Desarrollo Web en Entorno Cliente.

Unidad 2.

JavaScript

Programación Orientada a Objetos.

- Objetos Predefinidos.
- Array, Number, Date, Math, String.

Introducción.

JavaScript es un lenguaje que permite trabajar con objetos:

– Un objeto es una entidad que reune datos y funciones para trabajar con esos datos.

– Los datos se denominan "Propiedades" del objeto.

– Las funciones se denominan "Métodos" del objeto.

Dentro de un programa podemos tener cero, una ó más instancias (ejemplares) de ese objeto.

Los objetos pueden ser definidos por el usuario ó pueden ser objetos predefinidos de JavaScript.

Objetos propios de JavaScript

Son objetos predefinidos ó intrínsecos del lenguaje.

Mediante el uso de estos objetos podemos interactuar desde el código JavaScript con todos los elementos de una página.

Ejemplos de objetos: El documento, las Tablas, los Formularios, el Navegador, etc.

Ejemplo: El objeto "document" tiene propiedades como: los "datos", el "title"...

Acceder a las Propiedades de los Objetos:

Algunas propiedades son de Lectura/Escritura, otras sólo de Lectura (no podemos modificar su valor).

Para acceder a una propiedad de un objeto: document.title

La sintaxis real sería: window.document.title ya que document es un subobjeto de window (pero la obviamos ya que se presupone que document es hijo de window).

Asignar un valor: document.title = "Nuevo título";

Uso de Métodos de un Objeto:

document.write ("Hola Mundo");

window.alert("Atención");

alert("Atención"); Presuponemos que el padre es Window, osea un método del objeto Window.

For...in

Para recorrer todas las propiedades de un objeto podemos usar:

```
for (var miProp in miObj){
}
```

Esto provoca un bucle por todas las propiedades del objeto miObj.

Ejemplo: Queremos recuperar todas las propiedades del objeto document.

```
for (var miProp in document) {
    document.write (miProp, " = ", document[miProp], "<BR>");
}
```

With

Simplifica la codificación en las instrucciones donde se hace referencia a objetos, pudiendo obviar la notación de puntos para referirnos a propiedades y métodos de un objeto.

Una forma:
```
document.write ("El título es: ", document.title);
document.write ("<br>");
```

otra forma:
```
with (document) {
    write ("El título es: ", title);
    write ("<br>");
    }
```

Definición y Creación de Objetos.

Definimos Objetos.

Ejemplo: Definimos el objeto "Empleado"

```
function Empleado (txtNombre, txtApellidos, iSueldo){
                    //definición de propiedades
        this.nombre = txtNombre;
        this.apellidos = txtApellidos;
        this.sueldo = iSueldo;
                    //definición de médotos
        this.imprimir = fnImprimir;
}

function fnImprimir () {
    with (this) {
        document.write ("Nombre: ", nombre, "Apellidos: ", apellidos, "Sueldo: ", sueldo, "<br />");
    }
}
```

this → Es el propio objeto que se está gestionando en ese código.

Cuando this se usa dentro de un formulario, hace referencia al objeto Form.

Creamos objetos.

New → Crea una copia del objeto y los parámetros de la función se usan para asignar valores a las propiedades.

Ejemplo: Creamos un objeto "miEmpleado"

```
var miEmpleado = new Empleado("José", "García", 5000);
        //Asignamos valor
miEmpleado.apellidos = "Rodriguez Lopez";
var iValor = miEmpleado.sueldo;
miEmpleado.imprimir();
```

Supongamos que queremos imprimir el sueldo del empleado:

```
document.write (miEmpleado.sueldo);
```

Ejemplo: El objeto "impresora"

- Propiedades.
  - Modelo
  - Tipo
  - AnchoPapel
  - Estado
  - EquipoConexión //Propiedad cadena que indica el host al que se conecta la impresora.
    // También puede ser un Objeto de tipo "Equipo"
- Métodos.
  - Imprimir()
  - Comprobar()
  - Pausar()
  - Detenerse()

Constructor del Objeto "Equipo":

```
function Equipo (txtHost, txtAddress, txtCPU, intRAM){
        this.host = txtHost;
        this.address = txtAddress;
        this.cpu = txtCPU;
        this.ram = intRAM;
}
```

