisegundos del objeto Date de acuerdo al horario universal (valor entero entre 0 y 999).
getUTCMinutes()	Devuelve los minutos del objeto Date de acuerdo al horario universal (valor entero entre 0 y 59).
getUTCMonth()	Devuelve el mes del objeto Date de acuerdo al horario universal (valor entero entre 0 y 11). A enero le corresponde el valor 0, ... etc.
getUTCSeconds()	Devuelve el número de segundos del objeto Date de acuerdo al horario universal (valor entero entre 0 y 59).
setDate(dia)	Establece el día del mes del objeto Date (valor entero entre 1 y 31).
setFullYear(año)	Establece el año del objeto Date con cuatro dígitos.
setHours(hora)	Establece la hora del objeto Date (valor entero entre 0 y 23).
setMilliseconds(segundos)	Establece el número de milisegundos del objeto Date
setMinutes(minutos)	Establece los minutos del objeto Date (valor entero entre 0 y 59).
setMonth(mes)	Establece el mes del objeto Date (valor entero entre 0 y 11). A enero le corresponde el valor 0, ... etc.

setSeconds(segundos)	Establece el número de segundos del objeto Date (valor entero entre 0 y 59).
setUTCDate(dia)	Establece el día del mes del objeto Date de acuerdo al horario universal (valor entero entre 1 y 31).
setUTCFullYear(año)	Establece el año del objeto Date de acuerdo al horario universal con cuatro dígitos.
setUTCHours(hora)	Establece la hora del objeto Date de acuerdo al horario universal (valor entero entre 0 y 23).
setUTCMilliseconds(segundos)	Establece el número de milisegundos del objeto Date de acuerdo al horario universal
setUTCMinutes(minutos)	Establece los minutos del objeto Date de acuerdo al horario universal (valor entero entre 0 y 59).
setUTCMonth(mes)	Establece el mes del objeto Date de acuerdo al horario universal (valor entero entre 0 y 11). A enero le corresponde el valor 0, ... etc.
setUTCSeconds(segundos)	Establece el número de segundos del objeto Date de acuerdo al horario universal (valor entero entre 0 y 59).
setTime(milisegundos)	Establece el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 horas.
setYear(año)	Establece el año del objeto Date. Si en el año se indica un número entre 0 y 99, este método asigna como año ese valor más 1900. si el año que se indica está fuera de este rango, este método asigna ese valor tal cual.
toDateString()	Convierte la parte de fecha del objeto Date en un string.
toGMTString()	Convierte la fecha del objeto Date en un string usando la convención de zona horaria del meridiano de Greenwich (GMT).
toLocaleDateString()	Convierte la parte de fecha del objeto Date en un string usando la convención de zona horaria local.
toLocaleTimeString()	Convierte la parte de hora del objeto Date en un string usando la convención de zona horaria local.
toLocaleString()	Convierte la fecha del objeto Date en un string usando la convención de zona horaria local.
toString()	Convierte la fecha del objeto Date en un string.
toTimeString()	Convierte la parte de hora del objeto Date en un string.
toUTCString()	Convierte la fecha del objeto Date en un string usando la convención de horario universal coordinado.
UTC(año,mes,dia,horas,min,sg)	Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 horas en la hora Universal coordinada (UTC), antes llamada la hora en el meridiano de Greenwich (GMT), hasta la fecha pasada como parámetro.

Ejemplo (Manipulación de propiedades y métodos del objeto Date**)**:

Fecha.html 3.1-ObjetoDate.html

Ejemplo: Determinación de la edad de una persona EdadUsuario.html

```
<script language="javascript">
  <!--
var edad_usuario

// Recopilar la fecha de nacimiento dia, mes y año
var anno_usuario = prompt("Introduzca el año de su nacimiento:", 1950)
var mes_usuario = prompt("Introduzca el mes de su nacimiento" +
                         " (1 para enero, etc.)", 1)
var dia_usuario = prompt("Introduzca el dia de su nacimiento:", 1)

// Crear un objeto Date y cambiar su valor
// con la fecha del cumpleaños del usuario en este año
var fecha_cumple = new Date()
fecha_cumple.setMonth(mes_usuario - 1)
fecha_cumple.setDate(dia_usuario)

// Cargar la fecha actual
var fecha_actual = new Date()
var anno_actual = fecha_actual.getYear()

if (fecha_actual < fecha_cumple) {
    edad_usuario = anno_actual - anno_usuario - 1
}
else {
    edad_usuario = anno_actual - anno_usuario
}

alert("Ud. tiene " + edad_usuario + " años.")
  -->
</script>
```

