Unidad 10

Lenguajes de script en cliente: Javascript

"Hay sólo dos clases de lenguajes de programación: aquellos de los que la gente está siempre quejándose y aquellos que nadie usa"

Bjarne Stroustrup (creador de C++)

Javascript es un lenguaje de programación interpretado por el navegador y creado por Netscape. Su función original era mejorar la funcionalidad de los formularios en los documentos HTML. Hoy en día es el lenguaje más usado en el lado del cliente de una aplicación web para aportar a estas dinamismo e interactividad con el usuario. Continúa ligado al funcionamiento de los formularios pero eso ya es sólo una más de sus funciones. Además, la nueva versión del estándar HTML (HTML5) incluye "de serie" multitud de nuevas librerías en Javascript que amplían enormemente la funcionalidad de una página web.

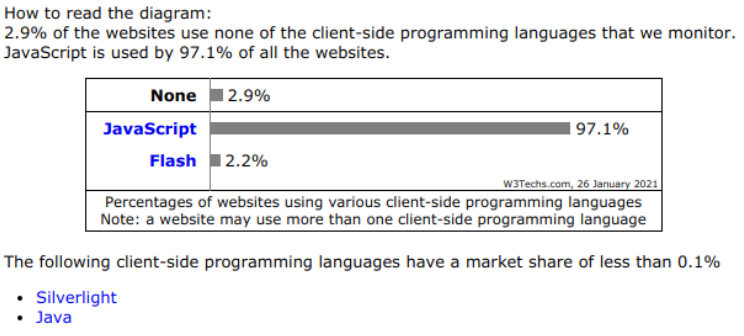

Imagen 4 – Fuente: http://w3techs.com/technologies

Sus principales características son las siguientes:

- Lenguaje interpretado y debilmente tipado. Se ejecuta en el lado cliente de la aplicación web (aunque existe una forma marginal de usarlo en el servidor, por ejemplo con node.js)
- Se interpreta en el navegador, así que la apariencia final y ciertos detalles de la ejecución dependen de este (aunque se supone que se siguen unos estándares así que la parte fundamental debería de ser muy similar).
- La velocidad de ejecución en el navegador es muy importante y es donde se está librando la parte más importante de la actual batalla de navegadores (ver capítulo anterior).
- Se creó inicialmente para validación de formularios, pero actualmente se usa, además de para esto, para crear aplicaciones web dinámicas
- Tiene una sintaxis muy similar a C y a Java
- El código Javascript se puede ubicar de diferentes formas en la página web y se puede ejecutar de dos formas distintas: durante la carga de la página web o en respuesta a eventos.

Recientemente Google ha impulsado una alternativa que se llama **Dart** pero que por el momento no ha sido acogida con demasiado entusiasmo por parte de los expertos.

El programa más sencillo escrito en Javascript sería la siguiente versión del ya clásico **"Hola Mundo".** Usaremos siempre para mayor sencillez el esqueleto que usan los documentos de HTML5. El código javascript es el contenido dentro del elemento con la etiqueta **script** que aparece en el ejemplo. Es lo único que aparece en el body, pero si introduces otros elementos html (títulos, párrafos, o cualquier otra cosa) diferentes antes

y/o después del script verás como la ejecución del mismo interrumpe el renderizado de la página HTML. Hasta que no pulsemos el botón que aparece en el diálogo y desaparezca el mensaje, no proseguirá el dibujo de la página.

```
<!DOCTYPE html>
<html lang="es">
	<head>
		<meta charset="UTF-8" / >
		<title>Hola Mundo en Javascript</title>
	</head>
	<body>
		<script type="text/javascript" >
			alert("¡Hola Mundo!");
		</script>
	</body>
</html>
```

Veamos sobre el ejemplo anterior las primeras reglas de sintaxis:

- Es sensible a mayúsculas y minúsculas. No es lo mismo alert que Alert o ALERT.

Los argumentos que usaremos más adelante para capturar eventos no son Javascript sino HTML y por tanto no son sensibles a mayúsculas o minúsculas. Por ejemplo, da igual ONCLICK que onclick. No obstante, XHTML recomienda escribirlos en minúsculas, así que esta será también la sintaxis preferida por nosotros.