Constructor del Objeto "Impresora":

```
function Impresora (txtModelo, txtTipo, intAncho, bEstado, ObjEquipo){
        this.modelo = txtModelo;
        this.tipo = txtTipo;
        this.anchoPapel = intAncho;
        this.estado = bEstado;
        this.equipoConexion = objEquipo;
        this.print = fnImprimeObjeto;
}

function fnImprimeObjeto () {
        document.write ("Impresora: ", this.Modelo, "Tipo: ", this.tipo);
        document.write ("Conectada al host ", this.equipoConexion.host);
}
```

Nota: El objeto "Equipo" es una propiedad del objeto "Impresora".
Es decir, para poder crear el objeto "Impresora" es necesario crear antes "Equipo" aunque puede ser nulo.

```
Var  miEquipo = new  Equipo ("Host1", "192.168.1.1", "Intel QX8300", 4000);
var miImpresora = new Impresora ("HP", "Láser", 12, true, miEquipo);
miImpresora.print ();
```

Como vemos, tenemos dos Constructores:

–En el Constructor del objeto "impresora" se hace referencia a un objeto de la Clase "Equipo".

–En el método print() del objeto Impresora se imprime el nombre del host usando:

nombreObjetoContenedor.nombreObjetoContenido.nombrePropiedadObjetoContenido

Si un Objeto (miEquipo) forma parte de varios objetos, osea está contenido en varios objetos distintos (miImpresora1, miImpresora2, …) y cambiamos una propiedad del objeto miEquipo, ese cambio afectará a todos los objetos en donde este objeto forme parte.

Copiar Objetos.

Var miEquipo2 = miEquipo;

Se crea el objeto y se copia el contenido desde otro objeto pero:

"Comparten las mismas direcciones de memoria"

Si se modifica "miEquipo" también se modificará miEquipo2

Cuando queremos copiar un objeto, lo mejor es:

- Creamos el objeto nuevo con "new"
- Y le asignamos las propiedades una a una.

Se crean dos objetos iguales con vidas diferentes, y pudiéndolo modificar independientemente.

```
Var miEquipo = new  Equipo ("Host1", "192.168.1.1", "Intel X8300", 4000);
var miEquipo2 = new Equipo (miEquipo.Host,  miEquipo.address, miEquipo.cpu, miEquipo.ram);
```

Podemos modificar "miEquipo2" y no afecta a "miEquipo":

```
miEquipo2.host = "Host2000";
```

Método para Copiar Objetos:

```
fuction copiar (objEquipo) {
        objEquipo.host = this.host;
        objEquipo.address = this.address;
        objEquipo.cpu = this.cpu;
        objEquipo.ram = this.ram;
}

var miEquipo = new  Equipo ("Host1", "192.168.1.1", "Intel X8300", 4000);
var miEquipo2 = new Equipo();
miEquipo.copiar(miEquipo2);
```

Objeto: "Array"

JavaScript implementa un objeto predefinido, solo crearemos una instancia para utilizarlo.

```
Var meses = new Array();
meses[0] = "Enero";
meses[1] = "Febrero";
```

También, podemos crear el objeto y asignarle valores:

```
var meses = new Array ("Enero", "Febrero", …);
```

Para acceder a un elemento, para cargar una variable ó para asignar un valor:

```
var unMes = meses[0];
meses[5] = "Abril";
```

Propiedad: "**length**" para conocer la cantidad de elementos del array.

```
Document.write (meses.length);
```

Nota:

Cuando copiamos variables por ejemplo de texto, si modificamos el valor de una variable no se modifica el valor de la otra variable.

Pero cuando copiamos objetos, si modificamos el contenido de un objeto se modificará el del otro objeto.

Los Arrays en JavaScript son objetos por lo que si hacemos una copia y la modificamos, se modificará en el otro.

```
Var  matMeses = new Array ("Enero", "Febrero", "Marzo");
var matMeses2 = matMeses;
matMeses2[0] = "Julio";
document.write(matMeses[0]);          → Se imprimirá Julio.
```

Matrices Asociativas:

Son aquellas en las que los índices no son números, sino claves de texto:

```
var meses = new Array();
meses["Enero"] = "un valor asignado al elemento";
meses["Febrero"] = "otro valor asignado al elemento";
```

Estas matrices no se recorren secuencialmente con un índice.

