



J.G. GUTIERREZ

HTML5, CSS3 Y JAVASCRIPT

PRIMERA EDICIÓN

Contents

| | |
|---|---------------|
| Tutorial de introducción a HTML5, CSS3 y al pensamiento computacional con JavaScript | 3 |
| Introducción | 3 |
| Cómo usar el curso | 3 |
| Resumen del curso | 3 |
| Objetivo | 4 |
| Presentación del curso | 5 |
| Semana 1: HTML5 - Sintaxis básica y tablas | 6 |
| ¿Cómo funciona Internet? | 6 |
| Lenguajes de marcas | 7 |
| Primer ejemplo | 7 |
| Etiquetas más comunes de HTML | 7 |
| Las nuevas etiquetas de HTML5 | 7 |
| Estructura básica | 8 |
| Títulos y párrafos | 9 |
| Formato de palabras | 10 |
| Hiperenlaces o anclas | 11 |
| Imágenes | 11 |
| Uso de audio y video en HTML5 | 12 |
| Formularios en HTML5 | 12 |
| Listas | 14 |
| La etiqueta <div> | 15 |
| Plantilla de una página Web con hiperenlaces | 15 |
| Editores de texto | 15 |
| Semana 2: HTML5 - El modelo de cajas | 16 |
| Cómo añadir CSS a una Web | 17 |
| Añadiendo estilos en la misma etiqueta | 17 |
| Añadiendo estilos en la cabecera de la página | 18 |
| Añadiendo estilos desde un archivo externo | 19 |
| El modelo de cajas | 19 |
| Altura y anchura de una caja | 19 |
| Margen | 21 |
| Relleno | 21 |
| Borde | 21 |
| Laterales | 22 |
| Más opciones para los bordes | 22 |
| Esquinas redondeadas | 23 |
| Sombras | 23 |
| Elementos flotantes | 23 |
| Tablas con estilo | 24 |
| Tablas en HTML | 24 |
| Ejercicio semana 2 | 26 |

| | |
|--|-----------|
| Semana 3: CSS3 - Con un poco de estilo. Menús | 29 |
| UI vs UX | 29 |
| Accesibilidad | 29 |
| Menús de aplicaciones | 30 |
| Preparando el esqueleto de nuestra APP/Web | 30 |
| El menú de nuestra aplicación | 31 |
| Ejercicio horario de clase | 37 |
| Semana 4: JavaScript - ¡Hola Mundo! | 39 |
| El desarrollo de aplicaciones | 39 |
| Primeros pasos | 39 |
| Sintaxis básica | 41 |
| Identificadores | 41 |
| JS es sensible a mayúsculas/minúsculas | 42 |
| Comentarios | 42 |
| Instrucciones o sentencias y bloques | 42 |
| Tipos de datos | 43 |
| Estructuras de control | 48 |
| Condicionales if-else | 48 |
| Switch..case...break | 48 |
| Bucle for | 49 |
| Bucle while | 50 |
| Bucle do..while | 50 |
| Ejercicio tablas de multiplicar | 50 |
| Ejercicios resueltos y videotutorial de la semana | 51 |
| Semana 5: JavaScript - Variables, clases, métodos y funciones | 52 |
| Introducción | 52 |
| Ámbito de una variable | 52 |
| Listas | 53 |
| Diccionarios | 55 |
| Ejemplo de función: creando el tablero de juego | 55 |
| Fichero Matriz.js | 58 |
| Ejercicio: Adivina en qué número pienso | 64 |
| Semana 6: JavaScript - El árbol DOM | 65 |
| jQuery | 67 |
| Fichero juego.js | 69 |
| Semana 7: JavaScript - Pensamiento computacional. Algoritmos | 75 |
| Ejercicio: La letra del DNI | 75 |
| Semana 8: JavaScript - Almacenando datos | 76 |
| Fichero marcadores.js: | 76 |

Tutorial de introducción a HTML5, CSS3 y al pensamiento computacional con JavaScript

Introducción

Este curso ha sido realizado para ayudar a nuestros alumnos durante los meses de confinamiento por la pandemia del COVID-19. Primavera del 2020.



Este obra está bajo una licencia de Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional.

Para seguir el curso te animamos a visitar este canal de Youtube donde podrás de seguir estas explicaciones.

Si estás viendo una copia en formato PDF o ePub del libro, recuerda que siempre puedes acceder a la última versión en el repositorio de Github.

Suponemos que tienes ciertas nociones de programación (sea por haber participado en alguna iniciativa como Codeweeek o La hora del código o con alguna herramienta visual como Scratch).

Cómo usar el curso

Para cada entrega (semana) vamos creando una rama nueva, de tal manera que si quieres seguir el proyecto viendo los vídeos, sólo necesitarás cambiar a la rama de la semana del vídeo.