- Los sangrados son muy importante para la claridad del código como en cualquier otro lenguaje, pero no son obligatorios como en python.
- Las sentencias se pueden separar con un punto y coma (;) o con un salto de línea. Si escribimos cada instrucción en una línea (la mejor opción por claridad en el código) el ; no es obligatorio pero si recomendado dado que nos permitirá adaptarnos con

mayor facilidad a otros lenguajes de programación que si lo exigen como, por ejemplo, PHP o java.

- Si queramos encadenar más de una sentencia en la misma línea si que es obligatorio usar el punto y coma como separador.

Comentarios

Los comentarios pueden escribirse de dos formas diferentes. Cuando ocupan una sóla línea (ya sea completa o parcialmente) se utiliza el símbolo //

```
// para una única línea de comentario
alert ("Hola"); // hasta el final de la línea
```

Cuando nuestro comentario ocupará varias líneas usamos el símbolo /* para el inicio y el */ para el final:

```
/* Para
varias líneas
de comentario */
```

No olvides nunca que todo lo que escribas en el código Javascript se envía al navegador del usuario y este puede verlo, así que no es ni el sitio ni el modo de dejar nada que pueda comprometer la seguridad de tu aplicación (contraseñas, por ejemplo) o de ofender al usuario o gastar bromas de ningún tipo.

Variables

Las variables son espacios de memoria que nos sirven para guardar datos. En javascript no hace falta declararlas previamente a su uso aunque podemos hacerlo si queremos y es una buena práctica. Los dos siguientes ejemplos son iguales:

```
// La variable x acaba valiendo 5
x = 2 + 3;

// Idem, pero aquí hemos predefinido antes la variable
var x;
x = 2 + 3;
```

También podemos asignar un valor a la variable en el momento de definirla:

```
var x = 5;
```

Javascript es lo que se denomina un lenguaje de programación "debilmente tipado" . Cuando definimos una variable no tenemos por que decir que tipo de dato guardará (enteros, decimales, cadenas de texto, etc.) pero una vez que inicializamos una con un determinado valor ya no deberíamos de cambiarla a otro diferente. Esto, además, dificultaría la interpretación del código y no es lo que buscamos.

Para nombrar una variable pueden usarse caracteres alfanuméricos y el símbolo de subrayado (_) pero nunca pueden empezar por un número. Contador, silla13, numero_alumnos o _encontrado son ejemplos de variables válidas. Las variables también son sensibles a mayúsculas o minúsculas así que Contador no es lo mismo que contador.

El ámbito de las variables (donde tienen existencia) es el bloque donde se declaren o usen por primera vez. Llamamos variables locales a las que tienen como ámbito una función o un bloque concreto y globales a las que tienen como ámbito cualquier función de una página. Esto se consigue declarándola fuera de todas las funciones (ya las veremos mas adelante). Si existen dos variables con el mismo nombre y diferentes ámbitos siempre se tomará la más cercana. En cualquier caso hay que tener mucho cuidado con esto porque puede que las variables no se destruyan de forma automática fuera del bloque donde tienen existencia y esto podría dar lugar a errores. Lo mejor es no repertir variables con el mismo nombre ni usarlas fuera del ámbito de su existencia.

Desde el año 2015 se introdujo como novedad en Javascript una nueva forma de declarar las variables con la palabra reservada let. Las variables declaradas con let no pueden redeclararse (con var si) lo cual nos proporciona una seguridad adicional al codificar

```
var x = "Hola mundo";
var x = 0;
// Funciona correctamente pero puede dar lugar a error
let x = "Hola mundo";
let x = 0;
// Error de sintaxis
```

Adicionalmente, las variables definidas con var dentro de un bloque que no sea una función (un if, un for, un while…) pueden usarse fuera de el. Sin embargo las definidas con let no, lo cual refuerza la idea de ámbito de las mismas

```
while(condicion){
        var x = 2;
        }
// x puede usarse aquí

while(condicion){
        let x = 2;
        }
// x no puede usarse aquí
```