Para ello, usamos "**for … in**"

```
var dias = new Array ("Lunes", "Martes", …);
for (var elem in dias) {
        document.write(dias[elem], "<br>");
}
```

Matrices Multidimensionales:

Cuando un elemento de una matriz es otra matriz:

```
var mat = new Arrray();
mat [0] = new Array(1, 2, 3);
mat [1] = new Array(4, 5, 6);
mat [2] = new Array(7, 8, 9);
```

Para Recorrerla usamos dos bucles anidados:

```
for (var fila=0; fila<mat.length; fila++){
    for (var col=0; col<mat[fila].length; col++){
        document.write ("Elemento mat[", fila,"] [", col, "] ", mat[fila][col], "<br/>");
    }
}
```

Métodos del Objeto "Array":

concat(): Devuelve una matriz resultante de dos matrices concatenadas.

Join(): Convierte una matriz en una cadena de caracteres formada por cada uno de los elementos de la matriz separados por un carácter (parámetro del método).

pop(): Elimina el último elemento de una matriz.

shift(): Elimina el primer elemento de la matriz.

unshift(): Inserta elementos al principio de la matriz.

push(): Añade elementos al final de la matriz.

reserve(): Invierte el orden de la matriz.

slice(): Recorta el Array. Utiliza dos valores (inferior y superior).
Devuelve una matriz formada por el elemento del índice inferior hasta el superior sin incluirlo.

sort(): Ordena la matriz en orden creciente.
Si después aplicamos el método **reverse()** nos devuelve una matriz de orden decreciente.

splice(): Reemplaza una parte de la matriz a partir de un índice por una cantidad de elementos determinado.

Objeto: "Number"

Para hacer operaciones con datos numéricos.

No se suelen crear objetos de tipo Number con el constructor New, ya que se asignan directamente los valores numéricos a una variable.

Se utiliza para indicar el máximo y el mínimo valor posible que podemos representar, e informar al usuario cuando sobrepase esos límites.

```
<script type="text/javascript">
    alert("Propiedad MAX_VALUE: " + Number.MAX_VALUE);
    alert("Propiedad MAX_VALUE: " + Number.MIN_VALUE);
    alert("Propiedad NaN: " + Number.NaN);
    alert("Propiedad NEGATIVE_INFINITY: " + Number.NEGATIVE_INFINITY);
    alert("Propiedad POSITIVE_INFINITY: " + Number.POSITIVE_INFINITY);
    var N1 = new Number (3.141592);
    alert ("Pi formateado: " + N1.toPrecision(3));;
</script>
```

Objeto: "Date"

Nos permite manipular fechas y horas.

- El año se representa con 4 dígitos.
- El mes es un índice de la matriz (0 enero, 11 diciembre)
- El día (1-31), Día de la semana (0 domingo, 6 sábado)
- Hora (0-23) , Minutos y Segundos (0-59), Milisegundos (0-999)
- Uso horario utiliza la configuración regional del equipo cliente.

Creación de un objeto "Date":

```
var  dtFecha = new  Date();
document.write (dtFecha);
alert ("La fecha actual es: " + dtFecha);
```

Constructor:

var dtFecha = new Date (2012, 11, 10); → Sale Diciembre ya que es el 11. 10/12/2012
var dtFecha = new Date (2010, 0, 25, 8, 30, 15); → Fecha 25/1/2010 Hora 8:30:15

Métodos de Lectura:

getDate(): Devuelve el número del día de la fecha. (1-31)
getDay(): Devuelve el número del día de la semana. (0-6)
getFullDay(): Devuelve el año en cuatro dígitos.
getHours(): Devuelve la hora de la fecha. (0-23)
getMilliseconds(): Devuelve los milisegundos de la hora de la fecha (0-999)
getMinutes(): Devuelve los minutos de la hora de la fecha (0-59).
getMonth(): Devuelve el número de meses de la fecha (0-11).
getSeconds(): Devuelve los segundos de la hora de la fecha (0-59).
getTime(): Devuelve los milisegundos transcurridos desde 1/1/1970 hasta la fecha del objeto.
getTimezoneOffset(): Devuelve los minutos de diferencia entre hora local y zona horaria.
getYear(): Devuelve el año de la fecha en cuatro dígitos, se recomienda getFullYear.
getUTC...(): Devuelve fecha/hora en modo UTC (Coordenadas Universales.