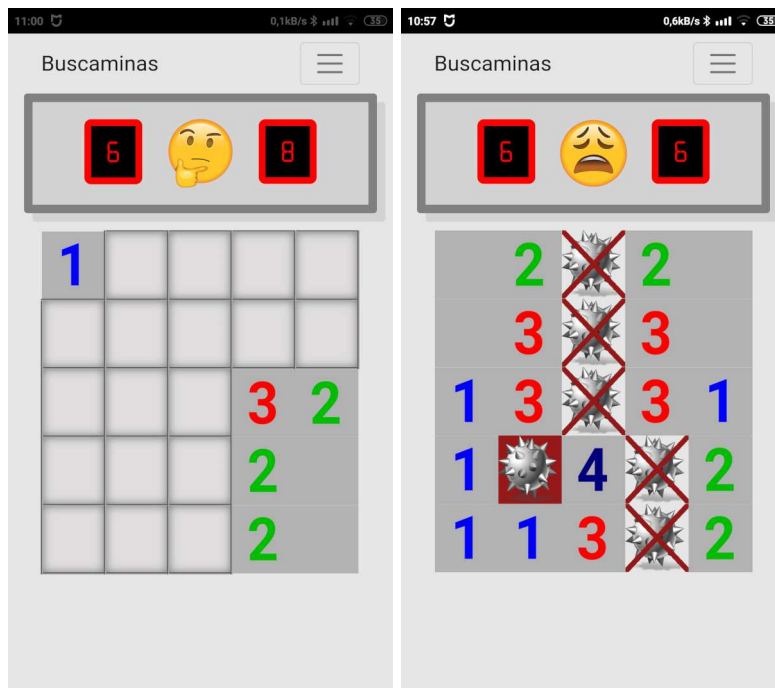
Resumen del curso

1. Semana 1: HTML5 - Sintaxis básica y tablas.
 - Ejercicio: Plantilla de una página Web con hiperenlaces.
2. Semana 2: HTML5 - El modelo de cajas.
 - Ejercicio: Esqueleto de una página Web.
3. Semana 3: CSS3 - Con un poco de estilo. Menús.
 - Ejercicio horario de clase.
4. Semana 4: JavaScript - ¡Hola Mundo!
 - Ejercicio: mostrar por consola las tablas de multiplicar.
5. Semana 5: JavaScript - Variables, clases, métodos y funciones.
 - Ejercicio: Adivina en qué número pienso.
6. Semana 6: JavaScript - El árbol DOM.
 - Ejercicio: Modificar el contenido de nuestra página al pulsar botón.
7. Semana 7: JavaScript - Pensamiento computacional. Algoritmos.

- Ejercicio: La letra del DNI.
8. Semana 8: JavaScript - Almacenando datos.
- Ejercicio: Guardando información en LocalStorage.

Objetivo

Al terminar el curso, habrás aprendido cómo funciona HTML5, CSS3 y algo de JavaScript y habrás generado un juego como éste:



Presentación del curso



Semana 1: HTML5 - Sintaxis básica y tablas

¿Cómo funciona Internet?

Cuando queremos visitar una página Web normalmente nos vamos a nuestro buscador favorito y hacemos la consulta. Pero si nos fijamos en la barra de direcciones, cuando estamos en esa Web, ahí hay escrito lo que llamamos un **identificador uniforme de recurso o URL** (Uniform Resource Locator), es decir una dirección que nos permite acceder de forma inequívoca a un recurso de un determinado servidor.

El formato general de un URL es:

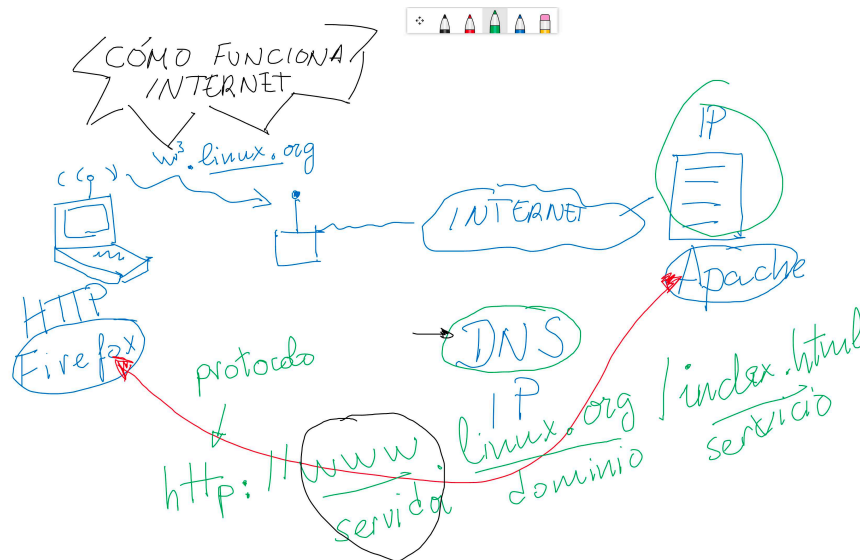
protocolo://máquina.directorio.archivo

aunque también pueden añadirse otros datos:

protocolo://usuario:contraseña@máquina:puerto.directorio.archivo.

Por ejemplo: https://es.wikipedia.org/wiki/Localizador_de_recursos_uniforme

- **https** es el protocolo
- **es** es el servidor para España de Wikipedia
- **wikipedia.org** es el dominio donde está la página
- **/wiki/Localizador_de_recursos_uniforme** es el archivo que estamos consultando



Normalmente confundimos Internet con la World Wide Web ó WWW. Como podemos leer en Wikipedia, el World Wide Web (WWW) o red informática

mundial es un sistema de distribución de documentos de hipertexto o hipermedia interconectados y accesibles a través de Internet. Con un navegador web, un usuario visualiza sitios web compuestos de páginas web que pueden contener textos, imágenes, vídeos u otros contenidos multimedia, y navega a través de esas páginas usando hiperenlaces.

