**3.2 Manejo del tiempo en JS**

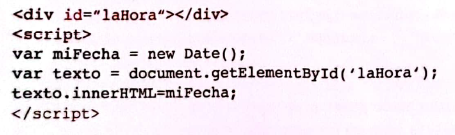
**Es mas eficiente ya que no recarga el servidor y tiene el objeto interno Date (útil para contadores, relojes, etc)**

**Se creo para almacenar fechas y horas. Una vez creado el objeto se puede modificar y realizar cálculos para modificar las fechas y horas.**

**Para crear el objeto:**

****

**Ahora miFecha tiene la fecha y hora actual. Para mostrarlo:**

****

**El campo la hora tendrá lo siguiente:**

****

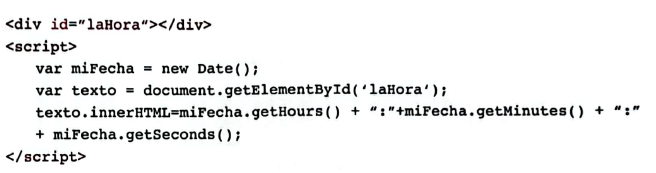
**Si queremos mostrar solo las horas, minutos o segundos:**

**getHours(). Extrae las horas del objeto Date actual.**

**getMinutes(). Extrae los minutos**

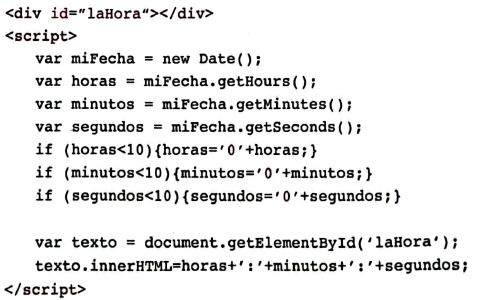
**getSeconds(). Extrae los segundos**

**Ejemplo:**

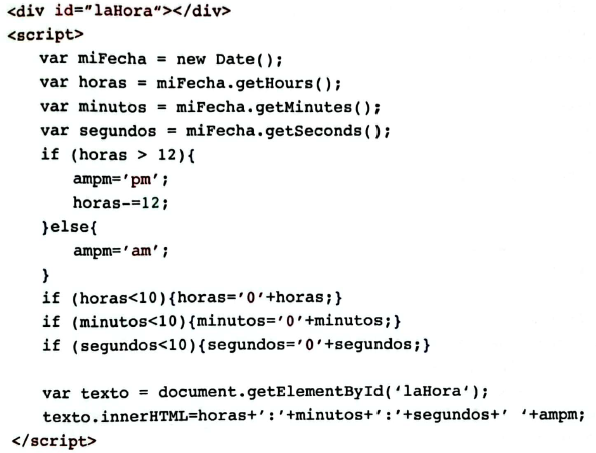
****

****

**El problema de las funciones anteriores es que no devuelven dos dígitos sin son menores de 10. Para mejorar las funciones:**

****

**Si queremos mostrar la hora en formato 12 horas:**

****

**Comprueba si son mas de las 12 o no y se ajustan las horas colocando am o pm:**

****

**3.2.1 Las funciones setTimeout y setInterval**

**setTimeout(función, milisegundos). Ejecutará la función cuando pase el tiempo indicado en el segundo parámetro. La función va sin paréntesis**

**setInterval(función, milisegundos). Se ejecuta la función de manera perodica según los milisegundos introducidos**

**clearInterval(). Para la ejecución iniciada con setInterval():**

****

**clearTimeout(). Para la ejecución iniciada con setTimeout():**

****

**En el ejemplo del reloj, la segunda función será la mas indicada para actualizar el cronometro cada segundo:**

****

**El evento onload ejecuta la sentencia setInterval programando la ejecución de la función crono cada segundo**

**3.3 Cookies**

**Sirven para recordar información del usuario. Son pequeños ficheros que almacenaran en una ruta en el equipo y contienen pares clave = valor ej:**

****

**Cuando el navegador pida una pag, las cookies asociadas a la pag se envían también en la petición y, los servidores tendrán vigilado al usuario**

**3.3.1 ¿Para qué se utilizan las cookies?**

**Monitorizar la actividad de los usuarios: patrones de actividad, gustos. Navegación, etc.**

**Para mantener opciones de visualización para el usuario. Cuando el usuario quiere ver cosas en alguna pag y otras no.**