Métodos de Escritura:

setDate(): Asigna el número del día de la fecha.
setFullYear(): Asigna el año en cuatro dígitos.
setHours(): Asigna la hora de la fecha (0-23)
setMilliseconds(): Asigna los milisegundos de la hora de la fecha (0-999)
setMinutes(): Asigna los minutos de la hora de la fecha (0-59)
setMonth(): Asigna el número de mes de la fecha (0-11)
setSeconds(): Asigna los segundos de la hora de la fecha (0-59)
setTime(): Asigna los milisegundos transcurridos desde 1/1/1970 hasta la fecha del objeto fecha
setTimeout(): Define una función que se ejecutará cuando pase un tiempo determinado:

```
timer = setTimeout("fn()", lapso);
```

→ Se ejecutará la función fn cuando transcurra el tiempo especificado en la variable lapso.
setInterval(): La llamada a la función no se interrumpe nunca sino que se reinicia automáticamente cada vez que se cumple el intervalo. Podemos interrumpir el proceso con la función clearInterval().

Métodos de Conversión:

Convierten objetos de tipo Date en cadenas de texto o en milisegundos según los criterios.

toDateString(): Convierte la fecha del Objeto Date en una cadena de caracteres.

toTimeString(): Convierte la parte de tiempo de un objeto Date en una cadena de texto.

toLocaleDateString():Convierte la fecha del objeto Date en una cadena según convención local.

Objeto: "Math"

Para operaciones matemáticas en JavaScript. No necesitas crear objeto con el constructor new.

```
<script type="text/javascript">
        Var r = prompt ("Ingresa el radio: ");
        var area = Math.PI + Math.pow(r, 2);
        alert ("El \xe1rea es: " + area + "cms cuadrados");
</script>
```

Operaciones de Conversión a Número:

Una variable puede contener en un momento dado un valor numérico y en el siguiente instante podría almacenar un objeto de cualquier clase.

Esto puede llevarnos a un problema cuando se realizan operaciones aritméticas ya que hay que operar con números y en muchos casos esas variables están almacenadas como cadena de caracteres, para ello usamos:

parseFloat(): Convierte a un número con decimales.

parseInt(): Convierte aun número entero.

NaN → Si la función de conversión no encuentra un número al inicio de la cadena a convertir, la función parse devuelve NaN (Not a Number, no es un número).

IsNaN(): Nos permite averiguar si una variable es numérica. (False ó True si es un número)

Operación de Conversión de Número a Cadena:

toString(): Convierte una variable numérica a tipo String. var str1 = toString(3000);

A través de un método indirecto:

```
var miNum = 3000;
miNum = miNum + " euros";
```

Nota: Al concatenar a una variable numérica una cadena, la variable resultante es tipo cadena.

Métodos del Objeto "Math":

ceil(): Devuelve el entero más cercano y superior al número pasado como parámetro.
floor(): Devuelve el entero más cercano e inferior al número pasado como parámetro.
max(): Devuelve el valor mayor de los parámetros pasados.
min(): Devuelve el valor menor de los parámetros pasados.
round(): Devuelve el valor entero más cercano al parámetro pasado a la función.
abs(): Devuelve el valor absoluto del número pasado como parámetro.
acos(): Devuelve el ángulo definido por el coseno del número pasado como parámetro.
asin(): Devuelve el ángulo definido por el seno del número pasado como parámetro.
atan(): Devuelve el ángulo definido por la tangente del número pasado como parámetro.
cos(): Devuelve el coseno del ángulo pasado como parámetro.
sin(): Devuelve el seno del ángulo pasado como parámetro.

tan(): Devuelve la tangente del ángulo pasado como parámetro.

exp(): Número e(2,7182) elevado a la potencia del número pasado por parámetro.

Math.exp(2) → Devuelve el número e elevado al cuadrado.

log(): Logaritmo neperiano del parámetro.

Math.log(1) → Es igual a 0

pow(): Valor del primer parámetro elevado a la potencia del segundo.

Math.pow(2,3) → Es 2 elevado a 3

sqrt(): Raíz cuadrada del número pasado como parámetro.

Math.sqrt(25) → Devuelve 5

random(): Devuelve un valor aleatorio entre 0 y 1.

Nota: Cuando no se pueda resolver la operación aritmética se devuelve NaN (Not a Number).

Pero en el caso de una división por cero, no se cancela, devuelve Infinity ó -Infinity.