Lenguajes de marcas

HTML es un lenguaje de marcas, es decir, que usamos unas etiquetas predefinidas para dar estructura o significado al texto de la página.

En un PC sabemos que un archivo es de este tipo porque **normalmente usa la extensión html o html**.

Ejemplos:

```
1 <h1> Esto es un título</h1>
2 <p> Esto es un párrafo </p>
```

HTML5 es la última versión de este lenguaje, que intenta diferenciarse de versiones anteriores porque sus etiquetas van tomando un valor más semántico (explican qué tipo de contenido contienen), dejando a las hojas de estilo (CSS3 o Cascade Style Sheets versión 3, ya las veremos más adelante).

Primer ejemplo

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="utf-8">
5   </head>
6   <body>
7     Aquí va el texto que vemos en el navegador.
8   </body>
9 </html>
```

Etiquetas más comunes de HTML

Las nuevas etiquetas de HTML5

Como decíamos con anterioridad, HTML5 plantea un nuevo esquema y secciones de un documento: <section>, <article>, <nav>, <header>, <footer>, <aside>.

Estructura básica

Sea el siguiente ejemplo adaptación de los tutoriales de Mozilla:

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="utf-8">
5   </head>
6   <body>
7     <nav>
8
9     </nav>
10    <section>
11      <h1>El fiero conejo</h1>
12    <section>
13      <h1>Introducción</h1>
14      <p>En esta sección presentamos al conocido mamífero.
15    </section>
16    <section>
17      <h1>Hábitat</h1>
18      <p>El conejo, como fiero depredador, necesita un entorno
19        con abundantes zorros que cazar.
20    </section>
21    <aside>
22      <p>otros estudiosos del conejo
23    </aside>
24  </section>
25  <footer>
26    <p>2010 The Example company
27  </p>
28  </footer>
29 </body>
30 </html>
```

En el anterior ejemplo tenemos que:

- La etiqueta `<!DOCTYPE html>` indica que se trata de un documento HTML.
- La etiqueta `<html lang="es">` y su pareja que cierra al final `</html>` sirven para indicar que dentro está el documento HTML. Fíjate en el atributo **lang="es"**, sirve para indicar que el documento está en español. Si lo cambiamos por **lang="en"**, ¿qué idioma crees que sería el del documento? Correcto, inglés (**E**nglish).
- `<header>` y `</header>`: sirven para definir un bloque de contenido que hará las veces de título de la página web.
- `<footer>` y `</footer>`: define el pie de página de nuestra web.

- `<nav>` y `</nav>`: donde incluiremos diferentes enlaces para que el usuario pueda desplazarse entre las partes de nuestro sitio web.
- `<section>` y `</section>`: para definir grandes secciones de nuestra página.
- `<article>` y `</article>`: marca los límites de un contenido específico, como una entrada de un blog o un artículo en general.
- `<aside>` y `</aside>`: se emplea para definir un contenido que está relacionado con la página, pero que se debe considerar como separado del contenido principal.

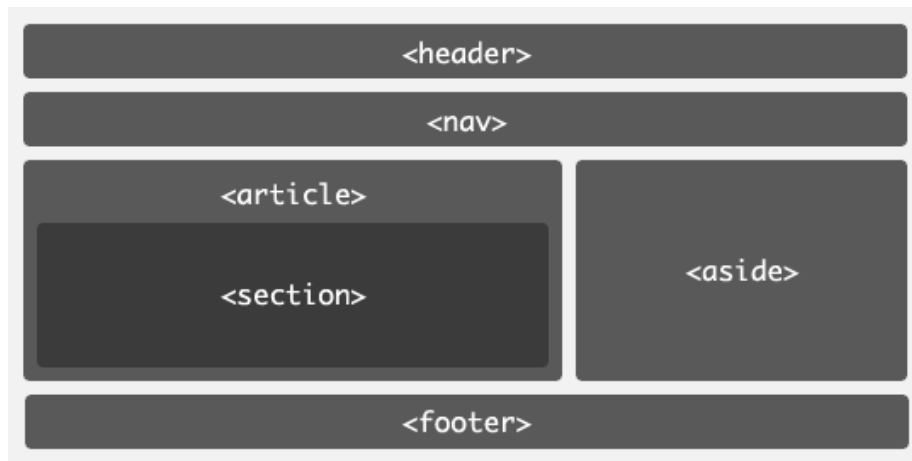


Figure 1: Esquema general de una página Web.

Los elementos en HTML usualmente son bien elementos “en línea” o bien elementos a “nivel de bloque”. Un elemento en línea ocupa sólo el espacio acotado por las etiquetas que lo definen. Un elemento en bloque pueden ser varias líneas.

Títulos y párrafos

El lenguaje HTML es muy cuidadoso con la organización de la información, por lo que lo primero que debemos conocer es cómo estructurar los títulos y cómo definir los párrafos de texto. Esto serían elementos de bloque.

Para dar formato a una palabra o conjunto de palabras (negrita, cursiva, subrayado, tachado...) usamos etiquetas que definen elementos “en línea”.

Párrafos Las etiquetas `<p>` y `</p>` se emplean para definir un bloque de texto que se comporta como un párrafo. Normalmente no dejaremos nunca una porción de texto suelta por la página web, sino que la rodearemos con esas etiquetas. El editor de texto se encargará de hacerlo por nosotros pero, si estamos usando otro tipo de editor, debemos asegurarnos de qué sucede.