Otro ejemplo:

```
let x = 10;
// Aquí x vale 10
while(condicion){
        let x = 2;
        // Aquí x vale 2
        }
// Aquí x vuelve a valer 10

var x = 10;
// Aquí x vale 10
while(condicion){
        var x = 2;
        // Aquí x vale 2
        }
// Aquí x sigue valiendo 2
```

En 2015 también Javascript introdujo una nueva palabra reservada para definir constantes. Las constantes se usan igual que las variables pero tienen que ser declaradas e inicializadas con un valor que luego ya jamás puede alterarse

```
const PI = 3.141592653589793;
PI = 3.14;      // No se puede hacer
PI = PI + 10;   // Tampoco se puede hacer
```

Las variables y constantes pueden guardar números (enteros o decimales), cadenas de texto (entre comillas simples o dobles) o valores booleanos (true o false). Los valores decimales se separan con punto y no con coma.

Operadores aritméticos

Los operadores aritméticos típicos son =, +, -, *, / y %. El último devuelve el resto de la división entera entre los dos operadores. Por ejemplo, 8 % 2 devolvería 0 y 7 % 4 devolvería 3

El resultado de una operación aritmética se suele asignar sobre una variable que siempre debe de estar a la izquierda de la operación. Se usa para esto el operador de asignación (=)

```
iva = precio * 0.21;
semanas_completas = 31 % 4;
```

> Más adelante veremos que el doble signo igual (==) tiene un uso diferente. ¡Presta atención a no confundirlo desde este primer momento!

Los operadores ++ y -- incrementan o decrementan en uno el valor de la variable que acompañan. Por ejemplo numero++ o stock--

Cuando a ambos lados del signo igual se repite un operando podemos escribirlo de forma simplificada con unos operadores especiales: +=, -=, *=, /= y %=. Por ejemplo, las dos siguientes líneas hacen lo mismo:

```
cuadrado = cuadrado * 2;
cuadrado *= 2;
```

Los operadores + y – se usan también como operadores unarios para determinar el signo de una cantidad: +5, -valor, etc. El signo + es también útil para otras cosas. Por un lado sirve para concatenar cadenas. "Hola" + " " + "mundo" daria como resultado la cadena "Hola mundo". Por otro lado, cuando tenemos caracteres numéricos dentro de una cadena el operador + nos ayuda a "transformarlos" fácilmente en números.

Cuando Javascript hace una operación aritmética devuelve el resultado con todos los decimales posibles. Para "remediar" esto usamos el método **toFixed**. Por ejemplo, si la variable num guarda el número 3.14.159 la expresión **num.toFixed(2)** nos devolverá ese mismo número con sólo dos decimales, o sea, 3,14

Operadores lógicos

Los operadores lógicos nos sirven para hacer comparaciones y siempre devuelven un valor booleano (true o false). Son: ==, !=, <, <=, >, >=

Podemos combinar diferentes operaciones lógicas con los operadores &&, || o ! (que significan AND, OR o NOT)

Cuando comparamos dos cadenas de texto lo que hacemos en realidad es ver cual es menor alfabéticamente que la otra. La comparación se hace caracter a caracter. Cuando la cadena combina letras (mayúsculas y

minúsculas), números y símbolos se usa la posición del caracter en las tablas ASCII para determinar cual es menor o mayor.

Cuando tenemos dudas sobre el resultado en operaciones complejas podemos usar los paréntesis para agrupar resultados. Esto es válido también cuando usamos operadores aritméticos.

Javascript es un lenguaje "credulo". Cualquier cosa diferente de 0 o una cadena vacía se evalúa como true.

Javascript tiene, además, el operador lógico === denominado de estricta igualdad que sólo valida como cierto cuando la comparación coincide en valor y tipo. Un ejemplo:

```
"54" == 54 //Esto valida como cierto
"54 === 54 // Esto valida como falso
```

Escribir en la página web

La función write nos permite escribir contenido nuevo en nuestra página web. Así, por ejemplo:

```
document.write("Hola Mundo");
```

La escritura se realiza en el punto donde el navegador se encuentre cuando vea en el código la sentencia de javascript, ya sea al principio, al final o en medio de la página.

