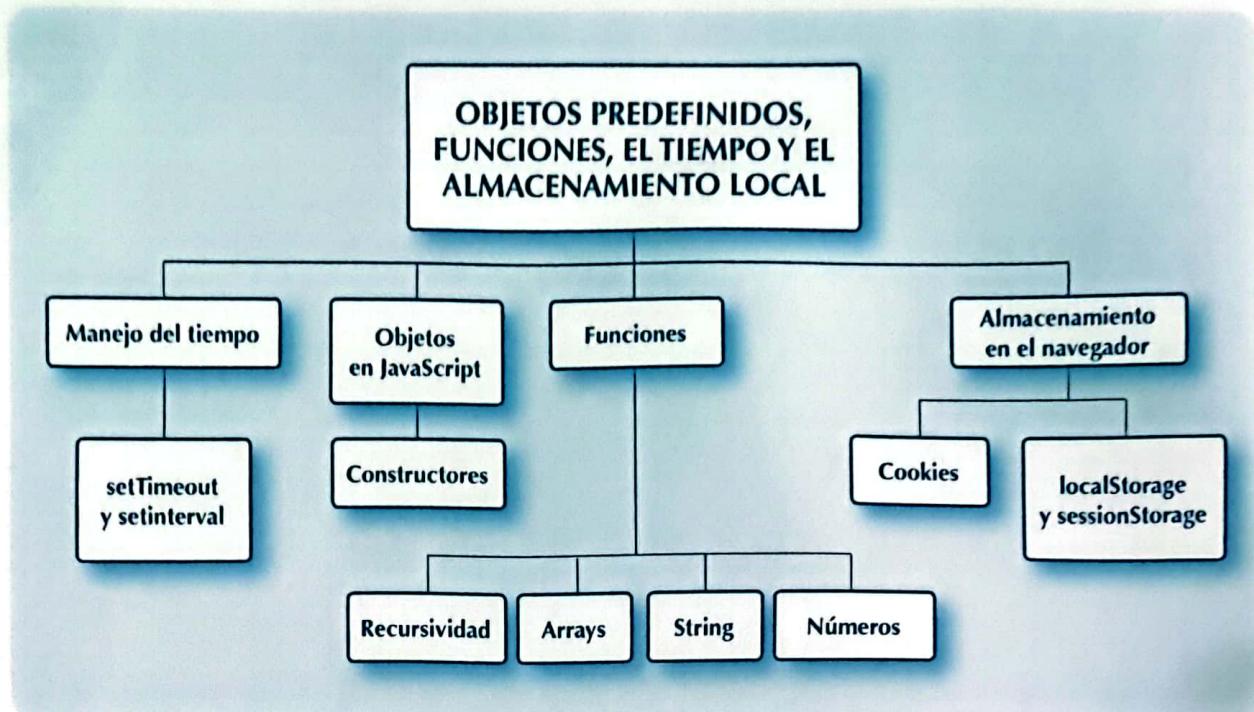


Utilización de los objetos predefinidos del lenguaje

Objetivos

- ✓ Aprender cómo se puede manejar el tiempo en JavaScript.
- ✓ Saber cómo se programa la ejecución de funciones de forma puntual y periódica en el tiempo.
- ✓ Registrar en el front-end información de navegación, usuario, etc., mediante cookies o mediante el almacenamiento local que ofrece el nuevo estándar HTML5.
- ✓ Profundizar en el concepto de objeto.
- ✓ Conocer y manejar funciones relativas al lenguaje sobre arrays, strings, números.

Mapa conceptual



Glosario

Backtracking. Estrategia utilizada en algoritmos que resuelven problemas que tienen ciertas restricciones. Este término fue creado por primera vez por el matemático D. H. Lehmer en la década de los cincuenta.

BOM (Browser Object Model). Convención específica implementada por los navegadores para que JavaScript pudiese hacer uso de sus métodos y propiedades de forma uniforme.

Expresión regular. Secuencia de caracteres que forman un patrón determinado, generalmente un patrón de búsqueda.

NaN. Propiedad que indica Not a Number (valor no numérico).

Objeto window. Aquel que soportan todos los navegadores y que representa la ventana del navegador. Se estudiará en profundidad en capítulos posteriores.

Sudoku. Juego matemático que ha sido muy popular en Japón. El objetivo es llenar una cuadrícula de 9×9 celdas dividida a su vez en 9 cuadrículas de 3×3 celdas con números diferentes del 1 al 9.

URI (Uniform Resource Identifier). Cadena de caracteres que identifica un recurso en una red de forma única. Una URI puede ser una URL, una URN o ambas.

URN. Localizador de recursos en la web que funciona de forma parecida a una URL, pero su principal diferencia es que no indica exactamente dónde se encuentra dicho objeto.

3.1. Introducción

En el capítulo 2, se había hecho un repaso de la sintaxis del lenguaje JavaScript. En este capítulo, se profundiza en conceptos como los *objetos* y las *funciones*. En JavaScript, existen muchas funciones sobre string, números, etc., que ahorran mucho tiempo al programador y que es preciso que maneje con destreza.

Otro concepto que se trabaja en este capítulo es el almacenamiento local. Se verán las cookies, que era la forma que tenía el navegador de almacenar en el equipo información del usuario, y el almacenamiento local que, a partir de HTML5, ha desbancado a las primeras.

Además de lo citado anteriormente, es importante que el programador sepa cómo manejar el tiempo en JavaScript y, para ello, se explican en profundidad el objeto Date y sus funciones asociadas, así como las funciones setTimeout y setInterval para programar la ejecución de funciones de forma puntual y periódica.

3.2. Manejo del tiempo en JavaScript

JavaScript es la manera más útil y eficiente de trabajar con el tiempo en un navegador, dado que no recarga el servidor y tiene el objeto interno Date –que puede ser de suma utilidad al programador para definir contadores, relojes, etc.–. Además, permite actualizar el reloj mediante algunas funciones que se detallarán en este capítulo.

FUNDAMENTAL

El BOM y el tiempo

El BOM (Browser Object Model), más concretamente el objeto window, proporciona los métodos setTimeout() y setInterval() que permiten manejar el tiempo dentro del navegador.

El objeto Date se creó en JavaScript para almacenar fechas y horas. Una vez creado dicho objeto, es posible modificarlo y realizar los cálculos que el programador crea convenientes para modificar las fechas y las horas.

La creación de este objeto se podría hacer de la siguiente forma:

```
var miFecha = new Date();
```

Ahora, la variable miFecha contendrá la fecha y hora actual. Si se desea mostrar la fecha y la hora actuales en un campo de una página web, es tan fácil como utilizar el siguiente código:

```
<div id="laHora"></div>
<script>
var miFecha = new Date();
var texto = document.getElementById('laHora');
texto.innerHTML=miFecha;
</script>
```

El campo laHora, en el momento de la ejecución, podrá tener un valor parecido al siguiente:

Mon Jan 07 2019 17:14:45 GMT+0100 (hora estándar de Europa central)

El problema es que el aspecto de la hora que se muestra no es el que normalmente se visualiza en una página web y, por lo tanto, hay que utilizar algunas funciones (métodos) para mostrar la hora, los minutos y los segundos como son:

- `getHours()`. Método que extrae las horas del objeto Date actual.
- `getMinutes()`. Método que extrae los minutos del objeto Date actual.
- `getSeconds()`. Método que extrae los segundos del objeto Date actual.

El siguiente ejemplo sí mostrará la hora en un formato más convencional:

```
<div id="laHora"></div>
<script>
    var miFecha = new Date();
    var texto = document.getElementById('laHora');
    texto.innerHTML=miFecha.getHours() + ":"+miFecha.getMinutes() + ":"
    + miFecha.getSeconds();
</script>
```

El resultado puede ser parecido al siguiente:

18:46:58

El problema que se puede encontrar es que las funciones anteriores (`getHours`, `getMinutes` y `getSeconds`) no devuelven dos dígitos siempre. Si, por ejemplo, la función `getSeconds` tiene que devolver tres segundos, devolverá el número 3 y no 03, con lo cual habrá que modificar o mejorar el código anterior:

```
<div id="laHora"></div>
<script>
    var miFecha = new Date();
    var horas = miFecha.getHours();
    var minutos = miFecha.getMinutes();
    var segundos = miFecha.getSeconds();
    if (horas<10){horas='0'+horas;}
    if (minutos<10){minutos='0'+minutos;}
    if (segundos<10){segundos='0'+segundos;}

    var texto = document.getElementById('laHora');
    texto.innerHTML=horas+":"+minutos+":"+segundos;
</script>
```

En este momento, el reloj implementado mostrará siempre la hora de forma más homogénea. Imagínese que se quiere mostrar la hora en formato 12 horas en vez de en el de 24, como se estaba haciendo. Habrá que modificar el código de la siguiente manera:

```

<div id="laHora"></div>
<script>
    var miFecha = new Date();
    var horas = miFecha.getHours();
    var minutos = miFecha.getMinutes();
    var segundos = miFecha.getSeconds();
    if (horas > 12){
        ampm='pm';
        horas-=12;
    }else{
        ampm='am';
    }
    if (horas<10){horas='0'+horas;}
    if (minutos<10){minutos='0'+minutos;}
    if (segundos<10){segundos='0'+segundos;}

    var texto = document.getElementById('laHora');
    texto.innerHTML=horas+':'+minutos+':'+segundos+' '+ampm;
</script>

```

Como se puede observar, lo que se hace es comprobar si son más de las 12 o no y se ajustan las horas colocando el sufijo correspondiente, ya sea *am* o *pm*.