Títulos Las etiquetas `<h1>` y `</h1>` se utilizan para definir un texto como título, indicando que es una cabecera (la h viene de header, cabecera en inglés) que queremos destacar sobre el resto del texto. Junto a `<h1>` contamos con `<h2>`, `<h3>` y así hasta `<h6>` para definir diferentes títulos, de mayor a menor importancia.

Una página web bien diseñada contará con estos encabezados para definir los distintos apartados del texto, con sus diferentes niveles. En la figura se puede observar cómo hemos incorporado algunos encabezados, en este caso h1 y h2, a nuestro texto. Se consigue añadiendo el texto y a continuación seleccionando el encabezado deseado en cuadro de la parte izquierda.

Cada uno de los niveles de encabezado tiene una apariencia diferente de tamaño y tipo de letra. Este aspecto se puede modificar como veremos un poco más tarde. Saltos de línea y líneas separadoras

Para complementar las opciones de separación del texto, contamos con dos etiquetas más:

- `
` inserta un salto de línea en el texto. No genera un nuevo párrafo, sino que parte la línea en dos. Es un elemento puntual, que no lleva etiqueta de cierre.
- `<hr>` inserta un salto de línea en el texto, pero mostrando una línea horizontal visible.

Formato de palabras

Aunque existen etiquetas para poner el texto en negrita: `` o ``, cursiva `<i>`, etc. hoy día ya están obsoletas en HTML5. Ahora deberíamos cercar con `` (de *abarcar* en inglés) el texto que queremos decorar y darle formato con CSS.

Ejemplos:

```
1 <p>
2   Esto es un párrafo con el siguiente texto en rojo
3   <span style="color: red;"> y esto un span dentro de un párrafo.
4   </span>
5 </p>
6 <p>
7   Y esto otro párrafo con el siguiente texto en negrita
8   <span style="font-weight: bold;"> y esto un span dentro de un
9   párrafo. </span>
10 </p>
11 <p>
12   Y esto otro párrafo con el siguiente texto en cursiva
13   <span style="font-style: italic;"> y esto un span dentro de un
14   párrafo. </span>
```

```

12 </p>
13 <p>
14   Y esto otro párrafo con el siguiente texto en negrita
15   <span style="text-decoration:underline;"> y esto un span dentro
      de un párrafo. </span>
16 </p>

```

Esto da lugar en un navegador a la siguiente salida:

Esto es un párrafo con el siguiente texto en rojo **y esto un span dentro de un párrafo.**

Y esto otro párrafo con el siguiente texto en negrita **y esto un span dentro de un párrafo.**

Y esto otro párrafo con el siguiente texto en cursiva y *esto un span dentro de un párrafo.*

Y esto otro párrafo con el siguiente texto en negrita y esto un span dentro de un párrafo.

Figure 2: uso de span

Hiperenlaces o anclas

El elemento ancla o hiperenlace <a> crea un enlace a otras páginas de internet, archivos o incluso partes dentro de la misma página, direcciones de correo, o cualquier otra URL:

```

1 <a href="https://www.youtube.com/juanguedu">Mi canal de Youtube</a>

```

Imágenes

la etiqueta se emplea para insertar una imagen en la página web, pero por si sola no funciona correctamente. Necesita que le incorporemos un parámetro en el que indiquemos qué imagen será la que se muestre. Quedaría así:

```

1 

```

En el ejemplo siguiente, además de indicar qué imagen se mostrará, establecemos el tamaño que ocupará en la pantalla:

```

1 

```

Uso de audio y video en HTML5

Los elementos `<audio>` y `<video>` permiten la manipulación de nuevo contenido multimedia.

Imagina que tienes un vídeo que has grabado con el móvil en formato MP4 (Android) o MOV (iPhone), ¿cómo insertarlo en una página Web? Sencillo, sería algo así:

```
1 <video src="mi_pelicula.mp4" autoplay
  poster="imagen_inicial.jpg">
2     Su navegador no soporta la etiqueta video.
3 </video>
```

Si nuestro navegador es moderno, veremos el vídeo, en caso contrario veríamos el mensaje *“Su navegador no soporta la etiqueta video”*.

Formularios en HTML5

Cuando abres un navegador y vas a la página de tu buscador favorito, **eso es un formulario**, fíjate cómo hay una caja donde escribes el texto a buscar y luego un botón para enviar que pulsaremos para ir a la siguiente página con los resultados de la búsqueda.

Aquí tienes un ejemplo de formulario. Fíjate en la etiqueta `<input>`. Verás que según el tipo de información que quieres consultar, es el atributo **type** de cada entrada de texto:

```
1 <p>Esto es un típico formulario Web</p>
2 <form action="getform.php" method="get">
3     <label>Nombre: <input type="text"></label><br>
4     <label>Apellido: <input type="text"></label><br>
5     <label>E-mail: <input type="email"></label><br>
6     <label>Edad: <input type="number" min="18" max="120"><br>
7     <input type="submit" value="Submit">
8 </form>
```

Esto dará lugar a un formulario como éste:

Además el trabajar con HTML5 nos ofrece otra ventaja como es aprovechar su mejora de los formularios web: Validación de restricción (p.ej. si un campo es un número, podemos indicar entre qué números deberá estar comprendido), varios atributos nuevos, nuevos valores para `<input>` como el atributo `type` y el nuevo elemento `<output>`.