Constantes: LN2, LN10, LOG2E, LOG10E, PI, SQRT2(Raiz Cuadrada de 2), E(Constante e Euler)

Formateo de Números:

JavaScript no incluye una función adecuada para añadir ceros no significativos para presentar correctamente los importes, pero podemos desarrollar una función para resolver ésto.

Objeto: "String"

Para declarar una cadena de caracteres:

```
var miCad = "Mi cadena de caracteres";
var miCad = 'Mi cadena de caracteres';
var miCad = "El modulo es: \"Desarrollo Web Entorno Cliente\"";
```

Concatenar:

```
var miCad1 = "El modulo es: ";
var miCad2 = " \"JavaScript\" ";
var miCad = miCad1 + miCad2;
```

Se puede **Concatenar Texto + Números** → JavaScript convierte el Número en texto.

```
Var miCad1 = "El modulo tiene: ";
var intTemas = 30;
var miCad = miCad1 + intTemas + " Temas";
```

Longitud de la Cadena:

```
var  intLong = miCad.length;
var  intLong = "El modulto tiene".length;
```

Métodos del Objeto "String":

charAt(): Accede a un carácter de la cadena usando un índice (comienza en 0)
charCodeAt(): Devuelve el valor numérico decimal del código de un carácter de la cadena.
fromCharCode(): Devuelve el carácter que corresponde a un código decimal.
substring(): Devuelve subcadena de una cadena, a partir de una posición de inicio y fin.
slice(): Equivale al método substring()
substr(): Devuelve subcadena de una cadena, a partir de inicio y una longitud.
IndexOf(): Devuelve posición de inicio de una subcadena correspondiente a una búsqueda que aparece dentro de una cadena. Si no la encuentra devuelve -1.
lastIndexOf(): Devuelve posición de inicio de la última subcadena correspondiente a una búsqueda que aparece dentro de una cadena. S no la encuentra devuelve -1.
toLowerCase(): Convierte a minúsculas.
toUpperCase(): Convierte a mayúsculas.

Métodos Vinculados al formateo HTML

Estos métodos del objeto String se relacionan con las etiquetas HTML de formateo de texto.

```
Var  str1 = "Pepe";
document.write(str1.bold());
```

→ Obtiene <B>Pepe<B> (añade las etiquetas <B>)

anchor(): Crea <A> </A> en la cadena de texto.

"Mi enlace".anchor("miNombre") → genera en HTML:

<A name="miNombre">Mi enlace</A>

big(): Crea <BIG> </BIG> → Aumenta el Tamaño del Texto

blink(): Crea <BLINK> </BLINK> → Texto que Parpadea

bold(): Crea <B> </B> → Texto en Negrita

fixed(): Crea <TT> </TT> → Todos los caracteres con el mismo ancho

fontcolor(miColor): Crea <FONT color="miColor"> </FONT> →Muestra texto en el color elegido

fontsize(miSize): Crea <FONT size="miSize"> </FONT> → Texto en el tamaño indicado.

Italics(): Crea <I> </I> → Texto en cursiva

link(url): Crea <A href="miLink"> </A> → "Mienlace".link("miLink")

small(): Crea <SMALL> </SMALL> → Reduce el Tamaño del Texto

strike(): Crea <STRIKE> </STRIKE> → Tacha el Texto.

sup(): Crea → Pone el Texto como exponente.

sub(): Crea → Pone el Texto como subíndice.

Acceder a los Caracteres de una Cadena:

```
<script type="text/javascript">
        var miCad = "Esto es una cadena";
        for (var i=0; i<miCad.length; i++){
                document.write(miCad.charAt(i));
        }
</script>
```

Nota: El método charAt(i) es de lectura. miCad.charAt(2)="X" //Es incorrecto

Saber el Código ASCII de un determinado carácter:

```
<script type="text/javascript">
        var miCad = "Esto es una cadena";
        for (var i=0; i<miCad.length; i++){
                document.write(miCad.charCodeAt(i));
        }
</script>
```

Saber el carácter a partir del Código Ascii:

```
var miCad = String.fromCharCode(201, 115, 116, 97);
```

Resolver Universalidad:

En la página se indica el idioma utilizado en el contenido para que el navegador sepa qué tabla de caracteres debe utilizar y qué idioma se utiliza en la página:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es-ES" lang="es">
```

Subcadenas:

substring(): Permite extraer una parte de una cadena y por parámetro se indica la posición inicial a partir de la que substraer la cadena y en otro parámetro la posición de final.

El método slice() es equivalente a substring().