La hora ahora podrá mostrarse en un formato parecido al siguiente:

06:46:58 pm

3.2.1. Las funciones setTimeout y setInterval

Para realizar una mejora sustancial al ejercicio desarrollado anteriormente, JavaScript ofrece las siguientes funciones:

- *setTimeout (Función_a_llamar,milisegundos)*. Ejecutará la función Función_a_llamar transcurrido el tiempo indicado en el segundo parámetro. El tiempo se expresa en milisegundos. Es importante introducir en el primer parámetro solamente el nombre de la función sin paréntesis.
- *setInterval (Función_a_llamar,milisegundos)*. Función parecida a la anterior, pero, en este caso, se ejecutará la función Función_a_llamar de manera periódica según los milisegundos introducidos en el segundo parámetro.
- *clearInterval()*. Para la ejecución iniciada con setInterval(). Véase un ejemplo de uso:

```

var elCrono = setInterval(crono, 1000);
clearInterval(elCrono);

```

- *clearTimeout()*. Para la ejecución iniciada con setTimeout(). Véase un ejemplo de uso:

```

var elTemporizador = setTimeout(laFuncion, 5000);
clearTimeout(elTemporizador);

```

En el caso del reloj que se está implementando, es obvio que la segunda función será la más indicada para actualizar el cronómetro cada segundo. El código completo de la página web quedará de la siguiente manera:

```
<!DOCTYPE html>
<html>
<head>
<title>Reloj</title>
</head>
<body>
<div id="laHora"></div>
<script>
function crono(){
var elCrono;
var miFecha = new Date();
var horas = miFecha.getHours();
var minutos = miFecha.getMinutes();
var segundos = miFecha.getSeconds();
if (horas > 12){
    ampm='pm';
    horas-=12;
} else{
    ampm='am';
}
if (horas<10){horas='0'+horas;}
if (minutos<10){minutos='0'+minutos;}
if (segundos<10){segundos='0'+segundos;}

var texto = document.getElementById('laHora');
texto.innerHTML=horas+':'+minutos+':'+segundos+' '+ampm;
}
window.onload=function(){
    elCrono = setInterval(crono,1000);
}
</script>
</body>
</html>
```

Como se puede observar, se utiliza el evento onload del objeto window, lo que implica que, en la carga de la página web, se ejecute la sentencia setInterval programando la ejecución de la función crono cada segundo.

Actividad propuesta 3.1



Basándote en el código expuesto en este apartado, crea tu propio cronómetro.

3.3. Cookies

Generalmente cuando se navega, supuestamente, es de forma anónima, pero, en muchos de los casos, esto no es necesariamente útil para el servidor. Por ejemplo, imagine que está comprando en una página web y va desplazándose de un artículo a otro y quiere mantener el carrito de la compra. Imagine que cierra el navegador por hoy y que continúa mañana con las compras.



SABÍAS QUE...

Las cookies aparecieron hace años por la necesidad que tenían las páginas web (servidores) de conocer a sus clientes (navegadores).

Un buen sistema debería almacenar todos los artículos que se han añadido al cesto de la compra para luego poder venderlos.

Por lo tanto, las cookies sirven para *recordar información del usuario*. Son pequeños ficheros que se almacenarán en una ruta determinada en el equipo y que contienen pares clave = valor como, por ejemplo:

`usuario = Dimas`

Cuando el navegador pide una página web, las cookies asociadas a dicha página web se envían también en la petición y, de esa manera, los servidores tendrán *vigilado* al usuario.

3.3.1. ¿Para qué se utilizan las cookies?

Estos son algunos ejemplos de uso de las cookies:

- *Monitorizar la actividad de los usuarios.* Esta es una de las opciones más controvertida, pues consiste en monitorizar patrones de actividad, gustos, navegación, etc. Muchas empresas utilizan esta técnica. Utilizar las cookies para estos propósitos hace que los usuarios se sientan espiados.
- *Para mantener opciones de visualización o de aspecto para el usuario.* Cuando un usuario quiere ver unas cosas en alguna página web y otras no, pueden utilizarse cookies para este propósito.
- *Almacenar variables en el lado del cliente.* Es conocido que, después de terminar la sesión, los datos desaparecen para el servidor, salvo que se utilicen cookies.
- *Identificación o autenticación.* Las cookies tienen un periodo de tiempo de validez, durante el cual se puede verificar cuándo un usuario se autentica en el sistema por primera vez. Se puede jugar con la caducidad de las cookies para mantener a un usuario autenticado y, de esa manera, mantener una utilización más eficiente de la página.

RECUERDA

- ✓ Existen límites para la longitud de las cookies, el número de cookies máximas por servidor web y el número de cookies máximas soportadas por el navegador.

3.3.2. ¿Cómo crear una cookie?

Bastaría con incluir la siguiente línea de código en un script:

```
document.cookie="usuario=Dimas";
```

El valor de la cookie siempre tiene que estar correctamente codificado, puesto que se envía en la cabecera HTTP. Por lo tanto, para evitar problemas, se debería hacer lo siguiente:

```
var usuarioCookie="Dimas"
document.cookie="usuario="+ encodeURIComponent(usuarioCookie);
```

encodeURIComponent() y decodeURIComponent() codifican y decodifican en HTML los caracteres especiales como , / ? : @ & = + \$ #.

Las cookies tienen una fecha de caducidad y, por tanto, las anteriores cookies se eliminarán al cerrarse el navegador. Si se desea que perduren se ha de añadir una fecha de caducidad:

```
document.cookie="usuario=Dimas;expires=Sat, 6 Aug 2020 12:15:00 GMT";
```

Como se puede imaginar, es posible hacer que caduque la cookie el año que viene o dentro de 5 años, así se quedará de forma "perpetua".

En el siguiente código se muestra cómo se crearía un botón que cree una cookie asociada:

```
<button type="button" onclick='document.cookie="usuario=Dimas;expires=-- Sat, 6 Aug 2020 12:15:00 GMT"'>Crear una Cookie</button>
```

3.3.3. ¿Cómo leer las cookies?

En el ejemplo del apartado 3.3.2, se ha creado un botón que, al pulsarlo, creará una cookie, pero ¿hay una manera de visualizar las cookies? En el siguiente código, se crea un botón que mostrará las cookies registradas:

```
<button type="button" onclick='alert(document.cookie)'>Ver las Cookies</button>
```

TOMA NOTA



Para modificar una cookie, basta con volver a crearla con los nuevos datos. Para borrar una cookie, solo hay que volver a crear la cookie con una fecha pasada. También se puede utilizar el parámetro max-age de la siguiente forma:

```
document.cookie = "usuario=; max-age=0";
```



Actividad propuesta 3.2

Teniendo en cuenta el código expuesto en los apartados 3.3.2 y 3.3.3, elabora una página web en la que se cree y se borre una cookie comprobando su funcionamiento en un navegador.

3.4. Almacenamiento local

Como se ha visto en el apartado 3.3, el almacenamiento en el lado del cliente solamente es posible a través de cookies. Tras aparecer HTML5, las cookies pasaron a un segundo plano para dar paso al almacenamiento local. El almacenamiento local es un sistema más sencillo y eficiente que las antiguas cookies.

HTML5 pensó que el navegador puede almacenar mucha información (como mucho 5 MB) y, de esa manera, no sobrecargar el servidor enviando y recibiendo datos. Se supone que, dentro de un dominio y un protocolo, la información es compartida por todas y cada una de las páginas.

3.4.1. El objeto localStorage

Este objeto permite guardar y recuperar información en el navegador sin importar que sea otra sesión. Su utilización es similar a las cookies y puede utilizarse con los métodos:

- setItem(clave,valor)*. Se emplea para guardar información y tiene dos parámetros, que son la clave y el valor.
- getItem(clave)*. Para recuperar información, habrá que especificar la clave para poder recuperar el valor de esta.

Véase un ejemplo de utilización de almacenamiento local con el objeto localStorage:

```
<!DOCTYPE html>
<html>
<body>
```

```
<script>
    localStorage.setItem("usuario", "Dimas");
    alert(localStorage.getItem("usuario"));
</script>
</body>
</html>
```

Para eliminar un dato del almacenamiento local, se utilizará el siguiente método:

```
localStorage.removeItem(clave);
```



SABÍAS QUE...

¿Qué ocurre si solamente se necesita que los datos se guarden durante la sesión y no de forma permanente? HTML5 ha creado el objeto sessionStorage, que se utiliza de la misma manera que localStorage, pero con la particularidad de que los datos solamente se almacenarán durante la sesión.

3.5. Objetos en JavaScript

En JavaScript, prácticamente todo es un objeto, las fechas, las expresiones regulares, los arrays, las funciones, etc. Salvo valores y tipos primitivos como false, true, null, etc., los demás componentes del lenguaje son objetos en JavaScript.