Ejemplo: Cálculo de una edad en semanas, días, etc. EdadUsuarioFracciones.html

```
<script language="javascript">
  <!--
// Declarar e inicializar las constantes de los milisegundos
var UN_SEGUNDO = 1000
var UN_MINUTO = UN_SEGUNDO * 60
var UNA_HORA = UN_MINUTO * 60
var UN_DIA = UNA_HORA * 24
var UNA_SEMANA = UN_DIA * 7

// Recopilar la fecha de nacimiento: día, mes y año
var dia_usuario = prompt("Introduzca el día de su nacimiento:", 1)
var mes_usuario = prompt("Introduzca el mes en el que nació" +
                         " (1 para enero, etc.)",1)
var anno_usuario = prompt("Indique el año de su nacimiento:", 1950)

// Crear un objeto Date y usar los datos antes
// solicitados para cambiar la fecha
var fecha_usuario = new Date()
```

```
fecha_usuario.setFullYear(anno_usuario)
fecha_usuario.setMonth(mes_usuario - 1)
fecha_usuario.setDate(dia_usuario)

// Almacenar la fecha y hora actuales
var fecha_actual = new Date()

// Convertir ambas fechas en milisegundos
var ms_usuario = fecha_usuario.getTime()
var ms_actual = fecha_actual.getTime()

// Calcular la edad en varias unidades
var ms_edad = ms_actual - ms_usuario
var segundos_edad = Math.round(ms_edad/UN_SEGUNDO)
var minutos_edad = Math.round(ms_edad/UN_MINUTO)
var horas_edad = Math.round(ms_edad/UNA_HORA)
var dias_edad = Math.round(ms_edad/UN_DIA)
var semanas_edad = Math.round(ms_edad/UNA_SEMANA)

// Mostrar el mensaje
alert("Ésta es su edad en diferentes unidades:\n\n" +
      segundos_edad + " segundos\n" +
      minutos_edad + " minutos\n" +
      horas_edad + " horas\n" +
      dias_edad + " días\n" +
      semanas_edad + " semanas")
  -->
</script>
```

Ejemplo: Cálculo de los días entre dos fechas. DiferenciaFechasDias.html

```
<script language="javascript">
  <!--
function dias_entre(fecha1, fecha2) {

    // número de milisegundos en un día
    var UN_DIA = 1000 * 60 * 60 * 24

    // Convertir las fechas a milisegundos
    var fecha1_ms = fecha1.getTime()
    var fecha2_ms = fecha2.getTime()

    // Calcular la diferencia en milisegundos
    var diferencia_ms = Math.abs(fecha1_ms - fecha2_ms)

    // Convertir el tiempo a días y devolver
    return Math.round(diferencia_ms/UN_DIA)
}
  -->
</script>
```

Ejemplo: Conseguir varios formatos de fecha. VariosFormatosFecha.html

Partes de código del ejemplo:

```
<script>

var meses = new Array
("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto","Septiembre","Octubre","Nov
iembre","Diciembre");

var diasSemana = new Array
("Domingo","Lunes","Martes","Miércoles","Jueves","Viernes","Sábado");

var f=new Date();

document.write(diasSemana[f.getDay()] + ", " + f.getDate() + " de " + meses[f.getMonth()] + "
de " + f.getFullYear());

</script>



<script>

var meses = new Array
("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto","Septiembre","Octubre","Nov
iembre","Diciembre");

var f=new Date();

document.write(f.getDate() + " de " + meses[f.getMonth()] + " de " + f.getFullYear());

</script>
```

4.4 CLASE/OBJETO Math

Esta clase/objeto se utiliza para poder realizar cálculos en nuestros scripts. Tiene la peculiaridad de que sus propiedades no pueden modificarse, sólo consultarse. Estas **propiedades** son constantes matemáticas de uso frecuente en algunas tareas, por ello es lógico que sólo pueda consultarse su valor pero no modificarlo. Sus **métodos** permiten realizar operaciones matemáticas, como raíces cuadradas, potencias y redondeos, así como trigonométricas, logarítmicas, etc.