**Almacenar variables en el lado cliente. Al terminar la sesión, los datos desaparecen de la sesión salvo que se usen cookies**

**Identificación o autenticación. Las cookies tienen caducidad, se puede jugar con eso para mantener a un usuario autenticado**

**3.3.2 ¿Cómo crear una cookie?**

****

**El valor de una cookie tiene que estar codificado:**

****

**encodeURIComponent() y decodeURIComponent() codifican y decodifican en HTML los caracteres especiales **

**las cookies tienen caducidad pero si se desea que perduren:**

****

**Ejemplo de cookie asociada a un botón:**

****

**3.3.3 ¿Cómo leer las cookies?**

**Ejemplo botón para ver las cookies:**

** para modificar una cookie basta con volver a crearla con nuevos datos. Para borrarla hay que volver a crearla con una fecha pasada. También se puede utilizar max-age:**

****

**3.4 Almacenamiento local**

**Es un sistema mas sencillo y eficiente que las cookies.**

**HTML5 pensó que el navegador puede almacenar mucha información y no sobrecargar el servidor enviando y recibiendo datos**

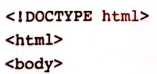
**3.4.1 el objeto localStorage**

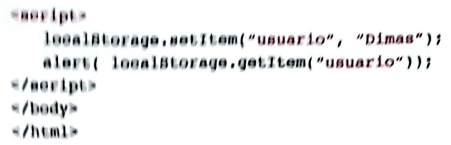
**Permite guardar y recuperar información en el navegador sin importar que sea otra sesión. Utiliza los métodos:**

**SetItem(clave,valor). Para guardar la información y tiene dos parámetros**

**getItem(). Recuperar información, hay que especificar la clave para recuperar el valor de esta.**

**Ej:**

****

****

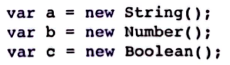
**Para eliminar un dato del almacenamiento local:**

****

**3.5 Objetos en JS**

**En js todos son objetos menos los tipos de primitivos como false, true, null, etc.**

**Aunque se desaconseja, se pueden declarar tipos de primitivos como objetos:**

****

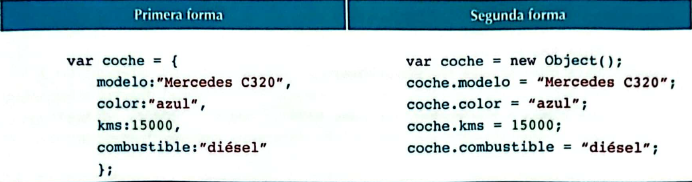
**En el siguiente código:**

****

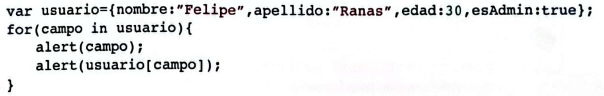
**Cadena == cadena2 es true, pero con comparación estricta (===) será false. Explicación:**

****

**Se pueden crear objetos de dos formas, la primera es más rápida:**

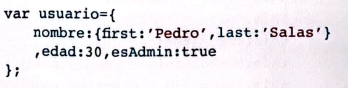
****

**3.5.1 Recorrer la información de un objeto**

****

**El navegador mostrara los mensajes con el nombre y valor de cada uno de los atributos del objeto**

**También se puede hacer asi:**

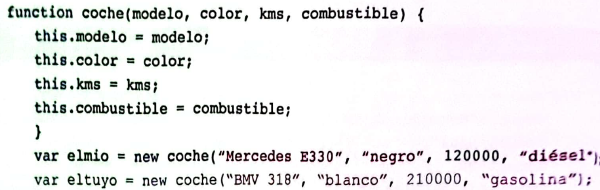
****

**Esto mostraria la misma info:**

****

**3.5.2 Constructores de JS**

**Ej constructor clase coche:**

****

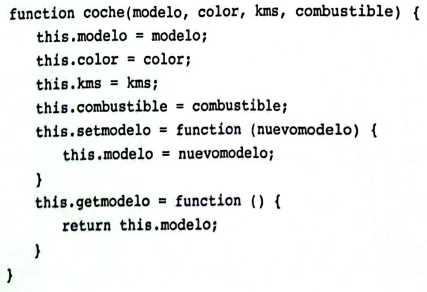
****

**En la última sentencia, JS no creara una copia del objeto, si no, que mi BMV y eltuyo apuntan al mismo objeto. Cualquier cambio de atributo repercute en la otra.**