Aunque se desaconseja (porque ralentizan el código y no añaden ninguna ventaja), se pueden declarar tipos primitivos como objetos. A continuación, se muestran las variables string, number y boolean de tipo objeto:

```
var a = new String();
var b = new Number();
var c = new Boolean();
```

Por lo tanto, en el siguiente código:

```
var cadena = "Hola";
var cadena2 = new String("Hola");
```

Tanto cadena como cadena2 contendrán lo mismo y, si se hace una comparación (cadena == cadena2), el resultado será true, pero, en una comparación estricta (cadena === cadena2), el resultado será false. Esto es así porque cadena es de tipo string, mientras que cadena2 es de tipo object. Véase el siguiente código:

```
alert(typeof(cadena)); //mostrará string
alert(typeof(cadena2)); //mostrará object
```

```

<script>
    localStorage.setItem("usuario", "Dimas");
    alert( localStorage.getItem("usuario"));
</script>
</body>
</html>

```

Para eliminar un dato del almacenamiento local, se utilizará el siguiente método:

```
localStorage.removeItem(clave);
```



SABÍAS QUE...

¿Qué ocurre si solamente se necesita que los datos se guarden durante la sesión y no de forma permanente? HTML5 ha creado el objeto sessionStorage, que se utiliza de la misma manera que localStorage, pero con la particularidad de que los datos solamente se almacenarán durante la sesión.

3.5. Objetos en JavaScript

En JavaScript, prácticamente todo es un objeto, las fechas, las expresiones regulares, los arrays, las funciones, etc. Salvo valores y tipos primitivos como false, true, null, etc., los demás componentes del lenguaje son objetos en JavaScript.

Aunque se desaconseja (porque ralentizan el código y no añaden ninguna ventaja), se pueden declarar tipos primitivos como objetos. A continuación, se muestran las variables string, number y boolean de tipo objeto:

```

var a = new String();
var b = new Number();
var c = new Boolean();

```

Por lo tanto, en el siguiente código:

```

var cadena = "Hola";
var cadena2 = new String("Hola");

```

Tanto cadena como cadena2 contendrán lo mismo y, si se hace una comparación (cadena == cadena2), el resultado será true, pero, en una comparación estricta (cadena === cadena2), el resultado será false. Esto es así porque cadena es de tipo string, mientras que cadena2 es de tipo object. Véase el siguiente código:

```

alert(typeof(cadena)); //mostrará string
alert(typeof(cadena2)); //mostrará object

```

En JavaScript, pueden crearse objetos de dos formas, tal y como se señala en el cuadro 3.1. Aunque los dos códigos tienen la misma funcionalidad, el primero es más rápido, por lo tanto, es aconsejable utilizar la primera forma.

CUADRO 3.1
Objetos en JavaScript

Primera forma	Segunda forma
<pre>var coche = { modelo:"Mercedes C320", color:"azul", kms:15000, combustible:"diésel" };</pre>	<pre>var coche = new Object(); coche.modelo = "Mercedes C320"; coche.color = "azul"; coche.kms = 15000; coche.combustible = "diésel";</pre>

3.5.1. Recorrer la información de un objeto

En ocasiones, es necesario recorrer los campos de un objeto para poder procesarlos. Véase el siguiente código:

```
var usuario={nombre:"Felipe",apellido:"Ranas",edad:30,esAdmin:true};
for(campo in usuario){
    alert(campo);
    alert(usuario[campo]);
}
```

El navegador mostrará mensajes emergentes con el nombre y valor de cada uno de los atributos del objeto (nombre, Felipe, apellido, Ranas, etc.).

FUNDAMENTAL

Acceso a los campos de un objeto

Imagínese que se tiene el siguiente objeto:

```
var usuario={
    nombre:{first:'Pedro',last:'Salas'}
    ,edad:30,esAdmin:true
};
```

Las dos líneas siguientes mostrarían la misma información (Pedro):

```
alert(usuario.nombre.first);
alert(usuario['nombre']['first']);
```

La primera forma de acceso a los miembros de un objeto es parecida a la utilizada en Java.

3.5.2. Constructores de JavaScript

Hasta ahora no se habían visto métodos de objetos en JavaScript; el siguiente código crea un constructor para la clase coche:

```
function coche(modelo, color, kms, combustible) {  
    this.modelo = modelo;  
    this.color = color;  
    this.kms = kms;  
    this.combustible = combustible;  
}  
var elmio = new coche("Mercedes E330", "negro", 120000, "diésel");  
var eltuyo = new coche("BMW 318", "blanco", 210000, "gasolina");
```

Imagínese ahora el siguiente código:

```
var miBMV = eltuyo;
```

En la última sentencia del código anterior, JavaScript no creará una copia del objeto variables miBMV y eltuyo apuntarán al mismo objeto. Cualquier modificación de un atributo en una variable repercutirá en la otra.



TOMA NOTA

JavaScript puede añadir nuevos atributos, a pesar de que el objeto haya sido creado previamente. La siguiente línea de código funcionará sin problema alguno:

```
elmio.matrícula = "4321 JPH"
```

Un objeto se caracteriza por tener un estado (atributos) y un comportamiento (métodos). Estos son algunos métodos (un setter y un getter) para la clase coche creada previamente:

```
function coche(modelo, color, kms, combustible) {  
    this.modelo = modelo;  
    this.color = color;  
    this.kms = kms;  
    this.combustible = combustible;  
    this.setmodelo = function (nuevomodelo) {  
        this.modelo = nuevomodelo;  
    }  
    this.getmodelo = function () {  
        return this.modelo;  
    }  
}
```

3.6. Funciones en JavaScript

Como se ha dicho anteriormente, las funciones en JavaScript son objetos. En el capítulo 2, se explicaba la función suma, que tenía el siguiente código:

```
function suma(a, b) {
    return a + b;
}
```

A diferencia de otros lenguajes de programación, esta función se puede crear mediante un constructor llamado *Function()* de la siguiente manera:

```
var suma = new Function("a", "b", "return a + b");
```

El resultado de la siguiente línea de código será idéntico tanto si se declara la función de forma clásica como si se crea mediante un constructor:

```
var c = suma(2,2);
```

También se puede declarar la función después de utilizarla. No es lo más común en los lenguajes de programación, pero JavaScript permite crear el siguiente código:

```
var c = suma(2,2);
var suma = new Function("a", "b", "return a + b");
```

Como se puede observar, la declaración de la función se realiza después de llevar a cabo su llamada. Esto es así porque el navegador recibe todo el código JavaScript, lo interpreta y luego lo ejecuta. En lenguajes de programación compilados (no interpretados), esto no sería posible, el compilador mostrará un mensaje de error.

Por otro lado, la función flecha o anónima es muy utilizada por los programadores en la actualidad. Sustituye a la expresión de función tradicional, como se puede observar a continuación:

```
// función tradicional JavaScript
function resta (a,b){return a-b;};
// función flecha 1
let resta = (a,b)=>{return a-b;};
// función flecha 2
let resta = (a,b)=>a-b;
```

Las tres funciones anteriores son equivalentes. Como se puede observar, en las funciones dos y tres se ha sustituido la palabra reservada *function* por su equivalente inline, que es la función *arrow "=>"*.

Esta función flecha se utiliza mucho en frameworks y otro tipo de funciones para arrays, como *map()*, *filter()*, etc.

A continuación, se muestran otras tres líneas de código equivalentes. En este caso, para la función cuadrado, la cual, dado un número, devuelve el cuadrado del mismo.

```
let cuadrado = (a)=> a*a;
let cuadrado = a=> a*a;
let cuadrado = (a)=> {return a*a;};
```

En los ejemplos anteriores, las funciones siempre tenían parámetros. Un ejemplo de función flecha sin parámetros sería el siguiente:

```
let hola = ()=>"hola";
```

3.6.1. La recursividad en JavaScript

La recursividad es una función o algoritmo recursivo que genera la solución realizando llamadas a sí mismo. La ventaja de la recursividad es la reducción de la complejidad del problema hasta que se llegue a un punto en la cual ya no haga falta utilizar la recursión, puesto que la resolución es algo directo o simple.

En JavaScript, es posible utilizar la recursividad para la resolución de problemas. Como ejemplo se muestra el siguiente código, en el cual se puede observar cómo se resuelve el factorial del número 10 mediante la aplicación de la recursividad:

```
<!DOCTYPE html>
<html>
    <head><title>Ejemplo de recursividad</title></head>
    <body>
        <script>
            function factorial(num){
                if(num == 0){
                    return 1;
                }else{
                    return (num * factorial(num -1));
                }
            }
            var numero = factorial(10);
            document.write(numero);
        </script>
    </body>
</html>
```

Actividad propuesta 3.3



Teniendo en cuenta el código expuesto en este apartado, crea una función recursiva que muestre la sucesión de Fibonacci. Comprueba en un navegador que la función visualiza correctamente la secuencia deseada.

3.6.2. Los parámetros de las funciones

En JavaScript, es posible llamar a una función con menos parámetros de los previstos. No dará error en ejecución, pero hay que tener previsto en el código esta eventualidad. Véase un ejemplo de esto en la siguiente función:

```
function suma(a, b) {
    if (b === undefined){ //en el caso que no se introduzca segundo parámetro
        return a + a;
    }
```

```

    }
    return a + b;
}
// llamada normal
var c = suma(2,2);
// llamada con un parámetro solo
var c = suma(2);

```



PARA SABER MÁS

Siempre que hay que utilizar fórmulas u operaciones matemáticas, es necesario utilizar una librería u objeto matemático. En JavaScript, está el objeto Math, el cual tiene numerosos métodos y propiedades que pueden servir para ejecutar muchas de las necesidades que se tengan en una aplicación.