```
var str1 = "Este es un texto de prueba".substring(1,3);  →  str1=st
```

Si el segundo parámetro se omite, se considera hasta el final de la cadena.

El método substr() es similar al método substring() la diferencia es que el segundo parámetro no indica la posición siguiente a la posición de fin sino la longitud de la subcadena.

Búsqueda dentro de una cadena:

```
var str1 = "texto1 texto2".lastIndexOf("texto");  //devuelve 7
var str2 = "texto1 texto2".lastIndexOf("NN");     //devuelve -1
```

Esto se utiliza para obtener la extensión de un archivo:

```
var str2 = "archivoCopiaSeguridad.zip".lastIndexOf(".");
//devuelve la posición del punto
```

El método indexOf() es similar a lastIndexOf() salvo que obtiene la primera coincidencia de la subcadena buscada (no la última) y opcionalmente se puede indicar un punto de partida a la búsqueda:

```
var str1 = "texto1 texto2".indexOf("texto");          //devuelve 0
var str2 = "texto1 text02".indexOf("texto", 2);       //devuelve 7
```

Conversión a Mayúsculas ó Minúsculas:

```
var strNombre = "marTínez";
document.write(strNombre.toLowerCase(), "<br/>");   → Conversión a minúsculas.
document.write(strNombre.toUpperCase(), "<br/>");   → Conversión a Mayúsculas.
document.write( strNombre.charAt(0).toUpperCase() + strNombre.substring(1).toLowerCase() );
```

→ El primer carácter de la cadena y después se fuerzan minúsculas.

Evaluación de una cadena:

```
var numpag = 10;          document.write(eval("numpag"));  → Imprime contenido de numpag
var var1 = "numpag";      document.write(eval(var1)); → Imprime 10
                          document.write(eval("20+4")); → Imprime 24
```

Escape de Caracteres:

Con la función escape() se puede codificar automáticamente una cadena de texto incluyéndole los escapes necesarios a los caracteres problemáticos.

```
<script type="text/javascript">
        var miCad = "áéíóú  aeiou &%'";
        document.write(escape(miCad));
</script>
```

Para poder recuperar una cadena codificada con caracteres de escape a su formato original se utiliza la función unescape() que devuelve una cadena limpia, con los escapes convertidos a su valor real.

```
<script type="text/javascript">
        var miCad = "Esto es un texto de ejemplo";
                document.write("Texto original =", miCad, "<BR/>");
        var miCad2 = escape(miCad);
                document.write("Texto con escapes =", miCad2, "<BR/>");
        var miCad3 = unescape(miCad2);
                document.write("Texto sin escapes =", miCad3, "<BR/>");
</script>
```

Expresiones Regulares:

Para crear un objeto RegExp: **var expreg = new RegExp (cadenaExpReg, opcion);**

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
        function esVálido(str1){       // se crea la expresión regular
                var expreg = new RegExp("^[a-zA-Z ]{1,20}$","g");
                if (expreg.test(str1)){
                        return str1 + " es OK";
                } else {
                        return str1 + " es KO";
                }
        }
        // KO: por tener un número
        document.write(esVálido("1Ricardo"), "<BR />");
        // OK
        document.write(esVálido("Juan"), "<BR />");
        // KO: por tener un acento
        document.write(esVálido("José"), "<BR />");
        // OK
        document.write(esVálido("Juan Antonio"), "<BR />");
        // KO: demasiado largo
        document.write(esVálido("Juan Antonio Rodriguez"), "<BR />");
</script>
</head>
<body>
</body>
</html>
```

```
1Ricardo es KO
Juan es OK
José es KO
Juan Antonio es OK
Juan Antonio Rodriguez es KO
```

Expresiones Regulares útiles:

/ \d{9}/ → Número de Teléfono

/ \d{8}[a-zA-Z]/ → Número de DNI

/ \d{2}-\d{2}-\d{4}/ → Fecha (día-mes-año)

/([0][1-9]|1[0-2]):[0-5]\d:[0-5]\d/ → Hora (hora:minutos:segundos en formato de 12 horas)

/([01]\d|2[0-3]):[0-5]\d:[0-5]\d/ → Hora (hora:minutos:segundos en formato de 24 horas)

/ \w+@\w+\.\w{2,3}/ → Dirección de Correo Electrónico

Método test() del objeto RegExp

El método test() me permite ejecutar la expresión regular y devuelve un valor booleano: verdadero ó falso, para indicar si la verificación fue correcta ó no.