Podemos escribir cualquier cosa: números, resultados de operaciones o, incluso, elementos propios del lenguaje de marcas:

```
document.write("<h1>Hola Mundo</h1><p>Buenos días.</p>");
```

Sentencias de Control

Las estructuras de control son también las típicas de cualquier lenguaje. Empezaremos por ver if y for

La estructura de control if, por ejemplo, evalua una expresión lógica y en función de su resultado (true o false) ejecuta una o varias acciones.

```
if(expresión lógica){
	// Haz algo
}
else{
	// Haz otra cosa
}
```

La parte del else no es obligatoria:

```
if(expresión lógica){
	// Haz algo
}
```

Las llaves sólo son obligatorias si necesitamos ejecutar más de una instrucción en el bloque. Se pueden anidar if hasta cualquier nivel. Los sangrados y las llaves son en este caso indispensables para que el código sea legible:

```
if(expresión lógica){
	if(otra expresión lógica){
		// Haz algo
	}
	else{
		// Haz otra cosa
	}
}
```

La sentencia for se usa cuando queremos ejecutar un mismo bloque de instrucciones un número determinado (o no) de veces:

```
for (i = 0; i < 10; i++) {
	// Haz algo
}
```

El paréntesis que acompaña al for está dividido en tres secciones separadas por punto y coma: inicialización, evaluación y actualización. La primera vez que entramos en el bucle se ejecuta la inicialización y la evaluación (en este orden). Las sucesivas veces se ejecuta la actualización y la evaluación (en este orden también). El código entre llaves se ejecuta mientras que la evaluación se cumpla (se resuelva como true).

La inicialización y/o la actualización pueden constar de más de una sentencia separándolas por comas:

```
for (i = 0, j=5; i < j; i++) {

}
```

Cualquiera de las tres secciones se puede omitir. Si omitimos la segunda puesto que no hay evaluación el bloque de código se ejecutaría de forma infinita a menos que incluyamos una sentencia break en su interior para salir.

Al igual que ocurría con los if, las llaves sólo son obligatorias cuando el bloque de instrucciones tiene más de una sentencia. También podemos anidar sentencias for:

```
for (i = 0,; i < 10; i++) {
        // Se ejecuta tantas veces como valores de i
        for (j = 1; j <= 15; i++) {
                /* Se ejecuta tantas veces como valores de j por
                cada valor de i */
        }
}
```

Un caso particularmente frecuente es cuando necesitamos tomar diferentes decisiones en función del valor de, por ejemplo, una variable. Hacerlo con if se vuelve tedioso cuando las alternativas son más de cuatro. Para facilitar este tipo de situaciones se usa la estructura switch:

```
switch(expresion_a_evaluar) {
    case valor_1:
        // Haz algo
        break;
```

```
case valor_2:
    // Haz otra cosa
    break;
 case valor_3:
    // Haz otra cosa
    break;
default:
     // Otra cosa más
}
```

La sentencia evalua la expresión que aparece en el paréntesis del switch y busca el valor obtenido en cada una de las expresiones case. En el momento en que se encuentra uno igual se ejecuta todo el código que venga a continuación hasta que aparece una sentencia break. Si ninguno concuerda con el valor evaluado se ejecuta el código que aparece continuación de la entrada marcada como default. El bloque default es opcional. Si no aparece y no hay ninguna coincidencia no se ejecuta nada.

Un par de cosas importantes:

- Si el valor de dos case diferentes coincide con la expresión a evaluar sólo se ejecutarán las acciones del primero de ellos ya que al encontrar un break se salta fuera de la sentencia de control.

- A veces podemos querer ejecutar las acciones de dos case en lugar de uno sólo. Observa el siguiente código:

  ```
  numero = Math.floor((Math.random()*3)+1);
  switch(numero) {
      case 1:
          // accion 1
      case 2:
          // accion 2
       case 3:
          // accion 3
      }
  ```

  En el ejemplo anterior se genera un número aleatorio entre el 1 y el 3. Si sale un 1 se ejecutan las acciones 1, 2 y 3. Si sale un 2 se ejecutan las acciones 2 y 3. Por último, si sale un 3 sólo se ejecuta

la acción 3. Esta sentencia (o cualquier otra variante poniendo break en algunos case y en otros no) es perfectamente válida.