De modo que para cualquier cálculo matemático complejo utilizaremos la clase/objeto Math, con una particularidad. Hasta ahora cada vez que queríamos hacer algo con una clase/objeto debíamos de crear un objeto de esa clase/objeto y trabajar con el objeto y **en el caso de la clase/objeto Math se trabaja directamente con la clase/objeto**.

Dicho de otra forma, para trabajar con la clase/objeto Math no deberemos utilizar la instrucción new y utilizaremos el nombre de la clase/objeto para acceder a sus propiedades y métodos.

Propiedades de Math

Propiedad	Significado
E	Constante de Euler o número **e** (se utiliza como base de los logaritmos neperianos y tiene un valor aproximado de 2,718).
LN2	Logaritmo neperiano de 2 (aproximadamente 0,693).
LN10	Logaritmo neperiano de 10 (aproximadamente 2,302).
LOG2E	Logaritmo en base 2 de **e** (aproximadamente 1,442).
LOG10E	Logaritmo en base 10 de **e** (aproximadamente 0,434).
PI	Constante **pi** (aproximadamente 3,14159).
SQRT1_2	Raiz cuadrada de 1/2 (aproximadamente 0,707).
SQRT2	Raiz cuadrada de 2 (aproximadamente 1,414).

Métodos de Math

Método	Significado
abs(numero)	Devuelve el valor absoluto del número.
acos(numero)	Devuelve el arcocoseno del número.
asin(numero)	Devuelve el arcoseno del número.
atan(numero)	Devuelve la arcotangente del número.
atan2(x,y)	Devuelve la arcotangente del cociente de los dos argumentos, que representa el ángulo del punto (x,y) respecto a la abscisa x positiva.
ceil(numero)	Redondea el número al valor entero inmediatamente superior.

cos(numero)	Devuelve el coseno del número.
exp(numero)	Devuelve el valor **e** elevado a **numero**.
floor(numero)	Redondea el número al valor entero inmediatamente inferior.
log(numero)	Devuelve el logaritmo natural del número.
max(n1,n2)	Devuelve el valor mayor de los dos pasados como parámetro.
min(n1,n2)	Devuelve el valor menor de los dos pasados como parámetro.
pow(base,exponente)	Devuelve el valor **base** elevado a **exponente**.
random()	Devuelve un número pseudoaleatorio entre 0 y 1.
round(numero)	Redondea el número al valor entero más cercano.
sin(numero)	Devuelve el seno del número.
sqrt(numero)	Devuelve la raiz cuadrada del número.
tan(numero)	Devuelve la tangente del número.

Ejemplo: Redondeo de una cantidad a un número de decimales especificados.
RedondearDecimales.html

```
<script language="javascript">
  <!--
function redondeo_decimales(numero, decimales) {
    var resultado1 = numero * Math.pow(10, decimales)
    var resultado2 = Math.round(resultado1)
    var resultado3 = resultado2 / Math.pow(10, decimales)
    return (resultado3)
}
  -->
</script>
```

Otro Ejemplo (dado el radio calcular el círculo) **3.2-ObjetoMath.html**

4.5 CLASE/OBJETO Boolean

Esta clase/objeto nos permite crear booleanos, esto es, el tipo de dato que es cierto o falso, tomando los valores true o false. Podemos crear objetos de este tipo mediante su constructor, el siguiente código crea un objeto Boolean llamado myBoolean:

```
var myBoolean=new Boolean();
```

Dependiendo de lo que reciba el constructor de la clase/objeto Boolean el valor del objeto booleano que se crea será verdadero o falso, de la siguiente manera

- **Se inicializa a false** cuando no pasas ningún valor al constructor, o si pasas

  - 0
  - -0
  - ""
  - false
  - undefinied
  - NaN

- **Se inicializa a true** cuando recibe cualquier valor entrecomillado (incluso con la cadena "false") o cualquier número distinto de 0 o true.