Observa cómo funciona esto de las restricciones en el ejemplo anterior cuando intentamos introducir una letra en la edad:

Esto es un típico formulario Web

Nombre:

Apellido:

E-mail:

Edad:

Figure 3: Formulario información personal

Esto es un típico formulario Web

Nombre:

Apellido:

E-mail:

Edad:

Introduce un número

Figure 4: Formulario Restricción Edad

Listas

Para declarar listas usamos o bien la etiqueta `` (*del inglés `ordered list`*) si queremos una lista numerada u **ordenada** o bien la etiqueta ``, si queremos una lista desordenada (*del inglés `unordered list`*).

El elemento `` del inglés **list item** o elemento de lista, declara cada uno de los elementos de una lista.

Ejemplo:

```
1 <ul>
2 <li>primer elemento</li>
3 <li>segundo elemento</li>
4 <li>tercer elemento</li>
5 </ul>
```

dará como salida:

- primer elemento
- segundo elemento
- tercer elemento

Figure 5: Lista ordenada

Sin embargo, el siguiente ejemplo:

```
1 <ol>
2 <li>primer elemento</li>
3 <li>segundo elemento</li>
4 <li>tercer elemento</li>
5 </ol>
```

dará como salida:

1. primer elemento
2. segundo elemento
3. tercer elemento

Figure 6: Lista ordenada

¿Has visto la diferencia entre usar `` y ``?

La etiqueta <div>

En un documento HTML el elemento <div> (del inglés division) permite crear divisiones, también llamadas secciones o zonas. Las divisiones se utilizan para agrupar elementos y aplicarles estilos. Es el contenedor genérico para dar diseño al contenido. Ejemplo:

```
1 <div class="container">
2   <div class="well">One</div>
3   <div class="well">Two</div>
4   <div class="well">Three</div>
5   <div class="well">Four</div>
6   <div class="well">Five</div>
7   <div class="well">Six</div>
8 </div>
```

Recuerda este elemento porque vamos a usarlo muchas veces.

Plantilla de una página Web con hiperenlaces

Ahora que ya tenemos una ligera noción de qué es eso del HTML te proponemos el siguiente ejercicio: Vamos a crear una Web a partir del ejemplo de estructura básica. La página debe tener un título, un autor y al menos un hiperenlace. Recuerda guardar el archivo con la extensión html (“ejemplo.html”).

Editores de texto

Si no puedes localizar en el equipo un editor de texto plano con el que poder hacer las prácticas y tareas del proyecto, puedes instalar uno de los siguientes:

- Microsoft Visual Studio Code (si no lo quieres instalar también hay opción de usarlo en modo “portable”)
- Geany
- Notepad++

En los vídeos y en clase tenemos Visual Studio Code, luego ante duda te recomendamos éste.

En Windows, el más ligero y fácil de usar es Notepad++.

En Linux, el más sencillo es Geany.

Si no quieres instalar nada, puedes trabajar on-line con alguna Web como Code-Pen.

¿Qué tal, has podido crear tu primera página Web, ha sido fácil?

Semana 2: HTML5 - El modelo de cajas

Antes de comenzar con el modelo de cajas vamos a ver cómo se organizan los archivos de una página Web.

Una página Web se organiza en archivos (realmente HTTP lo llama objetos). Cuando hacemos una petición HTTP (nuestro navegador en nuestro PC llama al servidor Web en el equipo remoto donde está hospedada la página) no sólo nos descargamos texto, también puede haber imágenes, sonidos, vídeo...

Fíjate en esta petición (hemos activado el modo “Desarrollador Web” del navegador para que se vea la consola que tienes debajo):