También podemos querer que dos cases diferentes ejecuten lo mismo:

```
numero = Math.floor((Math.random()*3)+1);
switch(numero) {
    case 1:
    case 2:
       // accion 1 y 2
        break;
     case 3:
        // accion 3
    }
```

Las sentencias while y do while son similares al for salvo que sólo se evalua la condición de salida y no incluyen ni inicialización ni actualización:

```
i = 0;
while (i < 10) {
    // Ejecuta algo
    i++;
    }

i = 0;
do {
    // Ejecuta algo
    i++;
    } while (i < 10);
```

La principal diferencia entre una y otra es que, puesto que en la segunda forma la condición siempre se evalúa después, el bloque de instrucciones se ejecuta siempre al menos una vez, mientras que en la primera si la evaluación de la condición es falsa desde el principio no se ejecutará nada.

Ventanas de diálogo con el usuario

Hemos usado ya la función alert que nos muestra una ventana con un texto y un botón de Aceptar. Esta función paraliza la ejecución hasta que el usuario no pulsa el único botón que aparece en ella. Javascript dispone de otros dos tipos de ventanas. Vamos a verlas.

La ventana confirm muestra dos botones (OK y Cancelar) y devuelve la elección del usuario para que actuemos en consecuencia como una variable booleana: true si hemos pulsado OK y false si hemos pulsado Cancel:

```
respuesta = confirm("Pulsa un botón");
if (respuesta == true){
        alert("Haz pulsado el botón de OK");
        }
else{
        alert("Haz pulsado el botón de Cancel");
        }
```

La ventana prompt nos muestra una caja de texto y devuelve el valor que hemos escrito en ella. Además, muestra dos botones y devuelve el valor null (false) si hemos salido pulsando el botón de Cancelar en lugar del de OK:

```
texto = prompt("Escribe tu nombre, por favor");
if(texto == null){
        alert("Haz pulsado el botón de cancelar");
        }
else{
        if(texto.length!=0){
                alert("Hola " + texto + ", buenos días \n ¿Cómo
                estás?");
        }
        else{
                alert("Buenos días aunque no quieras decirme tu
                nombre... :-(");
        }
}
```

Números aleatorios

Los números aleatorios son importantísimos en múltiples ámbitos del mundo de la programación. La función de Javascript que nos permite generar un número aleatorio es Math.random()

```
azar = Math.random();
```

La anterior sentencia almacena en la variable azar un número aleatorio comprndido entre el 0 (incluido) y el 1 (excluido). Es decir, el número más alto generado sería el 0,99999999999999999…

Así visto de primeras no parece que nos sirva de gran cosa. Nosotros por regla general necesitamos generar un número entre 1 y 6 (para simular la tirada de un dado convencional), entre 1 y 49 (para un número de la primitiva), entre 0 y 3 (para simular en un juego de rol si aparecen 0, 1, 2 o 3 monstruos)… Veamos como podemos arreglar esto.

```
dado = Math.floor((Math.random() * 6) + 1);
```

La expresión anterior genera un número entre el 1 y el 6. Analicémosla. Al multiplicar por 6 el número generado por random obtenemos un número aleatorio entre el 0 y el 5,9999999999… y si le sumamos un 1 a esto tenemos algo entre el 1 y el 6,99999999999. La función floor, por último, lo que hace es quedarse con la parte entera del número resultante obteniendo finalmente un número entre el 1 y el 6.