Ejemplo: Validar el DNI

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
	function esVálido(str1){	// se crea la expresión regular
		var expreg = new RegExp("^[0-5][0-9]{7}-[A-Z]$","g");
		if (expreg.test(str1)){
			return str1 + " es OK";
		} else {
			return str1 + " es KO";
		}
	}
	// KO: no tiene el formato correcto
	document.write(esVálido("ABC"), "<BR />");
	// OK
	document.write(esVálido("11222333-T"), "<BR />");
	// KO:  es más grande que 59.999.999
	document.write(esVálido("77777777-M"), "<BR />");
	// KO :  faltan dígitos y la letra final
	document.write(esVálido("12-"), "<BR />");
	// KO: falta el guión
	document.write(esVálido("22333444T"), "<BR />");
</script>
</head>
<body>
</body>
</html>
```

Método exec() del objeto RegExp

Devuelve la **primera coincidencia** encontrada dentro de una cadena en base a la expresión regular utilizada y la cadena de texto pasada como argumento.

Ejemplo: Supongamos que en una carta de 2000 caracteres queremos encontrar un número de DNI, hay que tener en cuenta el punto separador de miles.

```
<script type="text/javascript">

        //Creamos el Objeto
        var expreg = new RegExp("[0-6] [0-9].[0-9]{3}.[0-9]{3}-[A-Z]]","g");

        // Texto de la carta
        var strTexto = "Les envío esta carta para informarles que mi número de DNI es
        38.999.234-V y que por error me han enviado uno con el número 36.666.666-W";

        //Impresión del texto completo
        document.write(strTexto, "<BR />");

        //Impresión del texto por separado
        document.write("DNI: ", expreg.exec(strTexto), "<BR />");

</script>
```

Ejemplos de Métodos del Objeto String:

– **Método math()**

Con exec() se devuelve la primera coincidencia, pero si queremos obtener todas las apariciones de los datos que coincidan con el patrón regular usamos math().

Ejemplo: Supongamos que en una carta de 2000 caracteres queremos encontrar un número de DNI, hay que tener en cuenta el punto separador de miles.

```
<script type="text/javascript">
        //Creamos el Objeto
    var expreg = strTexto.math(/ \b[0-6] [0-9].[0-9]{3}.[0-9]{3}-[A-Z]]\b/ g);
        // Texto de la carta
    var strTexto = "Les envío esta carta para informarles que mi número de DNI es 38.999.234-V y
                    que por error me han enviado uno con el número 36.666.666-W";
                    //Impresión del texto completo
    document.write(strTexto, "<BR />");
    document.write("**********<BR />")

    if (expreg == null) {
        document.write("No se encontraron resultados");
    } else {
        document.write("Lista de DNI encontrados dentro del texto:<br>");
        document.write(expreg + "<br>");
            // En el objeto RegExp la propiedad input nos da la cadena analizada.
        document.write("texto analizado:<br>" + RegExp.input);
    }
</script>
```

– **Método split()**

Cuando queremos localizar subcadenas que se encuentran definidas entre unos separadores determinados, devolviendo una matriz de resultados compuesta por todos los elementos detectados entre los separadores (sin incluirlos). La expresión regular va a definir los separadores o delimitadores, entre los que se encuentra la subcadena.

Ejemplo: Extraer los nombres entre separadores.

Supongamos: PHP - JavaScript - Visual Basic .NET; C++, Visual C++, Java, Pascal-Fortran; PL/1
Definimos un patrón de separadores: [-;,]+ los separadores posibles son: - ; ,
El signo + indica que al menos se espera alguno de estos separadores: cualquiera que aparezca es suficiente.

Si aparecen dos o más separadores seguidos, todos se tomarán como si fueran un único separador.

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
  //lista de elementos con separadores
    var strTexto = "PHP-JavaScript- Visual Basic .NET; C++, Visual C++, Java, Pascal-Fortran; PL/1";
    document.write(strTexto, "<br />");
    var regexp = new RegExp("[-:,]+","g");
    var miLista = strTexto.split(regexp);
    if (miLista == null) {
        document.write ("Posible error en la expresion regular");
    }else {
        document.write ("Lista de elementos <br />");
        for (var i=0; i < miLista.length; i++) {
            document.write (miLista[i] + "<br />");
        }
    }
</script>
</head>
<body>
</body>
</html>
```

– **Método replace()**

Reemplaza caracteres de una cadena de texto.