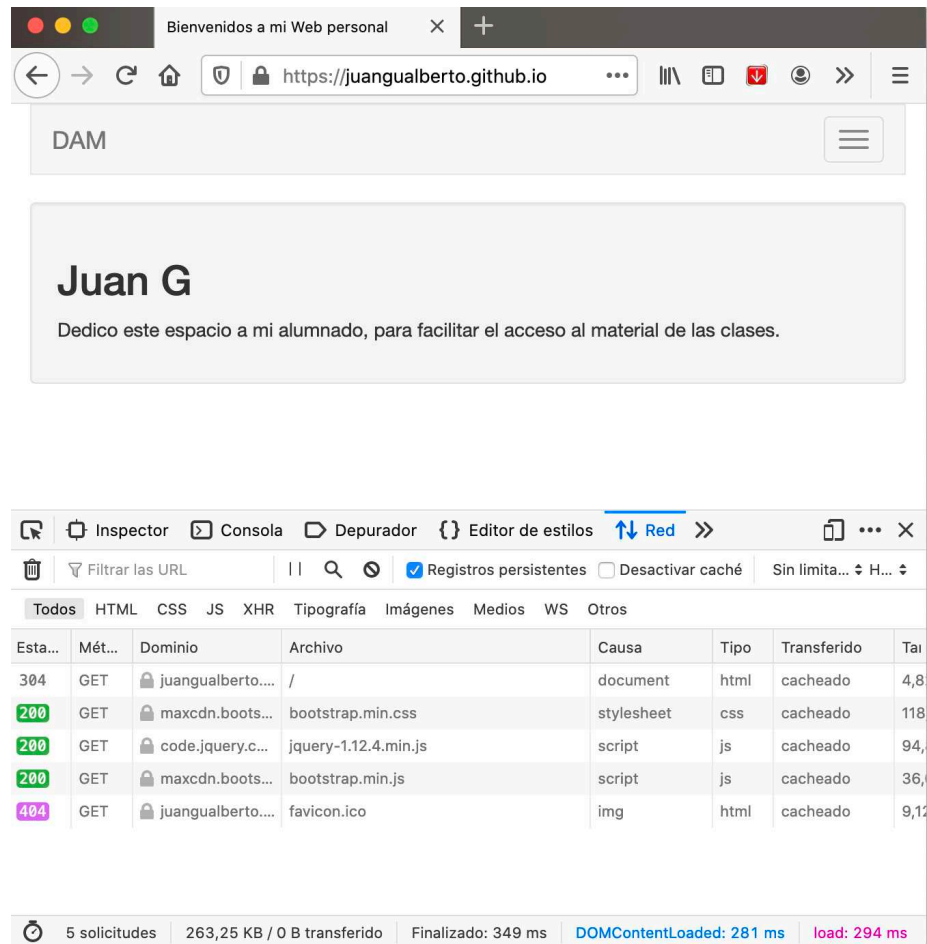


Figure 7: Firefox en modo Desarrollador Web

Yo he visitado una página, pero sin embargo se han completado 5 solicitudes

(mira en la esquina inferior izquierda). Para ver esa página hemos descargado 5 objetos: el fichero HTML, la hoja de estilo BootStrap, el JavaScript jQuery, el JavaScript BootStrap y el icono de la página Web o “favicon” (éste ha dado error 404, no lo ha encontrado).

Como puedes observar, por sencilla que sea una Web, siempre tendremos varios archivos, luego **es importante organizar el contenido en carpetas** para que quede todo bien **ordenado**.

Un ejemplo de organización podría ser éste (muy importante, intenta **siempre** que estén todos los nombres de archivos y carpetas **en minúsculas**):

- el fichero index.html en la carpeta raíz de la página
- todas las fotos e imágenes en la carpeta **img**
- las hojas de estilo (para dar formato, colores, espaciado) al texto en la carpeta **css**
- el código JavaScript para dinamizar en la carpeta **js**



Figure 8: Estructura de archivos

Cómo añadir CSS a una Web

Aún no hemos explicado qué son ni para qué sirven las hojas de estilo (lo veremos en el siguiente apartado) pero antes sí queremos mostrarte cómo tenemos que añadir esos “estilos” a la página Web.

Añadiendo estilos en la misma etiqueta

Podemos aplicar estilos dentro de un elemento concreto de la página web, mediante el atributo **style** que se puede establecer para cualquier etiqueta. Veamos

un par de ejemplos:

```
1 <h1>Qué entendemos por un <span style="color: rgb(0, 153, 0);">párrafo</span></h1>
2 <p style="font-style: italic; color: rgb(20, 20, 200);">Un párrafo de texto se compone de un bloque de texto independiente con una apariencia concreta, delimitado por un espacio superior y otro inferior.</p>
```

Esto se vería parecido a la siguiente imagen en un navegador (Estilos en elementos).

Qué entendemos por un párrafo

Un párrafo de texto se compone de un bloque de texto independiente con una apariencia concreta, delimitado por un espacio superior y otro inferior.

Figure 9: Estilos en elementos

Añadiendo estilos en la cabecera de la página

En la cabecera (dentro del elemento `<head></head>`) podemos indicar las reglas CSS que necesitamos, afectando de este modo sólo a la página web en cuestión.

Imagina que, queremos que todos los párrafos (etiqueta `<p></p>`) de mi página Web y el h1 (*importante*: sólo debe haber un `<h1></h1>` en cada página)

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <style>
5       h1 {
6         color: rgb(0, 153, 0);}
7       p  {
8         font-style: italic;
9         color: rgb(20, 20, 200);
10      }
11     </style>
12  </head>
13  <body>
14    <h1>¿Qué entendemos por un párrafo?</h1>
15    <p>Un párrafo de texto se compone de un bloque de texto independiente con una apariencia concreta, delimitado por un espacio superior y otro inferior.</p>
```