**Js puede añadir nuevos atributos después de crear el objeto:**

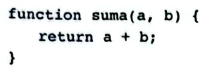
****

**Un objeto se caracteriza por tener atributos y métodos. Ej de métodos (setter y getter).**

****

**3.6 Funciones en JS**

**Las funciones en JS son objetos. Ej:**

****

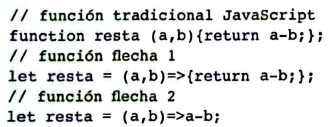
**También se puede crear a través del constructor function():**

****

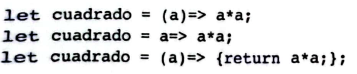
**También se puede declarar la función después de utilizarla:**

****

**La función anónima sustituye a la expresión de función tradicional:**

****

**Las tres son equivalentes. Ej de otras funciones equivalentes:**

****

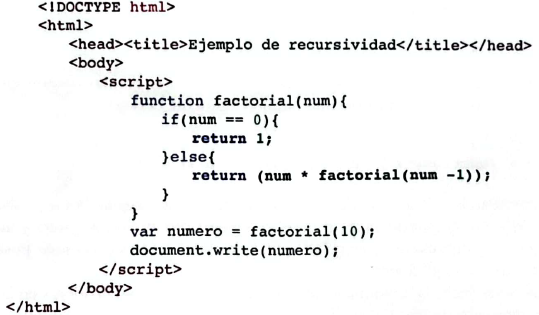
**Ej de función anónima sin parámetros:**

****

**3.6.1 La recursividad en JS**

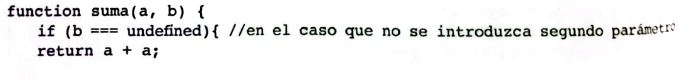
**Es una función que genera la solución realizando llamadas a si misma. La ventaja es la reducción del problema hasta llegar un punto de que no haga falta la recursividad.**

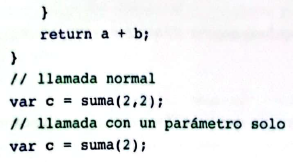
**Ej factorial de 10:**

****

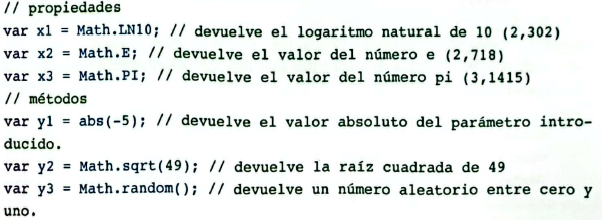
**3.6.2 Los parámetros de las funciones**

**Es posible llamar a una función con menos parámetros de los previstos. Pero hay que tener previsto esta eventualidad:**

****

****

**Para resolver problemas matemáticos tenemos el objeto Math que tiene multitud de métodos y propiedades:**

****

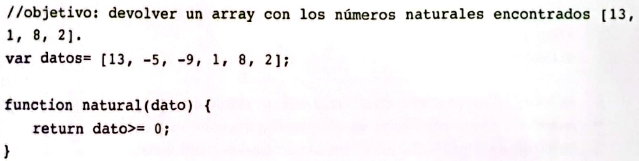
**3.6.3 Funciones y métodos en objetos de tipo array**

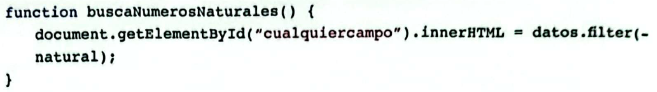
**Los Arrays son uno de los objetos con más métodos:**

**concat(). Une y devuelve una copia del array**

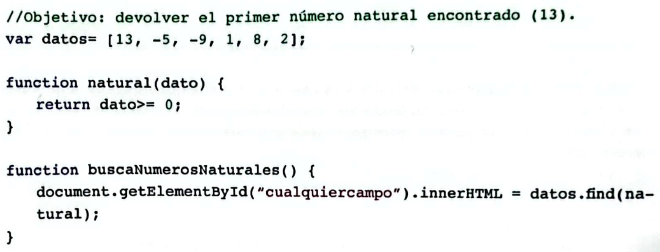
**fill(). Rellena un array con un valor suministrado**