Siguiendo el mismo método, si quisiéramos obtener un número entre el 0 y el 3 haríamos lo siguiente:

```
monstruos = Math.floor(Math.random() * 4);
```

Por último, si queremos un número aleatorio entre dos cualesquiera (pongamos el 7 y el 50) el método a seguir es multiplicar por la diferencia entre el mayor y el menor mas uno ((50-7)+1 en este caso), sumarle al resultado el menor (7 en el ejemplo) y truncar la parte decimal del resultado:

```
azar = Math.floor((Math.random() * 44) + 7);
```

Funciones

Una función es un conjunto de instrucciones que siempre se ejecutan unidas. Javascript trae muchas funciones definidas y ya hemos visto algunas de ellas (alert, prompt, etc..). Además, nos permite crear las nuestras.

La sintaxis básica para definir una función es esta:

```
function nombre_funcion(){
   alert("Hola mundo")
   }
```

La anterior es una función que no recibe ningún argumento ni devuelve ningún valor. Lo único que hace es mostrar una ventana con el texto "Hola Mundo". El código que ejecuta la función va siempre entre llaves. En este caso las llaves no son opcionales aunque sólo haya una instrucción, como en el caso anterior. Son siempre obligatorias.

Una función puede además recibir uno o varios argumentos (en el interior del paréntesis y separados por comas si hay más de uno) y devolver un valor al final de su ejecución. La siguiente función, por ejemplo, muestra una ventana con un texto que recibe como argumento y devuelve la longitud del mismo:

```
function nombre_funcion(texto){
   alert(texto);
   return texto.length;
   }
```

Como hemos dicho, las funciones pueden recibir tantos argumentos como queramos pero si devuelven un valor, este sólo puede ser uno y siempre va después de la palabra clave return.

Para ejecutar la función simplemente se pone su nombre:

```
nombre_funcion();
```

o

```
longitud = nombre_funcion("hola mundo");
```

No se puede llamar a una función si esta no ha sido declarada aún. Por eso es buena idea poner siempre las funciones en el head del html y llamarlas desde el body. Otra opción aún mejor es extraer las funciones a un fichero externo distinto al html desde donde se las llama. De esta forma, además, podermos reutilizarlas desde distintas páginas sin tener que duplicar el código.

Para incluir un fichero externo con funciones javascript en nuestro html incluimos la siguiente línea en el head de la página:

```
<script type="text/javascript" src="funciones.js"></script>
```

El fichero funciones.js incluirá las definiciones de las funciones aquí sin necesidad de usar ninguna etiqueta script ni de otro tipo.

Como valor del argumento src hay que expresar una ubicación absoluta o relativa pero completa y válida para el archivo que contiene las funciones.

Vectores

Los vectores son un tipo especial de variables que siempre hay que declarar:

```
var notas_alumnos = new Array(30);
```

También podemos crear un array así:

let notas_alumnos = new Array()

O así:

notas_alumnos = [7, 5.5, 6, 10, 9.2, 4]

Puedes imaginarte un vector como un conjunto de valores homogeneos y que referenciamos con el mismo nombre acompañado de un índice o número de orden. Por ejemplo, el vector anterior valdría para almacenar las notas de los 30 alumnos de una clase. Para referenciar un determinado elemento del vector usamos corchetes e indicamos dentro el número de orden. El primer elemento siempre es el 0.

Por ejemplo:

```
notas_alumnos[0] = 5.5;

if(notas_alumnos[11] < 5)
       alert("El alumno número 12 está cateado");
```

Podemos recuperat también a los elementos del array usando la propiedad at que, además, nos permite hacerlo también desde el final usando índices negativos (la posición -1 se corresponde con la última)

```
primer_alumno = notas_alumnos.at(0)
ultimo_alumno = notas_alumnos.at(-1)
```

Todos los vectores tienen la propiedad length que nos da su longitud. Las propiedades se usan separándolas con un punto del nombre del vextor de esta forma:

```
longitud = notas_alumnos.length;
```

Una propiedad en Javascript es como una variable asociada a otra variable. length, por ejemplo, es una variable que va asociada a cualquier vector y que guarda su longitud. La sintaxis que usamos es siempre separarla por un punto como en el ejemplo que acabamos de ver. Algunas propiedades pueden ser modificadas y otras no. En este caso, por ejemplo, la propiedad length no puede modificarse. Nos dice la longitud de un vector pero no podemos modificar la longitud del vector modificando su propiedad length

Otros métodos interesantes son: **pop, push, shift y unshift.**

Pop elimina el último elemento de un vector y **push** añade un nuevo elemento al final del vector. Por el contrario, **shift** elimina el primer elemento del vector y **unshift** agrega un nuevo elemento al principio del mismo. Práctica con ellos.