Véanse algunos ejemplos:

```

// propiedades
var x1 = Math.LN10; // devuelve el logaritmo natural de 10 (2,302)
var x2 = Math.E; // devuelve el valor del número e (2,718)
var x3 = Math.PI; // devuelve el valor del número pi (3,1415)
// métodos
var y1 = abs(-5); // devuelve el valor absoluto del parámetro introducido.
var y2 = Math.sqrt(49); // devuelve la raíz cuadrada de 49
var y3 = Math.random(); // devuelve un número aleatorio entre cero y uno.

```

3.6.3. Funciones y métodos en objetos de tipo array

Los arrays son uno de los objetos que tienen más métodos (funciones) para su tratamiento. A continuación, se muestran algunos de los métodos más útiles y comunes de este objeto:

- concat()*. Concatena y devuelve una copia de los arrays concatenados.
- fill()*. Rellena un array con un valor suministrado.
- filter()*. Devuelve un array con los elementos que pasen el test suministrado por parámetro. Ejemplo:

```

//objetivo: devolver un array con los números naturales encontrados [13,
1, 8, 2].
var datos= [13, -5, -9, 1, 8, 2];

function natural(dato) {
    return dato>= 0;
}

```

```

function buscaNumerosNaturales() {
    document.getElementById("cualquiercampo").innerHTML = datos.filter(
        natural);
}

```

- d) *find()*. Devuelve el valor del primer elemento que pase el test suministrado por parámetro. Ejemplo:

```

//Objetivo: devolver el primer número natural encontrado (13).
var datos= [13, -5, -9, 1, 8, 2];

function natural(dato) {
    return dato>= 0;
}

function buscaNumerosNaturales() {
    document.getElementById("cualquiercampo").innerHTML = datos.find(na-
tural);
}

```

- e) *forEach()*. Realiza una llamada a una función por cada uno de los elementos de un array. Ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<p id="diasDeLaSemana">Aquí aparecerán los días de la semana:<br></p>

<script>
textoDias = document.getElementById("diasDeLaSemana");
var dias = ["lunes","martes","miércoles","jueves","viernes","sábado","-domingo"];

function myFunction(item, index) {
    textoDias.innerHTML = textoDias.innerHTML + "index[" + index + "]: " +
    item + "<br>";
}
</script>

<button onclick="dias.forEach(myFunction)">Dale!</button>

</body>
</html>

```

- f) *includes()*. Comprueba si en el array está un elemento especificado.
g) *indexOf()*. Busca en el array un elemento y devuelve su posición.
h) *isArray()*. Comprueba si un objeto determinado es un array.

- i) `lastIndexOf()`. Devuelve la posición de un elemento de un array. Importante, este método, al contrario de lo que pudiese pensarse, comienza la búsqueda por el final. Ejemplo:

```
var dias = ["Lunes", "Jueves", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];
var pos = dias.lastIndexOf("Jueves"); //ahora pos valdrá 3 porque busca
desde el final.
```

- j) `pop()`. Elimina un elemento de la última posición de un array. Este método devuelve dicho elemento.
- k) `push()`. Añade un elemento nuevo al array en su última posición. Devuelve la longitud del nuevo array. Con los métodos `push()` y `pop()`, se puede implementar una pila.
- l) `shift()`. Elimina el primer elemento del array. Este método devuelve dicho elemento. Útil cuando se desea realizar algún proceso con los elementos de un array, pero no se desea conservarlos. También es útil para hacer una cola de elementos en combinación con `push()`.
- m) `slice()`. Selecciona parte de un array.

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];
var finde = dias.slice(5, 7); // finde contendrá ahora "Sábado,Domingo"
```

- n) `sort()`. Ordena los elementos de un array.
- o) `splice()`. Método muy útil para añadir o eliminar elementos de un array.
- p) `toString()`. Convierte un array en un string.
- q) `valueOf()`. Devuelve los valores de un array. Puede ser útil para hacer copias de un objeto array. Ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];
var dias2 = dias.valueOf(); //copiará los elementos de la variable dias a
dias2
```

A) La función para arrays `map()`

Muchas veces necesitamos realizar una operación por cada elemento de un array y la solución pasa por realizar un bucle entre todos los elementos del array e ir recuperando y ejecutando cierto código por cada elemento. ¿Sería posible hacer esto de una forma sencilla? La respuesta es sí, mediante la función `map()`.

A continuación, se muestra un código sencillo explicado paso a paso:

```
<!DOCTYPE html>
<html>
<body>

<p id="resultado">Tenemos el array definido de la siguiente manera:
```

```

let numeros = [1, 2, 3, 4, 5, 6, 7]; cuando pulses el botón aquí aparecerán
los mismos números incrementados en uno</p>

<button onclick="dale()">ejecuta</button>

<script>
let numeros = [1, 2, 3, 4, 5, 6, 7];

function dale() {
    let obj = document.getElementById("resultado")
    obj.innerHTML = numeros.map(function(valor){return valor+1;});
    /* map irá ejecutando la función incluida como parámetro la cual no tiene
    nombre porque no es necesario. Esta función tomará como parámetro valor (el
    valor de esa posición del array) y devolverá dicho valor aumentado en uno.
    La función map devuelve otro array con el resultado de aplicar la función
    anterior a los elementos de todo el array */
}
</script>

</body>
</html>

```

RECUERDA

- ✓ La función map devuelve otro array con el resultado de aplicar la función pasada como parámetro a los elementos del array. Map es la función SIN (sin bucles y sin tener que crear un array, puesto que lo hace ella sola).

El prototipo de esta función map() es el siguiente:

```
array.map(function(elemento, indice, el_array), valor_this)
```

La función map tiene dos parámetros, como se puede observar. El primero es la función callback que, a su vez, tiene tres parámetros, y el segundo parámetro es valor_this, el cual es opcional y se podrá utilizar como “this” para referenciar al objeto de la llamada.

En la función callback el único parámetro obligatorio “es elemento” (el primero). “Índice”, que indica el índice del elemento, y “el_array”, que apunta al array protagonista de la llamada, son opcionales.

Si queremos realizar un código que tenga en cuenta el elemento y su índice, un ejemplo podría ser el siguiente:

```

let numeros = [1, 2, 3, 4, 5, 6, 7];

function dale() {
    let obj = document.getElementById("resultado")

```

```
obj.innerHTML = numeros.map(function(valor,indice){return valor+indice;});
/* el resultado de esta llamada será [1,3,5,7,9,11,13] */
}
```

El anterior código es una modificación del programa ejemplo primero y su función es **sumar** a cada elemento del array el índice que ocupa en dicho array.

FUNDAMENTAL

No hace falta que utilices la función map() obligatoriamente para devolver un array. Puedes utilizarla para realizar una operación por cada elemento del array:

```
let numeros = [1, 2, 3, 4, 5, 6, 7];
numeros.map(function(valor,indice){console.log("valor:"+valor+
    indice:"+indice);});
```

Como se puede ver en el código anterior, se mostrará por consola el valor de los elementos del array y su índice:

En muchas ocasiones, lo más normal es encontrarnos con la función map() en combinación con la función flecha => o función anónima. Veamos dos líneas de código funcionalmente iguales:

```
obj.innerHTML = numeros.map(function(valor,indice){return valor+indice;});
obj.innerHTML = numeros.map((valor,indice)=>{return valor+indice;});
```

Como se puede observar, en el código anterior se ha sustituido la palabra reservada *function* por su equivalente *inline*, que es la función arrow “=>”.

La función flecha se utiliza para escribir funciones callback *ad hoc* y es muy común su utilización en muchos frameworks como ReactJS o Angular. Muchos de estos frameworks utilizan muy a menudo funciones callback. La función arrow es una forma intuitiva, fácil y rápida de programar dichos callback.

B) La función para arrays reduce()

La función reduce permite iterar un array acumulando las operaciones que se vayan haciendo sobre los elementos en otro llamado acumulador. Se ponen dos ejemplos de uso:

```
let numeros = [1, 2, 3, 4];
console.log (numeros.reduce((acc,v)=>{return acc+v;}));
let numeros2 = ["uno","dos","tres","cuatro"];
console.log (numeros2.reduce((a,b)=>{return a.concat(b);}));
```

En el primero de ellos se sumarán todos los números del array ($1 + 2 + 3 + 4$), lo cual mostrará por consola en la segunda línea el valor 10.

El segundo ejemplo mostrará por consola la concatenación de todos y cada uno de los elementos del array. El resultado será: “unodostrescuatro”.

En el siguiente esquema de la figura 3.1 se puede ver tanto el prototipo de la función reduce() como su aplicación al primer ejemplo del código anterior.

Ejemplo