**filter(). Devuelve un array con los elementos que pasen el test pasado por parámetro. Ej:**

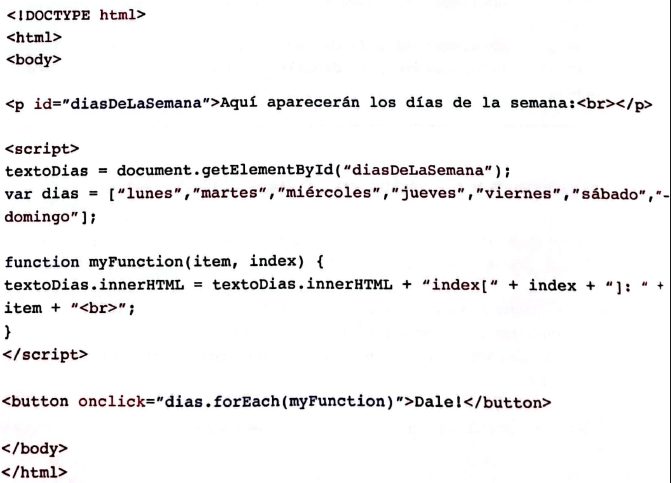
****

****

**find(). Devuelve el valor del primer elemento que pase el test. Ej:**

****

**forEach(). Llama a una función por cada uno de los elementos del array. Ej:**

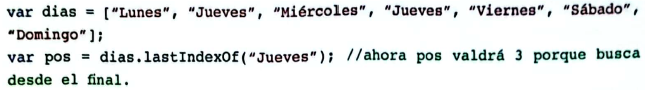
****

**includes(). Comprueba si en el array un elemento**

**indexOf(). Busca un elemento y devuelve la posición**

**isArray(). Comprueba si el objeto es un array**

**lastIndex(). Devuelve la posición de un elemento en un array, comenzando por el final.**

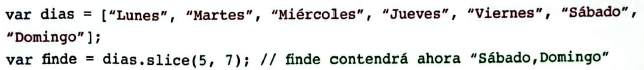
****

**pop(). Elimina un elemento de la última posición y lo devuelve**

**push(). Añade un elemento al array en su ultima posición. Devuelve la longitud del nuevo array. Con push() y pop() se hace una pila**

**shift(). Elimina el primer elemento de un array y lo devuelve**

**slice(). Selecciona parte de un array**

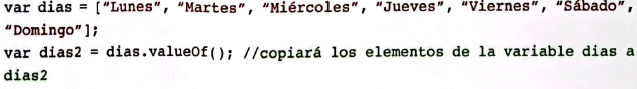
****

**sort(). Ordena el array**

**splice(). Añadir o eliminar elementos de un array**

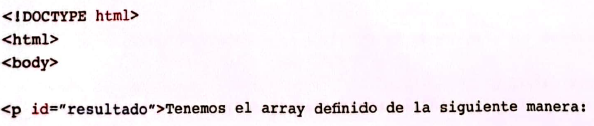
**toString(). Convierte un array en un string**

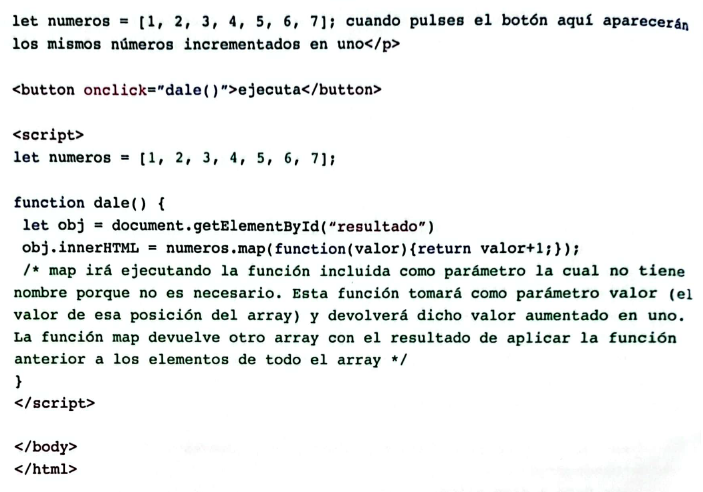
**valueOf(). Devuelve los valores de un array.**

****

**A) La función para arrays map()**

**Sirve para realizar una operación por cada elemento sin utilizar bucles. Ej:**

****

****

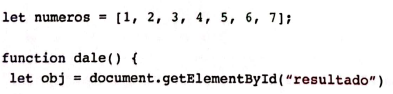
**El prototipo es:**

****

**Tiene dos parámetros. El primero es función callback que tiene tres parámetros, y el segundo parámetro es valor\_this que es opcional.**