Mediante una expresión regular se define un grupo de caracteres y después el método replace() permite reemplazar cualquier elemento del grupo de la expresión por un carácter determinado que se pasa como segundo parámetro del método.

```
var nuevoTexto = textoCadena.replace (expresiónRegular, carácterReemplazo);
```

El método replace() devuelve una nueva cadena con los caracteres reemplazados, si es que se encontraron, sino sería una cadena idéntica a la original.

Ejemplo: Queremos reemplazar un conjunto de caracteres por un guión.

```
<!DOCTYPE HTML>
<html>
  <head>
      <script type="text/javascript">
          // crea el objeto RegExp
          var expreg = new RegExp("[aeiou]","g");
          var strTexto = "Éste es un texto de prueba en el que se reemplazan las vocales
                          por un guión.";
          // Impresión del texto completo
          document.write(strTexto, "<br />");
          var miTexto = strTexto.replace(expreg,"-");
          document.write(miTexto);
      </script>
   </head>
<body>
</body>
</html>
```

Ejemplo: Verificar una Dirección de Correo

Las expresiones regulares se usan para validar los datos ingresados por el usuario.

La validación se hace en el propio Navegador sin enviar los datos al servidor.

La dirección de Correo tiene el formato: xxx@yyy.zzz con longitudes diferentes.

xxx: Nombre (Al menos un carácter alfanumérico sin acentos, se permiten puntos y guiones)

yyy: Dominio (Al menos dos caracteres alfanuméricos sin acentos, se permiten puntos y guión medio, no guión bajo).

zzz: Extensión (Tiene entre 2 y 5 caracteres alfabéticos).

-En la dirección de correo se admiten mayúsculas y minúsculas.

La cadena empieza con ^ y termina con $, indica que la cadena no acepta caracteres antes ó despues, sino esos.

```
<!DOCTYPE HTML>
<html>
<head>
  <script type="text/javascript">
        function validarEmail(str1) {
                // crea el objeto RegExp
                var expregMail = new RegExp("^[a-z0.9._-]+@{1}[a-z0.9.-]{2,}[.]{1}[a-z]{2,5}$", "i");
                return expregMail.test(str1);
        }
        var strEmail = "xxx@yyy.com";
        document.write(strEmail, " Validación: ", validarEmail(strEmail), "<br />");
        strEmail = "xxx44yyy.com";
        document.write(strEmail, " Validación: ", validarEmail(strEmail), "<br />");
        strEmail = "juan@ . co.uk";
        document.write(strEmail, " Validación: ", validarEmail(strEmail), "<br />");
        strEmail = "juan@telefonica.net";
        document.write(strEmail, " Validación: ", validarEmail(strEmail), "<br />");
  </script>
</head>
<body>
</body>
</html>
```

Ejemplo: Convertir las posibles direcciones Web en Enlaces reales a la página Web.

Supongamos un texto donde aparecen uno o varios nombres de direcciones webs, la idea es convertir dichas direcciones en enlaces a las páginas.

Por lo que en el texto habrá que incluir ...

```
<!DOCTYPE HTML>
<html>
  <head>
      <script type="text/javascript">

          function  incluirEtiquetasLink(str1) {
             // crea el objeto RegExp
             var expreg = new RegExp("http://www.[a-zA-Z0.9._-]{4,})", "gi");

             //$1 va a hacer referencia al contenido que coincide con la expresión regular, de esta
             // manera se pueden estructurar variables.

             return  str1.replace(expreg, "<a  href=\"$1\">$1</a>");
           }

          var strTexto = "Éste es un texto de prueba en el que se las direcciones web se crean como
                          enlaces por ejemplo www.google.es y por ejemplo www.iescamas.com ";

             // Impresión del texto completo sin crear los link
             document.write(strTexto, "<br />**********<br />");

             // Impresión del texto conviertiendo las direcciones en link
             document.write(incluirEtiquetasLink (strTexto));


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