```
let numeros = [1, 2, 3, 4];
console.log (numeros, reduce((acc, v)=>{return acc+v;}));
```

Prototipo:
array.reduce(f_callback(acumulador, valorActual, índice, Array){código})

Figura 3.1
Prototipo de la función reduce().

Como se puede observar, reduce también utiliza funciones callback y, por lo tanto, la función arrow ($=>$), vista anteriormente, es una candidata perfecta para utilizar.

RECUERDA

- ✓ Utiliza la función map() cuando tengas que realizar una operación por cada elemento de un array.
- Utiliza la función reduce() cuando necesites realizar una operación por cada elemento de un array y haya que ir acumulando el resultado generado.
- Utiliza la función arrow ($=>$) para funciones callback.

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

JavaScript, al igual que otros lenguajes de programación, tiene muchos métodos para manejar strings. Véase un ejemplo para calcular la longitud de un string:

```
var web = "myfpschool.com";
var longitud = web.length; //longitud valdrá 14
```

Imagínese que se quiere conocer la posición (el lugar donde comienza) de la palabra *myfpschool* dentro del string *web*:

```
var web = "La web myfpschool es una de las mejores en tecnología";
var posicion = web.lastIndexOf("myfpschool"); //posición valdrá 7
```

Imagínese que la palabra o substring (no tiene por qué ser una palabra) se repite dentro del texto varias veces. Con el método `LastIndexOf()`, puede conocerse la posición de la última ocurrencia.

Otras funciones muy útiles de los string son las siguientes:

- `search(string1)`. Busca en un string un determinado valor (o expresión regular) 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 la posición inicio y terminando en la posición fin.
- `substr(inicio, longitud)`. Extrae caracteres de un string comenzando en la posición especificada en el primer parámetro con una longitud expresada en el segundo parámetro.
- `replace(string1, string2)`. Reemplaza en un string una cadena de caracteres (o una expresión regular) por la cadena expresada en el segundo parámetro. Esta función devuelve un string con los reemplazos realizados.
- `toUpperCase()`. Convierte un string a mayúsculas.
- `toLowerCase()`. Convierte un string a minúsculas.
- `concat(string1, string2)`. Concatena dos o más cadenas de caracteres devolviendo el string resultante.
- `charAt(indice)`. Devuelve el carácter que se especifica en la posición expresada en el primer parámetro.
- `split(separador, límite)`. Crea un array de un string utilizando como separador el primer parámetro. El segundo parámetro es opcional y especifica el número máximo de elementos que puede tener dicho array. Pasado dicho número, JavaScript no creará nuevos elementos en el array.
- `repeat(numVeces)`. Repite el string del cual se invoca el método el número de veces especificado en el primer parámetro.

Véase el siguiente código con los comentarios correspondientes:

```
var stringSearch = "Hola Mundo";
console.log(stringSearch.search("Mun"));
// mostrará por consola 5

console.log(stringSearch.slice(2, stringSearch.length));
// mostrará por consola la Mundo

console.log(stringSearch.substring(1, 4));
// mostrará por consola ola
console.log(stringSearch.substr(1, 4));
// mostrará por consola ola

console.log(stringSearch.replace("a", "X"));
// mostrará por consola HolX Mundo

console.log(stringSearch.toUpperCase());
// mostrará por consola HOLA MUNDO

console.log(stringSearch.toLowerCase());
// mostrará por consola hola mundo
```

Imagínese que la palabra o substring (no tiene por qué ser una palabra) se repite dentro del texto varias veces. Con el método `LastIndexOf()`, puede conocerse la posición de la última ocurrencia. Otras funciones muy útiles de los string son las siguientes:

- `search(string1)`. Busca en un string un determinado valor (o expresión regular) 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 la posición inicio y terminando en la posición fin.
- `substr(inicio, longitud)`. Extrae caracteres de un string comenzando en la posición especificada en el primer parámetro con una longitud expresada en el segundo parámetro.
- `replace(string1, string2)`. Reemplaza en un string una cadena de caracteres (o una expresión regular) por la cadena expresada en el segundo parámetro. Esta función devuelve un string con los reemplazos realizados.
- `toUpperCase()`. Convierte un string a mayúsculas.
- `toLowerCase()`. Convierte un string a minúsculas.
- `concat(string1, string2)`. Concatena dos o más cadenas de caracteres devolviendo el string resultante.
- `charAt(indice)`. Devuelve el carácter que se especifica en la posición expresada en el primer parámetro.
- `split(separador, límite)`. Crea un array de un string utilizando como separador el primer parámetro. El segundo parámetro es opcional y especifica el número máximo de elementos que puede tener dicho array. Pasado dicho número, JavaScript no creará nuevos elementos en el array.
- `repeat(numVeces)`. Repite el string del cual se invoca el método el número de veces especificado en el primer parámetro.

Véase el siguiente código con los comentarios correspondientes:

```
var stringSearch = "Hola Mundo";
console.log(stringSearch.search("Mun"));
// mostrará por consola 5

console.log(stringSearch.slice(2, stringSearch.length));
// mostrará por consola la Mundo

console.log(stringSearch.substring(1, 4));
// mostrará por consola ola
console.log(stringSearch.substr(1, 4));
// mostrará por consola ola

console.log(stringSearch.replace("a", "X"));
// mostrará por consola HolX Mundo

console.log(stringSearch.toUpperCase());
// mostrará por consola HOLA MUNDO

console.log(stringSearch.toLowerCase());
// mostrará por consola hola mundo
```

```

var cad = "mmm";
console.log(stringSearch.concat(cad));
// mostrará por consola Hola Mundommm

console.log(stringSearch.charAt(5));
// mostrará por consola M

console.log(stringSearch.split(" "));
// Creará el Array [ "Hola", "Mundo" ]. Se ha utilizado como separador
el espacio

console.log(stringSearch.repeat(2));
// mostrará por consola Hola MundoHola Mundo

```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

Las funciones siguientes son sumamente útiles cuando se validan formularios o campos especiales. Es conocido que HTML5 ya provee al programador de ciertas validaciones nativas, pero, en ocasiones, se necesita una validación más sofisticada. Es en estas ocasiones cuando se utilizarán estas funciones.

1. `isFinite()`. Evalúa si un dato determinado es un valor finito. En caso positivo, devuelve true y, de lo contrario, devuelve false. Si se le suministra algo que no es un número o un número no finito, devuelve false.

```

var dato1 = isFinite(5/0); //devuelve false
var dato2 = isFinite(-3.14); //devuelve true
var dato3 = isFinite(0); //devuelve true
var dato4 = isFinite("5/9"); //devuelve false
var dato5 = isFinite("2018/12/05"); //devuelve false
var dato6 = isFinite("myFPschool.com"); //devuelve false

```

2. `isNaN()`. Evalúa si un dato determinado no es numérico.

```

isNaN(456) //devuelve false porque es un número
isNaN(0) //devuelve false
isNaN(-3.14) //devuelve false
isNaN(9+4) //devuelve false porque al evaluar la expresión el resultado
es numérico
isNaN('666') //devuelve false porque al evaluar el contenido el resultado
es numérico
isNaN('') //devuelve false
isNaN(true) //devuelve false
isNaN(8/0) //devuelve false
isNaN(0/0) //devuelve true
isNaN('Sintesis') //devuelve true al ser un string
isNaN('2005/12/12') //devuelve true al ser una fecha
isNaN(undefined) //devuelve true

```

```
isNaN('NaN') //devuelve true
isNaN(NaN) //devuelve true
```

3. *Number()*. Convierte un dato en un número.

```
Number(true) // devuelve 1
Number(false) // devuelve 0
Number("666") // devuelve 666
```

4. *parseFloat()*. Analiza un string y devuelve un número en coma flotante.

```
parseFloat("10") //devuelve 10
parseFloat("3.14") // devuelve 3.14
parseFloat(" 100 ") // devuelve 100
parseFloat("80 monos") // devuelve 80
```

5. *parseInt()*. Analiza un string y devuelve un número entero.

```
parseInt(`10`) //devuelve 10
parseInt(`3.14`) // devuelve 3
parseInt(` 100 `) // devuelve 100
parseInt(`80 monos`) // devuelve 80
```

6. *String()*. Convierte un dato en un string. Generalmente, se utiliza cuando se quiere convertir un dato numérico en un string.