**En la función callback el único parámetro obligatorio es elemento. (Indice indica el índice del elemento) y el array (que apunta al array) son opcionales**

**Ej que tenga en cuenta el elemento y su índice:**

****

****

**Su función es sumar a cada elemento el índice que ocupa**

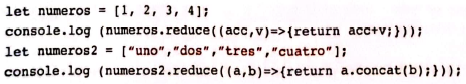
**Ej de map() combinado con la función anónima:**

****

**Se sustituye function por =>**

**B) La función para arrays reduce()**

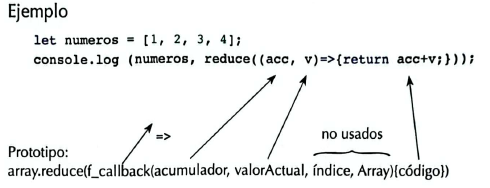
**Permite iterar un array acumulando las operaciones que se vayan haciendo sobre los elementos en otro llamado acumulador. Ej:**

****

**El primero suma todos los números del array (1+2+3+4)**

**El segundo mostrara la concatenación de todos y cada uno de los elementos = “unodostrescuatro”**

**Prototipo:**

****

**Reduce utiliza función callback y se puede hacer con función anónima**

**Usar map() para realizar una operación por cada elemento y reduce() para realizar una operación por cada elemento y acumular el resultado**

**3.6.4 Funciones y métodos en objetos y variables de tipo string**

**Tiene muchos métodos. Ej para calcular la longitud del string:**

****

**Otras funciones útiles:**

**search(string). Busca el valor y devuelve la posición donde la encontró**

**slice(inicio, fin). Extrae parte de un string y devuelve el nuevo string**

**substring(inicio,fin). Extrae parte de un string comenzando por inicio hasta fin**

**substr(inicio,longitud). Extrae caracteres comenzando en inicio con la longitud especificada**

**replace(cad1,cad2). Reemplaza en un string una cadena por otra y la devuelve**

**toUpperCase(). Convierte un string a may**

**toLowwerCase(). Convierte un string a min**

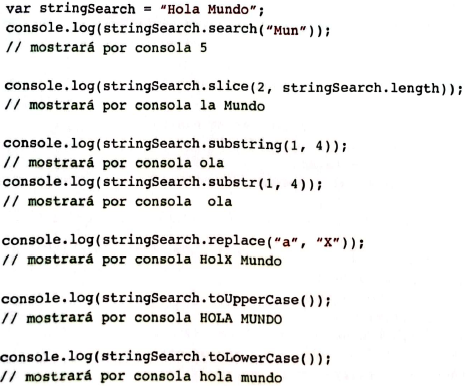
**concat(cad1, cad2). Une dos o mas cadenas y lo devuelve**

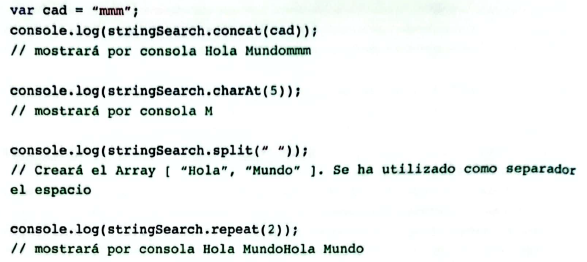
**charAt(indice). Devuelve el carácter que esta en la posición indice**

**split(separador,limite). Crea un array utilizando como separador el primer parámetro. El segundo es opcional e indica el max de elementos del array.**

**repeat(numVeces). Repite el string las veces especificadas**

**ejemplos:**

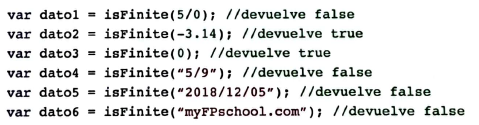
****

****

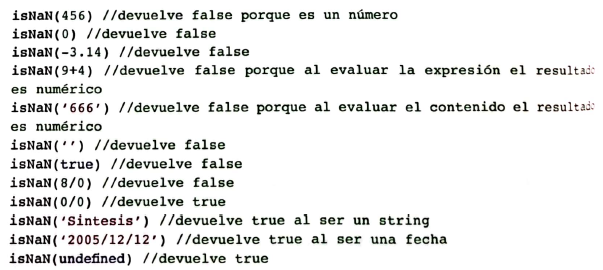
**3.6.5 funciones globales de JS relativas a números**