Veamos varios ejemplos:

```
a = new Boolean(); asigna a 'a' el valor 'false'
a = new Boolean(0); asigna a 'a' el valor 'false'
a = new Boolean(""); asigna a 'a' el valor 'false'
a = new Boolean(false); asigna a 'a' el valor 'false'
a = new Boolean(numero_distinto_de_0); asigna a 'a' el valor 'true'
a = new Boolean(true); asigna a 'a' el valor 'true'
```

Ejemplo anterior: **comprobar_valor_boolean.html**

Propiedades de Boolean

Propiedad	Significado
prototype	Permite añadir nuevos atributos al objeto Boolean.

Métodos de Boolean

Método	Significado
toString()	Devuelve una string que representa al objeto Boolean. Si el valor del objeto boolean es **true**, ese método devuelve la string "**true**" y si es **false** la string "**false**".

Como ejemplo de aplicación de objetos booleanos, vamos a recordar el ejemplo de cálculo de una combinación de lotería primitiva, que estudiamos en el capítulo de estructuras de control.

```
<script language="javascript">
  <!--
     var primitiva=new Array(6);

     function GenerarNumeros()
       {
      var i, j, k;
      var intnumero;
      var blnrepetido=new Boolean();
       document.getElementById("txtNumeros").value="";

      for(i=0;i<6;i++)
        {
          do
            {
              blnrepetido = false;
              intnumero = Math.floor(Math.random()*49)+1;
              j = 0;
              while( (j < i) && (!blnrepetido))
                 {
                  if( primitiva[j] == intnumero)
                      blnrepetido = true;
                  j ++;
                  }
            }while(blnrepetido);
          primitiva[i] = intnumero;
        }
      primitiva.sort(Comparar);
      for(i=0;i<6;i++)
        document.getElementById("txtNumeros").value+=primitiva[i]+"  ";
    }


     function Comparar(a,b)
       {
       return a-b;
       }
  -->
</script>
```

4.6 CLASE/OBJETO Number

Esta clase/objeto representa el tipo de dato número con el que JS trabaja. Podemos asignar a una variable un número, o podemos darle valor, invocando al constructor de la clase/objeto Number, de esta forma:

var a = new Number(valor);

El valor del objeto Number que se crea depende de lo que reciba el constructor de la clase/objeto Number. Con estas reglas:

- Si el constructor recibe un número, entonces inicializa el objeto con el número que recibe. Si recibe un número entrecomillado lo convierte a valor numérico, devolviendo también dicho número.
- Devuelve 0 en caso de que no reciba nada.
- En caso de que reciba un valor no numérico devuelve NaN, que significa "Not a Number" (No es un número)
- Si recibe false se inicializa a 0 y si recibe true se inicializa a 1.

Esta clase/objeto cuenta con varias propiedades, sólo accesibles desde Number. Esto quiere decir que no podemos escribir a = new Number(); alert(a.MAX_VALUE); sino que tenemos que hacerlo directamente sobre el objeto Number, es decir, tendremos que consultar los valores que hay en Number.MAX_VALUE, Number.MIN_VALUE, etc.

Propiedades de Number

Propiedad	Significado
MAX_VALUE	Mayor número representable.
MIN_VALUE	Menor número representable.
NaN	Valor especial que representa que un valor no es un número (**Not a Number**).
NEGATIVE_INFINITY	Valor negativo infinito devuelto cuando se produce un desbordamiento (overflow).
POSITIVE_INFINITY	Valor positivo infinito devuelto cuando se produce un desbordamiento (overflow)
prototype	Permite añadir nuevos atributos al objeto Number.

Ejemplo de utilización de estas propiedades: Number1.html

Métodos de Number

Método	Significado
toExponential(n)	Convierte un número a su notación exponencial, indicando el número de cifras significativas
toFixed(n)	Formato de número con n cifras decimales
toPrecision(n)	Formato de número con n cifras en total
toString(base)	Devuelve un string que representa al objeto Number. La base es un valor entero entre 2 y 16 y es opcional (por defecto se utiliza base 10).