```
String(`666`) //devuelve 666
String(3.14) //devuelve 3.14
```

3.6.6. Propiedades globales de JavaScript

JavaScript tiene propiedades globales a todos los objetos creados, como son las siguientes:

- *infinity*. Es un valor numérico que representa el infinito. Se visualiza en ocasiones cuando se supera el límite superior o inferior (*infinity* y *-infinity*) de números en coma flotante. A continuación, se muestra un ejemplo de *infinity*.

La siguiente página web:

```
<!DOCTYPE html>
<html>
<body>

<p id="masinfinito"></p>
<p id="menosinfinito"></p>

<script>
//1.797693134862315E+308 es el límite de un número en coma flotante.

document.getElementById("masinfinito").innerHTML = 1.8E+308;
```

```
document.getElementById("menosinfinito").innerHTML = -1.8E+308;  
</script>  
  
</body>  
</html>
```

Mostrará en el navegador lo siguiente:

```
Infinity  
-Infinity
```

Porque ambos valores asignados a las etiquetas `<p>` superan positiva y negativamente el límite de un número en coma flotante.

- `NaN`. La propiedad `NaN` representa un valor no numérico. En JavaScript, generalmente, se utiliza la función `isNaN()` vista anteriormente.
- `undefined`. La propiedad `undefined` indica que a una variable no le ha sido asignado ningún valor (o que no ha sido declarada). Ejemplo:

```
var x;  
if (typeof x === "undefined") {  
    y = "no definida";  
}
```

El código anterior asignará a la variable `y` el valor “no definida”.

3.6.7. Algunas funciones globales de JavaScript

JavaScript tiene funciones globales como las que se exponen a continuación:

- `encodeURI()`. Codifica a URI. Codifica caracteres que no están permitidos en una URL como `>`, `“`, `[`, etcétera.
- `encodeURIComponent()`. Codifica un componente URI. Codifica, además de los caracteres anteriores (`encodeURI`), otros como `?`, `=`, `:`, etcétera.
- `decodeURI()`. Decodifica una URI.
- `decodeURIComponent()`. Decodifica un componente URI.

Ejemplo de uso de las funciones anteriores:

```
var uri = "http://myfpschool.com";  
var uriencoded = encodeURIComponent(uri);  
var uridecoded = decodeURIComponent(uriencoded);  
  
//uriencoded contiene: http%3A%2F%2Fmyfpschool.com  
//uridecoded contiene: http://myfpschool.com
```

- `eval()`. Evalúa un string y lo ejecuta como si fuera código (un script).

Ejemplo de uso de la función eval():

```
var a = 2;
var b = 3;
var c = eval("a+b");
// c valdrá 5
```



Actividad propuesta 3.4

Teniendo en cuenta el código expuesto en este apartado, crea una página web en la que se compare el funcionamiento de las funciones encodeURI() y encodeURIComponent().

3.7. Prácticas guiadas

Con objeto de afianzar lo estudiado, proponemos dos interesantes prácticas, cuyo desarrollo queda claramente detallado.

3.7.1. Uso de estructuras JavaScript: creación de un generador automático de historias

A continuación, se muestra un divertido ejercicio que es un generador automático de historias. El objetivo es practicar todas las estructuras de JavaScript vistas hasta ahora y crear una página web como la de la figura 3.1.

Habrá que crear varios párrafos diferentes y estos se combinarán aleatoriamente (para ello, habrá que utilizar arrays). Las historias tendrán que enlazar correctamente.

También tendrá que aparecer el nombre del protagonista al menos una vez y, dependiendo de si es una historia de leperos o murcianos, el argumento tendrá que cambiar.

Otra cuestión importante es que, para dejar el ejercicio más bonito, habría que hacer que el aspecto de la solución y del ejemplo sean lo más parecidos.

Nombre del protagonista:
Gonzalo
LEPERO <input checked="" type="radio"/> MURCIANO <input type="radio"/>
Generar una historia aleatoria
Estaban a 44 grados centígrados afuera, así que Mariano el Chapas se fue a dar un paseo. Cuando llegaron a la cocina, se quedaron mirando horrorizados durante unos instantes, luego se comieron un potaje y se echaron una siesta. Gonzalo lo vio todo, pero no se sorprendió: su padre es capaz de comerse una vaca en monopatín.

Figura 3.2
Generador de historias.

Archivo index.html:

```

<!DOCTYPE html>
<html lang="es" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Historias de Leperos y Murcianos</title>
    </head>
    <body>

        <label for="nombre"><strong>Nombre del Protagonista:</strong></
        label><br />
        <input type="text" id="nombre" required /><br /><br />

        <label for="lepero"><strong>LEPERO:</strong></label>
        <input type="radio" name="localidad" value="lepero" id="lepero"/>
        <label for="murciano"><strong>MURCIANO:</strong></label>
        <input type="radio" name="localidad" value="murciano" id="mur-
        ciano"/>

        <br /><br />

        <input type="button" id="btnSubmit" value="Generar Historia" />

        <p id="texto" style="width: 40%; background-color: #FFC126; co-
        lor:black;"></p>

        <script src=".//script/script.js"></script>
    </body>
</html>

```

Archivo script.js:

```

var texto = document.getElementById('texto');
var button = document.getElementById('btnSubmit');
if(document.getElementById('lepero').checked){
    button.addEventListener("click", creaHistoriaLepero, false);
} else{
    button.addEventListener("click", creaHistoriaMurciano, false);
}

function creaHistoriaMurciano() {

    texto.innerHTML = "";
    var nombre = document.getElementById('nombre').value.toString();

    var historiasMurciano = [

```

"En Murcia las naranjas siempre han sido muy baratas, pero ahora le parecen caras.",
"La ciudad se llena siempre en verano, pero en invierno es un desierto. ",
"" + nombre + " visitaba a su abuela, ella siempre le hacía buena comida. ",
"La Juventud lo dejó deprimido y ahora " + nombre + " solo tiene a su peluche. ",
"No tenía ganas de hacer deporte así que " + nombre + " engordó hasta el límite. ",
"Los donuts eran su debilidad, pero últimamente estaba comportándose de forma extraña. ",
"" + nombre + " antes era calvo, pero se hizo un injerto capilar. "

};

```
while (historiasMurciano.length > 0) {  
  
    var nAleatorio = Math.round(Math.random() * historiasMurciano.length -1);  
  
    if(historiasMurciano[nAleatorio] == undefined){  
        nAleatorio = Math.round(Math.random() * historiasMurciano.length  
        -1);  
    }else{  
        texto.innerHTML += historiasMurciano[nAleatorio];  
        historiasMurciano.splice(nAleatorio, 1);  
    }  
}  
}
```

```
function creaHistoriaLepero() {
```

```
    texto.innerHTML = "";  
    var nombre = document.getElementById('nombre').value.toString();  
  
    var historiasLepero = [  
  
        "En Lepe las flores se marchitan en Primavera, así que no había  
        excusa. ",  
        "" + nombre + " era una persona modesta que conducía un Ford Ka. ",  
        "Cada semana " + nombre + " visitaba a sus padres. ",  
        "Cualquiera lo hubiera adivinado. ",  
        "Llovía fuerte aquella noche y " + nombre + " no podía dejar de  
        pensar en los campos de fresas. ",  
        "A " + nombre + " no le gustaba la informática. "  
    ];
```

```

        while (historiasLepero.length > 0) {

            var nAleatorio = Math.round(Math.random() * historiasLepero.length
                -1);

            if(historiasLepero[nAleatorio] == undefined){
                nAleatorio = Math.round(Math.random() * historiasLepero.length
                -1);
            }else{
                texto.innerHTML += historiasLepero[nAleatorio];
                historiasLepero.splice(nAleatorio, 1);
            }
        }
    }
}

```

A continuación se comentarán las partes del código más relevantes:

```

if(document.getElementById('lepero').checked){
    button.addEventListener("click", creaHistoriaLepero, false);
}else{
    button.addEventListener("click", creaHistoriaMurciano, false);
}

```

En las sentencias anteriores, se asocia el evento click del botón que tiene la página a una función diferente dependiendo si se ha seleccionado el radiobutton “lepero” o “murciano”.

```

while (historiasMurciano.length > 0) {

    var nAleatorio = Math.round(Math.random() * historiasMurciano.length
        -1);
    if(historiasMurciano[nAleatorio] == undefined){
        nAleatorio = Math.round(Math.random() * historiasMurciano.length
        -1);
    }else{
        texto.innerHTML += historiasMurciano[nAleatorio];
        historiasMurciano.splice(nAleatorio, 1);
    }
}

```

En el código anterior, se va a hacer un bucle en el array definido anteriormente (en este caso, historiasMurciano), mientras que su longitud sea mayor que 0 (que tenga algún elemento).

Con la sentencia:

```
var nAleatorio = Math.round(Math.random() * historiasMurciano.length -1);
```

Se genera un número aleatorio entre 0 y el número de elementos del array menos 1. Este número indicará una posición del array, la cual se va a añadir al texto de la página web. Posteriormente, se elimina esa posición del array con la sentencia:

```
historiasMurciano.splice(nAleatorio, 1);
```

En la sentencia anterior, se elimina un elemento del array (segundo parámetro) partiendo de la posición nAleatorio.

En este ejercicio, HTML y JavaScript estarán en ficheros separados. El código JavaScript estará en una carpeta llamada *script*. Para una carga rápida, se colocará la llamada al script al final del fichero HTML.



Actividad propuesta 3.5

Basándote en el ejemplo expuesto en este apartado, crea tu propio generador automático de historias.

3.7.2. Utilización avanzada de arrays: generación de un sudoku aleatorio

Para generar un sudoku aleatorio, hay dos posibilidades:

1. *Generar un sudoku desde cero*. El problema que tiene esta técnica es que, en ocasiones, al crear el sudoku, se encuentran situaciones en las que no hay una solución y el algoritmo tiene que volver hacia atrás (backtracking) y volver a intentar generar una solución válida.
2. *Generar un sudoku a partir de otro ya realizado*. Es la más sencilla. Bastaría con intercambiar filas y columnas con cierto criterio, de tal manera que los números se vayan descolocando y el sudoku vaya siendo más y más aleatorio.

1	2	3
2 1 5 9 3 8 4 6 7		
1	7 8 6 1 2 4 3 5 9	
9 3 4 6 5 7 2 8 1		
8 6 9 5 4 2 1 7 3		
2	1 4 3 7 8 6 5 9 2	
5 2 7 3 9 1 8 4 6		
6 7 2 4 1 5 9 3 8		
3	4 9 8 2 6 3 7 1 5	
3 5 1 8 7 9 6 2 4		

Figura 3.3
Filas y columnas de un sudoku agrupadas (tres grupos por fila y columna).

Para intercambiar las columnas, hay que tener en cuenta lo siguiente: hay que intercambiar dos columnas que pertenezcan al mismo grupo (1, 2 o 3) y no se pueden intercambiar columnas de distinto grupo.

Para intercambiar las filas, hay que tener en cuenta lo siguiente: hay que intercambiar dos filas que pertenezcan al mismo grupo (1, 2 o 3) y no se pueden intercambiar filas de distinto grupo.

El resultado del ejercicio tendrá que ser algo parecido al de la figura 3.3. Cada vez que refresque la página el sudoku aleatorio será diferente.

Sudoku original:

2	1	5	9	3	8	4	6	7
7	8	6	1	2	4	3	5	9
9	3	4	6	5	7	2	8	1
8	6	9	5	4	2	1	7	3
1	4	3	7	8	6	5	9	2
5	2	7	3	9	1	8	4	6
6	7	2	4	1	5	9	3	8
4	9	8	2	6	3	7	1	5
3	5	1	8	7	9	6	2	4

Sudoku aleatorio:

7	8	6	1	2	4	3	5	9
9	3	4	6	5	7	2	8	1
2	1	5	9	3	8	4	6	7
8	6	9	5	4	2	1	7	3
1	4	3	7	8	6	5	9	2
5	2	7	3	9	1	8	4	6
6	7	2	4	1	5	9	3	8
4	9	8	2	6	3	7	1	5
3	5	1	8	7	9	6	2	4

Figura 3.4

Resultado del ejercicio resuelto de un sudoku original y de un sudoku aleatorio generado.

Véase el fichero HTML del programa (fichero sudoku.html):

```
<!doctype html>
<html>
    <head>
        <meta charset="uft-8"/>
        <title>Sudoku aleatorio</title>
    </head>
    <body>
        Sudoku original:
        <p id="sudokuOrig"></p>

        Sudoku aleatorio:
        <p id="sudokuAleat"></p>

        <script src=".//script/sudoku.js"></script>
    </body>
</html>
```

A continuación, se muestra el archivo donde se almacena todo el código JavaScript (ficher sudoku.js). El fichero se ha comentado para una mejor comprensión:

```
// En original se almacena el sudoku original. Es un sudoku ya resuelto
// Consiste en un array de 81 posiciones, las primeras 9 corresponderán
// a la primera fila,
// las segundas 9 a la segunda y así sucesivamente.
var original=[
    2,1,5,9,3,8,4,6,7,
    7,8,6,1,2,4,3,5,9,
    9,3,4,6,5,7,2,8,1,
```

```

8,6,9,5,4,2,1,7,3,
1,4,3,7,8,6,5,9,2,
5,2,7,3,9,1,8,4,6,
6,7,2,4,1,5,9,3,8,
4,9,8,2,6,3,7,1,5,
3,5,1,8,7,9,6,2,4];

// En modificado en principio se va a colocar el sudoku original pero este
array es el que va a ir modificándose para generar
// el sudoku aleatorio.
var modificado=[
    2,1,5,9,3,8,4,6,7,
    7,8,6,1,2,4,3,5,9,
    9,3,4,6,5,7,2,8,1,
    8,6,9,5,4,2,1,7,3,
    1,4,3,7,8,6,5,9,2,
    5,2,7,3,9,1,8,4,6,
    6,7,2,4,1,5,9,3,8,
    4,9,8,2,6,3,7,1,5,
    3,5,1,8,7,9,6,2,4];

// La función generaSudoku realiza todo el trabajo de generación
// del sudoku
function generaSudoku()
{
    var salida,i,j;
    var pos=0;
    //En este primer bucle se muestra en la página web el sudoku
    //original
    //Se va recorriendo el array de 9 en 9 posiciones para ir presentando el sudoku en la web
    //dentro del párrafo con id="sudokuOrig"
    for (i=0;i<9;i++){
        salida="";
        for (j=0;j<9;j++){
            pos=9*i+j;
            salida += original[pos]+" ";
        }
        document.getElementById("sudokuOrig").innerHTML = document.
        getElementById("sudokuOrig").innerHTML + salida + "<br>";
    }

    // En las siguientes 4 líneas es donde se van cambiando las
    // filas y columnas para hacer el sudoku
    // lo más aleatorio posible.
    cambiaColumnas();
    cambiaFilas();
    cambiaColumnas();
    cambiaFilas();
}

```

```

//Cuantas más llamadas se hagan a las funciones cambiaColumnas() y cambiaFilas(), más aleatorio será el sudoku.
//Se pueden invocar estas funciones cuantas veces se quiera.

//En este primer bucle se muestra en la página web el sudoku
generado
//Se va recorriendo el array de 9 en 9 posiciones para ir presentando el sudoku en la web
//dentro del párrafo con id="sudokuAleat"
pos=0;
for (i=0;i<9;i++){
    salida="";
    for (j=0;j<9;j++){
        pos=9*i+j;
        salida += modificado[pos]+" ";
    }
    document.getElementById("sudokuAleat").innerHTML = document.getElementById("sudokuAleat").innerHTML + salida +
    "<br>";
}
// La función aleatorio genera un número aleatorio entre min y max
function aleatorio(min,max){
    var horquilla=max-min;
    var aleatorio = Math.round(Math.random()*horquilla)+min;
    return aleatorio;
}

//La función cambiaColumnas() intercambia columnas de cada grupo de manera aleatoria.
function cambiaColumnas(){
    var pos=0,i,aux;
    // En este primer bucle se cambian las columnas del grupo 1
    var columnaA=aleatorio(0,2)+pos;
    var columnaB=aleatorio(0,2)+pos;
    // columnaA y columnaB contienen el valor de las columnas a intercambiar.
    // Ambas variables contendrán un valor aleatorio entre 0 y 2.
    for (i=0;i<9;i++){
        aux = modificado[columnaA];
        modificado[columnaA]=modificado[columnaB];
        modificado[columnaB]=aux;
        //En las tres líneas anteriores se intercambian los valores
        columnaA+=9;
        columnaB+=9;
        //hay que saltar 9 porque el siguiente elemento de la columna
        está 9 posiciones más adelante en el array
    }
}

```

```

pos=3;
// En este primer bucle se cambian las columnas del grupo 1
columnaA=aleatorio(0,2)+pos;
columnaB=aleatorio(0,2)+pos;
// columnaA y columnaB contienen el valor de las columnas a
intercambiar.
// Ambas variables contendrán un valor aleatorio entre 3 y 5.
for (i=0;i<9;i++){
    aux = modificado[columnaA];
    modificado[columnaA]=modificado[columnaB];
    modificado[columnaB]=aux;
    columnaA+=9;
    columnaB+=9;
}

pos=6;
// En este primer bucle se cambian las columnas del grupo 1
columnaA=aleatorio(0,2)+pos;
columnaB=aleatorio(0,2)+pos;
// columnaA y columnaB contienen el valor de las columnas a
intercambiar.
// Ambas variables contendrán un valor aleatorio entre 6 y 8.
for (i=0;i<9;i++){
    aux = modificado[columnaA];
    modificado[columnaA]=modificado[columnaB];
    modificado[columnaB]=aux;
    columnaA+=9;
    columnaB+=9;
}

}

//La función cambiaFilas() intercambia filas de cada grupo de manera
aleatoria.

function cambiaFilas(){
    var pos=0,i,aux;
    var filaA=9*aleatorio(0,2)+pos;
    var filaB=9*aleatorio(0,2)+pos;
    //En filaA y filaB se almacenan los primeros elementos de cada fila
    a intercambiar.
    //En este caso las filas a intercambiar serán de la 0 a la 2.
    for (i=0;i<9;i++){
        aux = modificado[filaA];
        modificado[filaA]=modificado[filaB];
        modificado[filaB]=aux;
        filaA+=1;
        filaB+=1;
    }
}

```