**Útiles para validar formularios. Ejemplos:**

**isFinite(). Evalua si el dato es finito. Devuelve true o false.**

****

**isNan(). Evalua si un dato no es numerico**

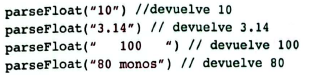
****

****

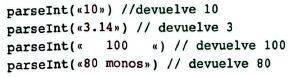
**Number(). Convierte un dato a numero**

****

**parseFloat(). Analiza un string y devuelve a float**

****

**parseInt(). Lo mismo pero a int**

****

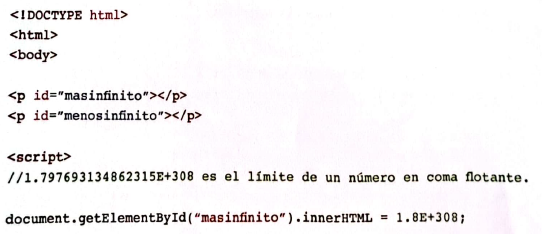
**String(). Convierte un dato a string**

****

**3.6.6 Propiedades globales de JS**

**JS tiene propiedades globales a todos los objetos:**

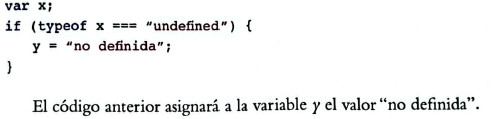
**infinity. Valor numérico que representa el infinito. (cuando se supera el limite superior e inferior float). Ej:**

****

****

**NaN. Representa un valor no numerico.**

**undefined. Indica que una variable no le ha sido asignado ningun valor. Ej:**

****

**3.6.7 Algunas funciones globales de JS**

**encodeURI(). Codifica caracteres no permitidos en URL**

**encodeURIComponent(). Codifica un componente**

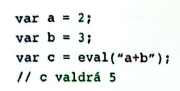
**decodeURI(). Decodifica una URI**

**decodeURI(). Decodifica un componente URI**

**Ej:**

****

**eval(): evalua un string y lo ejecuta como un código**

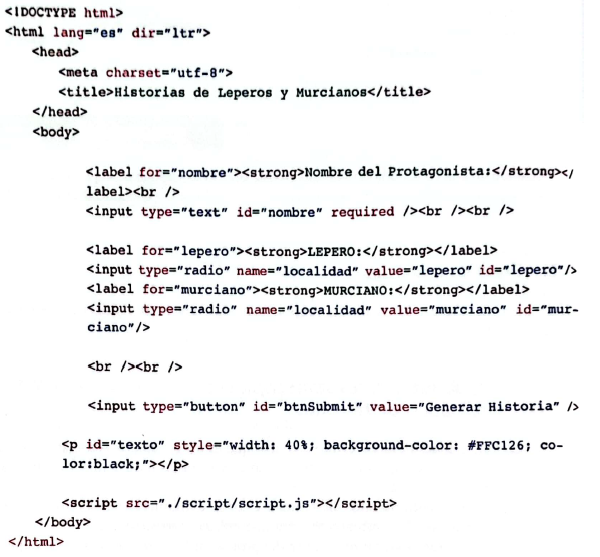
****

**3.7 Practicas guiadas**

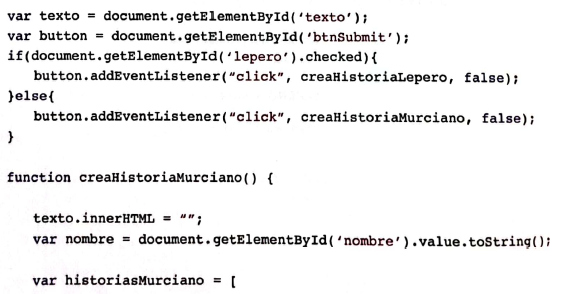
**3.7.1 uso de estructuras JS: creación de un generador de historias**

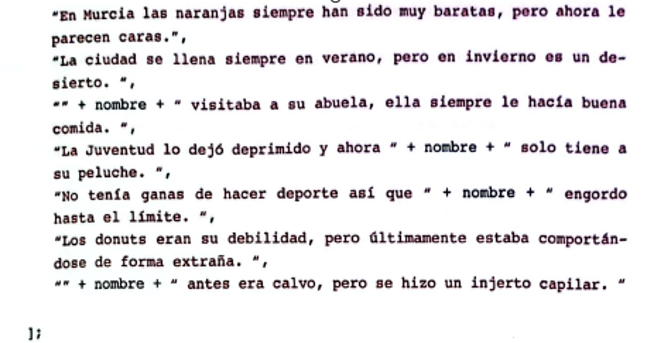
**Crear varios párrafos diferentes y estos se combinaran aleatoriamente. Tendra que aparecer el nombre y el argumento cambia según sea leperos o murcianos.**