Cadenas de Texto (Strings)

Las cadenas de texto en Javascript son también una clase de vectores, pero con algunas peculiaridades muy importantes. Cada posición del vector es uno de los caracteres de la letra y su longitud es la longitud de la cadena de texto. Un ejemplo:

```
texto = "OLA K ASE";
alert("La longitud de la cadena de texto es " + texto.length + "
caracteres");
```

Las cadenas de texto en Javascript están protegidas y no se pueden modificar directamente. Por ejemplo, la siguiente operación no tendría ningún efecto:

```
texto = "esto es una cadena de caracteres";
for(i = 0; i < texto.length; i++)
       texto[i]= "x";
```

Lo anterior es correcto sintácticamente y debería de dar como resultado una cadena completamente rellena de letras 'x' pero en realidad la cadena original permanecerá inalterada porque, como ya hemos dicho, está protegida contra modificaciones. Para construir cadenas de texto tendremos que usar funciones auxiliares, copias, etc. Iremos viendo diversos mecanismos en los ejemplos.

Javascript tiene muchos métodos y funciones auxiliares para manipular cadenas. También veremos algunas en los ejemplos, pero enumeraremos aquí también algunas de las más útiles:

La propiedad charAt(i) devuelve el carácter situado en la posición i de una determinada cadena de texto. Por ejemplo, para preguntar si la posición 0 de una cadena llamada texto es un espacio haríamos esto:

```
if(texto.charAt(i) == " ")
	alert("La cadena empieza por un espacio en blanco");
```

charAt() no permite usar valores negativos pero la propiedad at() también está disponible en strings y funciona igual que con cualquier otro array

ultima_letra = texto.at(-1)

Los métodos son como las propiedades pero en lugar de ser variables son funciones asociadas a una variable. La forma de usarlas es similar a las propiedades, como hemos visto en el ejemplo anterior, pero como se parecen a las funciones también llevan paréntesis, pueden requerir algún argumento en dicho paréntesis y devuelven un valor como resultado.

Los métodos toUpperCase() y toLowerCase() convierten una cadena a mayúsculas o minúsculas respectivamente.

```
texto = prompt("escribe un texto");
mayusculas = texto.toUpperCase()
document.write("tu texto escrito en mayúsculas es: " + mayusculas);
```

Cuando queremos saber el código ASCII de un carácter de una cadena usamos el método charCodeAt(i). Por ejemplo, si queremos mostrar en pantalla los códigos ASCII de todos los caracteres de una cadena podríamos hacer esto:

```
for(i = 0; i < texto.length; i++)
	document.write(texto.charCodeAt(i) + "<br/>")
```

La operación contraria (obtener el carácter definido por un determinado código ASCII) la hacemos mediante el método estático fromCharCode(i). Por ejemplo, el siguiente código genera ocho números

aleatorios entre el 33 y el 126 y escribe los caracteres que corresponden a los mismos (podría ser una forma de generar contraseñas de forma aleatoria, no? ;-))

```
for(i = 0; i  < 8; i++){
   num = Math.floor((Math.random()*94)+33);
   document.write(String.fromCharCode(num));
   }
```

fromCharCode es un método estático y esto es algo ligeramente diferente a los métodos que hemos visto hasta ahora. En lo que a nosotros respecta nos vale con saber que no va asociado a una variable en particular, sino a un tipo de variable. En este caso como es un método que va asociado a las cadenas de texto y el nombre de estas es String, es esta palabra clave la que usamos para referenciar el método.

Otras funciones útiles a conocer: **parseInt** y **parseFloat** convierten un string a su número entero o con decimales equivalente. **Number** es una función similar. Practica con ellas

Para lo contrario (convertir un número en una cadena) tenemos el método **toString** o **toFixed(n)** siendo n el número de decimales que queremos redondeando el resultado