Como ejemplo de aplicación del objeto Number, vamos a comprobar todos sus métodos.

```
var num = new Number(13.3714);
document.write("var num = new Number(13.3714);<br />");
document.write("toExponential<br />");
document.write("toExponential() "+num.toExponential()+"<br />");
document.write("toExponential(2) "+num.toExponential(2)+"<br />");
document.write("toExponential(3) "+num.toExponential(3)+"<br />");
document.write("toExponential(10) "+num.toExponential(10)+"<br />");
document.write("-------------------------------------------<br />");
document.write("toFixed<br />");
document.write("toFixed() "+num.toFixed()+"<br />");
document.write("toFixed(2) "+num.toFixed(2)+"<br />");
document.write("toFixed(3) "+num.toFixed(3)+"<br />");
document.write("toFixed(10) "+num.toFixed(10)+"<br />");
document.write("-------------------------------------------<br />");
document.write("toPrecision<br />");
document.write("toPrecision() "+num.toPrecision()+"<br />");
document.write("toPrecision(2) "+num.toPrecision(2)+"<br />");
document.write("toPrecision(3) "+num.toPrecision(3)+"<br />");
document.write("toPrecision(10) "+num.toPrecision(10));
```

Ejemplo anterior: Number.html

4.7 CLASE/OBJETO RegExp

Con esta clase/objeto se pueden crear objetos para almacenar una expresión regular (Regular Expression), patrones de búsqueda que permiten buscar coincidencias dentro de cadenas de texto muy utilizadas en muchos lenguajes de programación como Perl, comandos del shell, PHP, etc.

Una expresión regular es un objeto que describe un patrón de caracteres.
Las expresiones regulares se utilizan para realizar reconocimiento de patrones y "buscar y reemplazar" las funciones de texto.

Sintaxis:

```
var patt=new RegExp(patrón,modificador);

      o más simple:

var  patt=/patrón/modificador;
```

patrón especifica el patrón de una expresión
modificadores especificar si la búsqueda debe ser global, mayúsculas y minúsculas, etc

Modificadores

Los modificadores se utilizan para realizar búsquedas entre mayúsculas y minúsculas (case-sensitive) y búsquedas globales:

Modificador	Descripción
i	Buscar las mayúsculas y minúsculas
g	Realizar una comparación global (todos los elementos en vez de parar después de la primera coincidencia)
m	Buscar coincidencia en varias líneas

Ejemplo: **expresiónReg_i_g.html**

Recordar estos métodos de la clase String

match(expr_reg)	Busca en la string el patrón indicado en la expresión regular que se pasa como parámetro. Devuelve un array que contiene el resultado de la búsqueda. Busca una coincidencia en una cadena y devuelve todas las coincidencias encontradas.
replace(expr_reg,nuevo_texto)	Devuelve un string cuyo contenido es el del string al que se le aplica este método, en el que se han sustituido las cadenas de caracteres que coinciden con la expresión regular que se pasa como primer parámetro por el substring indicado en el segundo parámetro.
search(expr_reg)	Devuelve la posición dentro del string en la que se encuentra el primer substring que coincida con la expresión regular que se pasa como parámetro. si no se encuentra ningún substring devuelve -1.

Soportes (corchetes)

Los corchetes se utilizan para encontrar un rango de caracteres:

Expresión	Descripción
[abc]	Encuentra cualquier caracter entre los corchetes
[^abc]	Encuentra cualquier caracter que no esté entre los corchetes
[0-9]	Encuentra cualquier dígito 0 a 9
[A-Z]	Encuentra cualquier caracter de mayúsculas A a Z mayúscula
[a-z]	Busca cualquier caracter de minúscula a minúsculas z
[A-z]	Encuentra cualquier caracter de mayúscula a minúscula z
[adgk]	Busca cualquier caracter en el conjunto dado
[^ adgk]	Encuentra cualquier caracter fuera del conjunto dado
(Rojo|azul|verde)	Buscar cualquiera de las alternativas especificadas

Ejemplo: **expresiónReg2.html**

Definición y uso

La expresión [^ abc] se utiliza para encontrar cualquier carácter que no esté entre los corchetes. Los caracteres dentro de los corchetes pueden ser cualquier carácter o conjunto de caracteres.