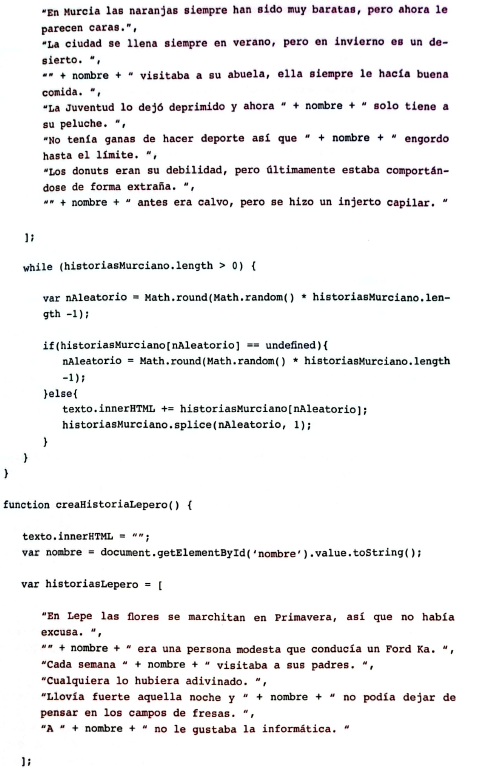
**html:**

****

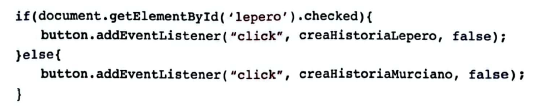
**Js:**

****

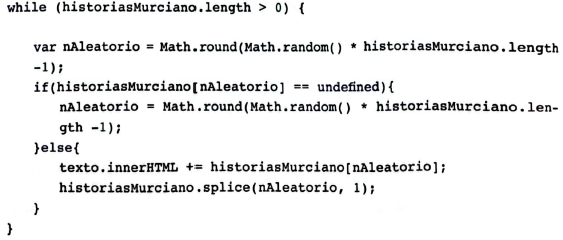
****

****

**Se asocia el evento click a una función diferente dependiendo del radiobutton:**

****

**Bucle en el array mientras que su longitud > 0:**

****

**Se genera un numero aleatorio entre 0 y el numero de elementos del array – 1. Ese número indica la posición donde se añadirá al texto. Luego se elimina esa posición:**

****

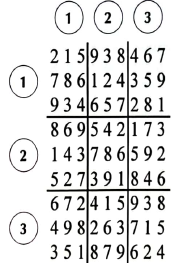
**Se elimina un elemento (segundo parametro) partiendo de la posición nAleatorio**

****

**3.7.2 Utilización avanzada de arrays: sudoku aleatorio**

**Hay dos posibilidades:**

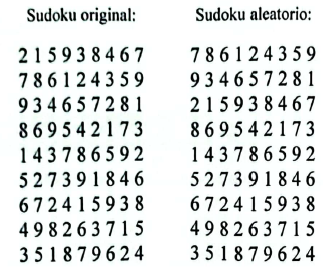
1. **Generar desde cero. Problema: hay ocasiones que no hay solución y se tiene que volver hacia atrás y volver a intentar**
2. **Generar a partir de otro ya realizado. Bastaría con intercambiar filas y columnas, de manera que los números se vayan descolocando**

****

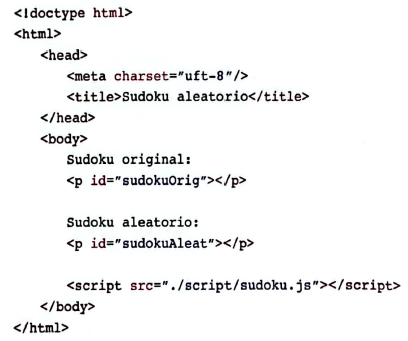
**Hay que intercambiar dos columnas que pertenezcan al mismo grupo (1, 2 o 3) y no intercambiar los de distinto grupo**