```
// como los elementos de una fila están consecutivos solamente  
// hay que incrementar en una unidad la posición.  
}  
  
pos=27;  
// Las siguientes tres líneas comienzan a partir de la posición  
27.  
filaA=9*aleatorio(0,2)+pos;  
filaB=9*aleatorio(0,2)+pos;  
  
for (i=0;i<9;i++){  
    aux = modificado[filaA];  
    modificado[filaA]=modificado[filaB];  
    modificado[filaB]=aux;  
    filaA+=1;  
    filaB+=1;  
}  
  
pos=54;  
// Las siguientes tres líneas comienzan a partir de la posición  
54.  
filaA=9*aleatorio(0,2)+pos;  
filaB=9*aleatorio(0,2)+pos;  
  
for (i=0;i<9;i++){  
    aux = modificado[filaA];  
    modificado[filaA]=modificado[filaB];  
    modificado[filaB]=aux;  
    filaA+=1;  
    filaB+=1;  
}  
}  
//Por último se llama a la función generaSudoku() para que se  
ejecute el script y se generen los sudokus.  
generaSudoku();
```

Actividad propuesta 3.6



Copia y ejecuta el código expuesto en este apartado y comprueba que funciona correctamente. Como habrás deducido del código, cuando se van a elegir dos filas o dos columnas de forma aleatoria, en ocasiones, las columnas o filas que han de intercambiarse son la misma. Mejora el código para que, cuando se elija una fila o columna de cada grupo, nunca se repitan.

Resumen

- El objeto Date() tiene los métodos getHours(), getMinutes() y getSeconds() para poder mostrar la hora del modo en que el programador desee.
- setTimeout() y setInterval() permiten al programador llamar una función en un momento determinado o poder ejecutarla repetidamente cada x milisegundos.
- Las cookies son una manera clásica de almacenar información local del usuario.
- Para modificar una cookie, basta con recrearla de nuevo y, para borrarla, con establecer una fecha de expiración pasada sería suficiente.
- El objeto localStorage con sus métodos.setItem(), removeItem() y getItem() es una versión más nueva de almacenamiento local que las cookies.
- Si solamente se desean mantener los valores durante una sesión, se puede utilizar el objeto sessionStorage, que funciona de la misma manera que el objeto localStorage.
- Para recorrer los campos de un objeto, se puede utilizar el bucle for(campo in nombreObjeto){ ... }.
- En JavaScript, se pueden crear métodos en los objetos como los setters y getters, por ejemplo. Los métodos en JavaScript no son tan utilizados como sus homólogos en Java.
- En JavaScript, es posible utilizar recursividad. No obstante, dado que se ejecuta en un navegador, la velocidad de procesamiento puede no ser alta.
- Se pueden realizar llamadas a las funciones con menos parámetros de los definidos. En ese caso, los parámetros no definidos tendrán valor undefined.
- El objeto Math tiene funciones y variables muy útiles cuando se necesita utilizar operaciones matemáticas complejas.
- Existen numerosas funciones para objetos de tipo array como concat(), fill(), filter(), forEach(), etcétera.
- Existen numerosas funciones para objetos de tipo string como search(), slice(), replace(), split(), etcétera.
- Los números en JavaScript tienen funciones que permiten su validación y conversión como isFinite(), isNaN(), Number(), parseFloat() y parseInt().

Ejercicios propuestos



1. Lleva a cabo un ejercicio que muestre un enlace a Myfpschool.com si sacando un número de forma aleatoria es menor que 0,5. En caso contrario, que salude con buenos días (antes de las 15:00) o buenas tardes dependiendo de la hora en la que se cargue la página. Utiliza el método Math.random() y el atributo document.getElementById("id").innerHTML para realizar el ejercicio.
2. Borrando atributos de un objeto:

```
function coche(modelo, color, kms, combustible) {  
    this.modelo = modelo;
```

```

        this.color = color;
        this.kms = kms;
        this.combustible = combustible;
    }
var elmio = new coche("Mercedes E330", "negro", 120000, "diésel");
var eltuyo = new coche("BMW 318", "blanco", 210000, "gasolina");

```

En el ejemplo anterior, ¿se puede borrar un atributo con la siguiente sentencia?:

```
delete elmio.matrícula;
```

Crea un programa que pruebe si esto es posible.

3. Realiza un script que, cuando pulse un botón, te diga la última posición de una determinada palabra dentro de una frase. El script solamente se ejecutará cuando se pulse el botón.

Imagínate que la palabra o substring (no tiene por qué ser una palabra) se repite dentro del texto varias veces. Con el método LastIndexOf(), puede conocerse la posición de la última ocurrencia.

4. Antonio está mostrando los mensajes relativos a la Navidad depurando código y utilizando funciones con string:

```

<html>
<body>
<ul>
</ul>
<script>
var list = document.querySelector('ul');
list.innerHTML = "";
var saludos = ['¡Feliz cumpleaños!', 'Feliz navidades a todos', 'Te
deseo una feliz navidad', 'En Navidades nos vamos de fiesta',
'Pasa un buen fin de semana'];

for(var i = 0; i < saludos.length; i++) {
    var input = saludos[i];
    if(saludos[i].indexOf('Navidad') !== -1) {
        var result = input;
        var listItem = document.createElement('li');
        listItem.textContent = result;
        list.appendChild(listItem);
    }
}
</script>
</body>
</html>

```

Ten en cuenta la web anterior, pero sin que realice lo que Antonio el programador desea. Échale una mano y modifica el código para que muestre correctamente todos los mensajes relativos a las Navidades.

5. Repite el ejercicio 4, pero consiguiendo que no salga ningún mensaje que contenga la palabra **fiesta**.
6. Mariano tiene un problema con la siguiente página web, cuando quiere mostrar los nombres de los equipos correctamente, le sale una serie de números que no desea. Por ejemplo:

MAN : Manchester United

Como no ha terminado todavía el ciclo de DAW, te ha pedido que le eches una mano para arreglar la página web.

```
<html>
<body>
<ul>
</ul>
<script>
var list = document.querySelector('ul');
list.innerHTML = "";
var equipos = [ 'MAN675847583748sjt567654;Manchester United',
'RMD576746573fhdg4737dh4;Real Madrid',
'LIV5hg65hd737456236dch46dg4;Liverpool FC',
'SEV4f65hf75f736463;Sevilla FC',
'BAR5767ghtyfyr4536dh45dg45dg3;Barcelona FC'];

for (var i = 0; i < equipos.length; i++) {
var input = equipos[i];
// escribe el código que haga que funcione aquí debajo
var result = input;
var listItem = document.createElement('li');
listItem.textContent = result;
list.appendChild(listItem);
}
</script>
</body>
</html>
```

7. En el código siguiente, se presenta un objeto del tipo persona que tiene una serie de atributos (name, age, gender e interests) y dos métodos: saluda y bio.

- a) Prueba el código y comprueba que funciona.
- b) Crea dos botones biografía y saluda y haz que cada uno llame al método correspondiente y muestre el resultado en la página web.

```
<html>
<body>
<script>
```

```

var persona = {
    name: ['Rafa', 'Nadal'],
    age: 30,
    gender: 'masculino',
    interests: ['tenis', 'futbol'],
    bio: function() {
        alert(this.name[0] + ' ' + this.name[1] + ' tiene ' + this.age
        + ' años y le gusta el ' + this.interests[0] + ' y el ' + this.
        interests[1] + '..');
    },
    saluda: function() {
        alert('Hola, me llamo ' + this.name[0] + '..');
    }
};
</script>
</body>
</html>

```

8. Benito no sabe si el código siguiente funciona ni para qué sirve:

```

var perrillos = ["Rocket","Flash","Bella","Slugger"];
Console.log(perrillos.toString());
var ciudades = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';
perrillos = ciudades.split(',');
Console.log(perrillos.toString());

```

- a) Explícale si funciona correctamente (corrige lo que esté mal) y realiza un procedimiento que elimine del array perrillos todos los elementos que contengan una C sin importar si está en mayúscula o minúscula.
- b) Modifica el programa para que perrillos contenga los nombres de los perros y las ciudades (ambos).
- c) Comprueba si funciona lo siguiente:

```
perrillos.unshift('Estepona');
```

9. Teniendo un array muy grande con elementos de la siguiente estructura:

```
"28924;Estepona"
```

Crea un texto y un botón que ponga *buscar* que muestre el CP y el nombre de la ciudad que coincida o contenga el texto del campo creado. No importan las mayúsculas y minúsculas en la búsqueda.

- 10.** Elabora una página web que cree dos cookies con el nombre y el contenido que consideres oportuno.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué hace setTimeout?:

- a) Ejecuta una función transcurrido un tiempo determinado.
- b) Hace que se ejecute el script transcurrido un tiempo determinado.
- c) Ejecuta una función de forma repetida cada x milisegundos.

2. ¿Qué se usa para borrar una cookie?:

- a) Se utilizará la función removeCookie().
- b) Se utilizará la función deleteCookie().
- c) Se puede utilizar el parámetro max-age=0.

3. ¿Hasta qué capacidad puede almacenar información el navegador?:

- a) 50 MB.
- b) 5 MB.
- c) Ilimitada.

4. El objeto localStorage para almacenar información utiliza el método:

- a) setItem(clave,valor).
- b) storeData(clave,valor).
- c) saveData(clave,valor).

5. Una de las siguientes líneas de código no funciona, ¿cuál es?:

- a) var suma = new Function("a", "b", "return a + b");
- b) var suma = new Function("a", "b", "{return a + b}");
- c) var suma = new Function("a", "b", {"return a + b"});

6. ¿Qué función se utiliza para llenar un array con un valor determinado?:

- a) fill().
- b) push().
- c) slice().

7. ¿Qué función se utiliza para eliminar elementos concretos de un array?:

- a) deleteItem().
- b) splice().
- c) removeItem().

8. ¿Qué función se usa para crear un array de un string que tenga campos delimitados por algún carácter?:

- a) split().
- b) slice().
- c) splice().

9. De las siguientes expresiones, ¿cuál devolverá false?:

- a) isFinite(-3.14)
- b) isFinite(0)
- c) isFinite("5/9")

10. De las siguientes expresiones, ¿cuál devolverá true?:

- a) `isNaN(true)`
- b) `isNaN(8/0)`
- c) `isNaN(0/0)`

SOLUCIONES:

- | | | |
|----------|----------|-----------|
| 1. a b c | 5. a b c | 9. a b c |
| 2. a b c | 6. a b c | 10. a b c |
| 3. a b c | 7. a b c | |
| 4. a b c | 8. a b c | |