Sintaxis

new RegExp("[^xyz]") o simplemente: /[^xyz]/

Metacaracteres

Los metacaracteres son caracteres con un significado especial:

Metacarácter	Descripción
.	Encuentra un solo caracter, excepto una línea nueva o final de línea
\w	Encuentra un caracter de palabra
\W	Encuentra un caracter no-palabra
\d	Encuentra un dígito
\D	Encuentra un caracter no numérico
\ s	Encontrar un espacio en blanco
\S	Que no se encuentre un espacio en blanco
\b	Encontrar una coincidencia al inicio / final de una palabra

\B	No encontrar una coincidencia al inicio / final de una palabra
\0	Encuentra un caracter NUL
\n	Encuentra un caracter de nueva línea
\f	Encuentra un caracter de avance de página
\r	Encuentra un caracter de retorno de carro
\t	Encuentra un caracter de tabulación
\v	Encuentra un caracter de tabulación vertical,
\xxx	Busca el caracter especificado por un número octal xxx
\xdd	Busca el caracter especificado por un número hexadecimal dd
\uxxxx	Buscar el caracter Unicode especificado por un número hexadecimal xxxx

Ejemplo: **expresiónReg_w.html**

Definición y uso

El \w metacaracter se utiliza para encontrar un caracter de palabra.
Un caracter de palabra es un caracter desde a-z, A-Z, 0-9, incluido el _ (guión bajo) caracter.

Sintaxis

new RegExp("\w") o simplemente: /\w/

Cuantificadores

Cuantificador	Descripción
n+	Coincide con cualquier cadena que contiene al menos un n
n*	Coincide con cualquier cadena que contiene cero o más apariciones de n
n?	Coincide con cualquier cadena que contiene cero o una ocurrencias de n
n{X}	Coincide con cualquier cadena que contiene una secuencia de *X n*'s
n{X, Y}	Coincide con cualquier cadena que contiene una secuencia de X a Y *n*'s
n{X,}	Coincide con cualquier cadena que contiene una secuencia de al menos X *n*'s
n$	Coincide con cualquier cadena con n al final de la misma
^n	Coincide con cualquier cadena con n al comienzo de la misma
?=n	Coincide con cualquier cadena que es seguido por una cadena específica n
?!n	Coincide con cualquier cadena que no es seguida por una cadena específica n

Propiedades de RegExp

Propiedad	Significado
global	Especifica si se está usando el modificador **g**
ignoreCase	Especifica si se está usando el modificador **i**
lastIndex	Índice en el cual comienza la siguiente coincidencia
multiline	Especifica si se está usando el modificador **m**
source	Establece o devuelve el patrón de la expresión regular

Ejemplo: **Propiedad_global.html**

Métodos de RegExp

Método	Significado
compile()	Compila una expresión regular
exec()	Comprueba las coincidencias en una cadena. Devuelve la primera coincidencia.
test()	Comprueba las coincidencias en una cadena. Devuelve **true** o **false**.

Ejemplo: **Metodo_exec.html**

Veamos un **ejemplo de validación de campos de formularios con expresiones regulares**, una de las aplicaciones más habituales de las expresiones regulares. La función de validación se invoca capturando el evento onSubmit del formulario. Si valida devuelve TRUE. Si no valida, la función de validación devuelve FALSE. Esto cancela el submit de modo que el usuario pueda corregir la entrada.

```
<head>
   <script type="text/javascript">
      <!--
      function ValidaTel(formulario) {
         //expresion regular para telefonos
         //permite numeros, espacios, + y -
         var exp_reg_tel = /^[0-9\s\+\-]*$/
```

```
            //comprobar campo tel de formulario
            //usa el metodo test de RegExp
            if(!exp_reg_tel.test(formulario.tel.value)) {
               alert("TELEFONO no valida")
               output = FALSE   //no submit
            } else {
               alert("TELEFONO valida")
               output = TRUE    //submit
            }
            return output
         }
         //-->
      </script>
   </head>

   <body>
<form method="post" name="Form1" action="" onSubmit="return ValidaTel(this)">
         Tel: <input type="text" name="tel" /><br />
         <input type="submit" name="enviar" value="Enviar" />
      </form>
   </body>
```

Ejemplo anterior: **ExpresionRegular.html**