**Hay que intercambiar dos filas que pertenezcan al mismo grupo (1, 2 o 3) y no intercambiar los de distinto grupo**

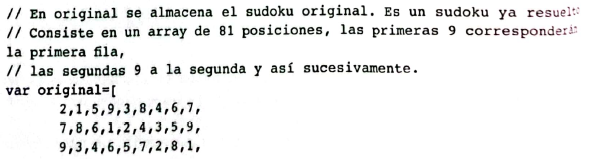
**El resultado:**

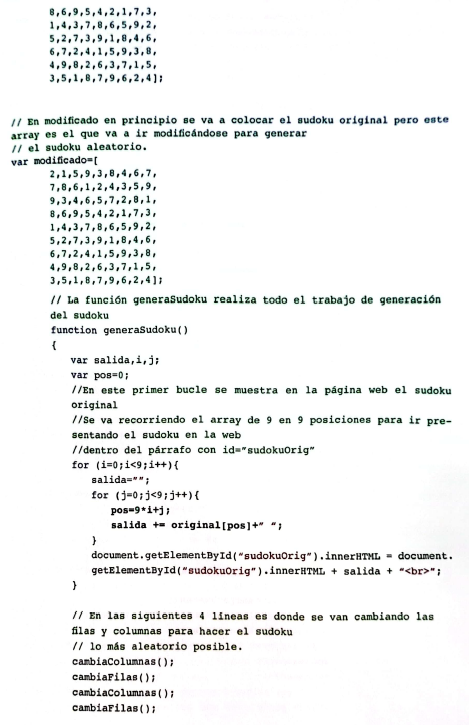
****

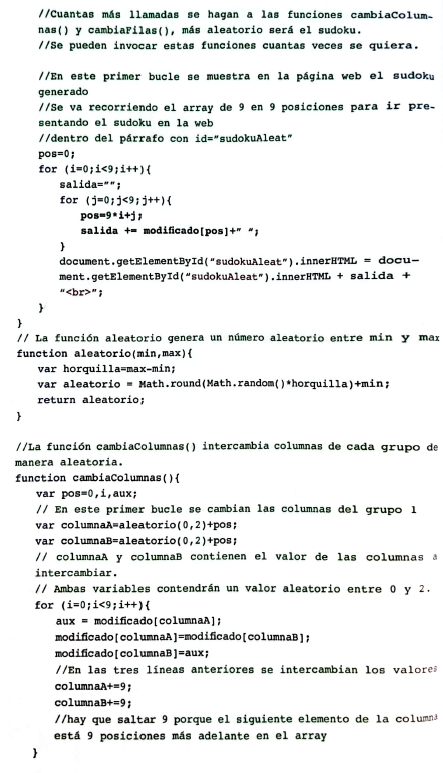
**Html:**

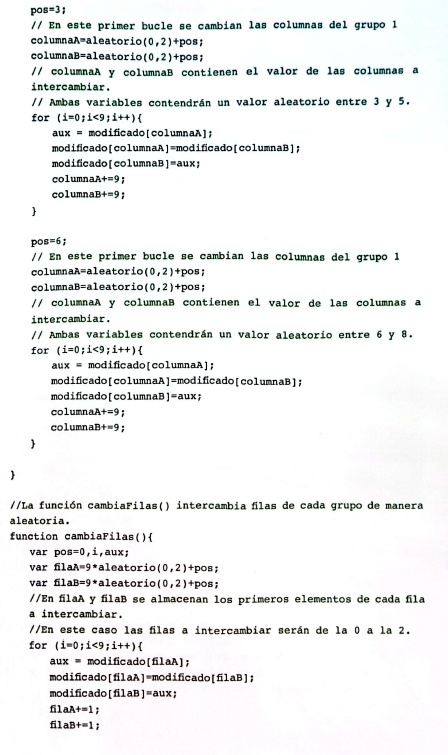
****

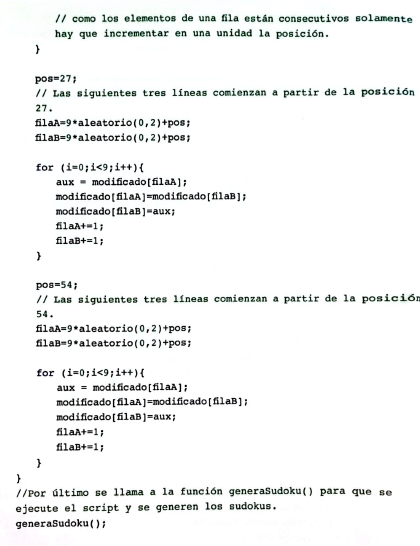
**Js:**

****

****

****

****

****