```
16 </body>
17 </html>
```

Esto se vería un poco diferente al ejemplo anterior en un navegador (intenta pensar porqué no es exactamente igual):

¿Qué entendemos por un párrafo?

Un párrafo de texto se compone de un bloque de texto independiente con una apariencia concreta, delimitado por un espacio superior y otro inferior.

Figure 10: Estilos en elementos

Añadiendo estilos desde un archivo externo

El modelo de cajas

Como ya te explicamos el contenido de una página Web lo vamos guardando entre etiquetas. Estas etiquetas contienen el texto, es decir con contenedores o cajas de texto. Para dar el aspecto que tienen a las páginas Web (y que no sean solamente texto), les damos estilos, es lo que se llama CSS (Cascade Style Sheets - hojas de estilo en cascada).

En general, hay dos tipos de cajas (o contenedores): cajas en bloque y cajas en línea. Estas características se refieren al modo como se comporta la caja en términos de flujo de página y en relación con otras cajas de la página:

Si una caja se define como un bloque, se comportará de las maneras siguientes:

- La caja se extenderá en la dirección de la línea para llenar todo el espacio disponible que haya en su contenedor. En la mayoría de los casos, esto significa que la caja será tan ancha como su contenedor, y llenará todo el ancho espacio disponible
- La caja provoca un salto de línea al llegar al final de la línea
- Tiene un ancho y alto (propiedades width y height)
- El área de relleno, el margen y el borde mantienen a los otros elementos alejados de la caja

Estas opciones nos proporcionan un gran control sobre cómo debe situarse cada elemento.

Altura y anchura de una caja

Cada elemento HTML de una página web cuenta con una anchura y una altura específica. En muchos casos esas dimensiones se las proporciona el propio con-

Caja de un elemento HTML

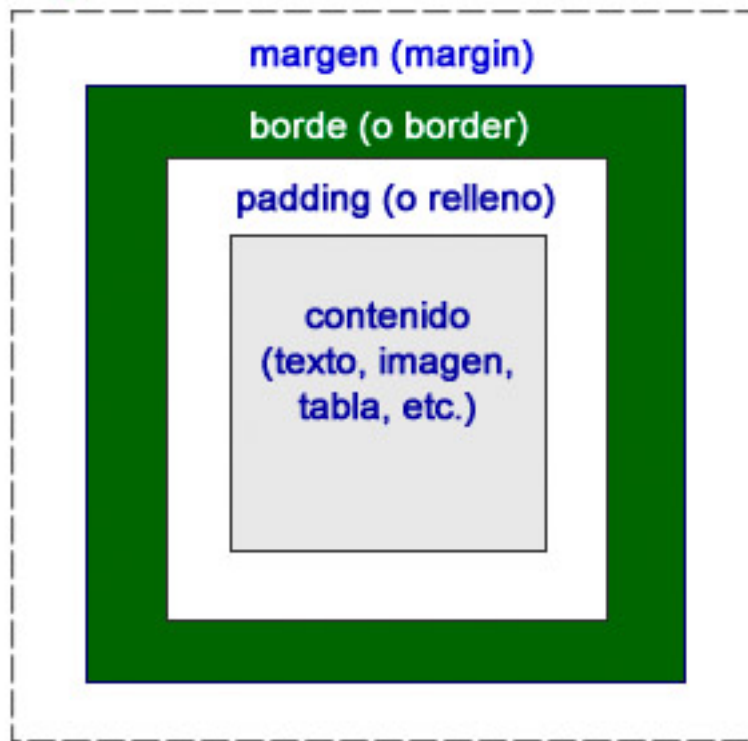


Figure 11: modelo de cajas

tenido, como en un párrafo o una imagen, por ejemplo. Esos valores de anchura (width) y de altura (height) pueden ser modificados mediante las hojas de estilo, gracias a las propiedades del mismo nombre.

Así podemos hacer párrafos más estrechos, imágenes que se sobredimensionen o simplemente ajustar diferentes bloques, para que se acomoden correctamente en la pantalla.

Los valores width y height se acompañan de un valor numérico exacto o de un porcentaje, como en otras muchas propiedades.

```
1 body {  
2     principal { width: 400px;  
3     background-color: rgb(0, 126, 0);  
4 }
```

Margen

Comenzaremos con la imagen. Con tan sólo modificar su margen, observaremos cómo se distancia del resto de los elementos. Usaremos la propiedad margin seguida de un valor numérico o de un porcentaje:

```
1 img { margin: 20px; }
```

Relleno

Probaremos ahora a modificar su relleno, es decir, la distancia imaginaria entre un hipotético borde y la imagen propiamente dicha. Para ello emplearemos la propiedad padding, exactamente igual que hicimos con la anterior. Probemos con un valor menos exagerado:

```
1 img { padding: 5px; }
```

Borde

Si recargamos la página con esta incorporación, observaremos que, en efecto, la imagen se separa un poco más, esos 5 píxeles por cada lado, pero no es posible distinguir dónde acaba el efecto del margen y comienza el del relleno. Para poder diferenciar los valores, deberíamos tener un borde en la imagen.

```
1 img {  
2     border-width: 2px;  
3     border-style: solid;  
4     border-color: #007000;  
5 }
```

Con los conocimientos que tenemos ya de CSS podemos intuir con facilidad qué es lo que hace cada una de esas tres propiedades: en una definimos el grosor del borde, en otra el tipo de línea y en la última su color.

Laterales

Tanto margin, como padding y border se pueden emplear para modificar laterales de una caja, con independencia de los demás. Añadiendo a cada uno de ellos la variación -left (izquierda), -right (derecha), -top (arriba) o -bottom (abajo) conseguimos que sólo afecte al valor o valores indicados.

En la figura hemos aplicado estas propiedades para el título de la página:

```
1 h1 {  
2     margin-top:40px;  
3     padding-left: 5px;  
4     padding-right:5px;  
5     border-top-width: 2px;  
6     border-top-style: dotted;  
7     border-top-color: #007000;  
8     border-bottom-width: 2px;  
9     border-bottom-style: double;  
10    border-bottom-color: #007000;  
11 }
```

Más opciones para los bordes

Para los bordes podemos definir tres propiedades: su anchura, su estilo y su color. La anchura y el color se definen con las medidas habituales y los sistemas que ya hemos analizado. El estilo, por su parte, se basa en una serie de valores concretos:

```
1 dotted: punteado.  
2 dashed: línea discontinua.  
3 solid: línea continua.  
4 double: línea doble.  
5 groove: tipo de relieve.  
6 ridge: tipo de relieve.  
7 inset: tipo de relieve.  
8 outset: tipo de relieve.  
9 none: empleado para indicar que no habrá borde.
```

El valor solid es la línea sencilla y la más empleada.

Como ya sucedía con otras propiedades, podemos reagrupar los valores referidos a los bordes en una sola propiedad genérica denominada border. Para ello es-

tableceremos los valores separados por espacios y en el orden de tamaño, estilo y color, como en este ejemplo que haría la misma función que el recuadro anterior:

```
1  img { border: 2px solid #007000;}
```

Esquinas redondeadas

Con los estilos actuales podemos trazar un borde alrededor de una figura y que tenga sus esquinas redondeadas.

La propiedad que lo permite es `border-radius`, acompañada de un valor numérico. El ejemplo anterior, con la incorporación de esta propiedad, daría como resultado el rectángulo de la figura:

```
1  img {  
2    border: 2px solid #007000;  
3    border-radius: 25px;  
4  }
```

Sombras

Las modernas hojas de estilo proporcionan a cualquier elemento la capacidad de proyectar una sombra. Ya vimos que esto funcionaba con el texto, pero además contamos con la propiedad `box-shadow` para crear sombras en cualquier caja de nuestra página web, lo que hace que sea posible aplicárselo a cualquier elemento.

```
1  table {  
2    box-shadow: 8px 8px 6px #aaaaaa;  
3  }
```

Los valores que conforman la sombra son similares a los que vimos para las sombras de texto, es decir, desplazamiento horizontal, vertical, difuminado y color de sombra.

Elementos flotantes

Los elementos de una página web pueden reubicarse a la izquierda o a la derecha con tan sólo emplear la propiedad `float`, haciendo que el resto del contenido se sitúe alrededor de ese elemento.

En el siguiente ejemplo la regla:

```
1  img {  
2    float: left;  
3  }
```

provoca que el texto se sitúe alrededor de la imagen.

Tablas con estilo

Tablas en HTML

Cuando manejamos información, es interesante representarla en diferentes formatos. Seguro que recuerdas lo que es una hoja de cálculo. Si tienes instalado un programa de hoja de cálculo, ábrelo y recuerda cómo teníamos filas y columnas.

Ahora compara el contenido de la siguiente tabla HTML (busca lo que hay dentro de `<table>` y `</table>` en este ejemplo) con las siguientes imágenes (fíjate como le digo que haga una fila con las etiqueta `<tr>` y una celda con la etiqueta `<td>`):

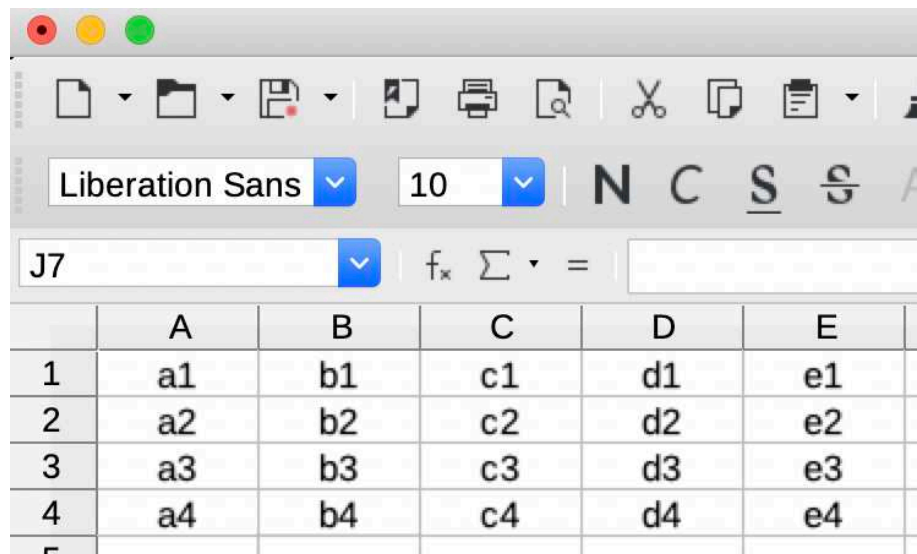
```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width,
6     initial-scale=1.0">
7   <title>Ejemplo de tabla</title>
8   <style>
9     table, th, td {
10       border: 1px solid black;
11     }
12     th {
13       color:white;
14       background-color:black;
15     }
16   </style>
17 </head>
18 <body>
19   <table>
20     <thead>
21       <tr>
22         <th>A</th><th>B</th><th>C</th><th>D</th><th>E</th>
23       </tr>
24     </thead>
25     <tbody>
26       <tr>
27         <td>a1</td><td>b1</td><td>c1</td><td>d1</td><td>e1</td>
28       </tr>
29       <tr>
30         <td>a2</td><td>b2</td><td>c2</td><td>d2</td><td>e2</td>
31       </tr>
32       <tr>
33         <td>a3</td><td>b3</td><td>c3</td><td>d3</td><td>e3</td>
34     </tbody>
35   </table>
```

```

34     <tr>
35         <td>a4</td><td>b4</td><td>c4</td><td>d4</td><td>e4</td>
36     </tr>
37 </tbody>
38 </table>
39 </body>
40 </html>

```

En una hoja de cálculo la tabla sería:



| | A | B | C | D | E |
|---|----|----|----|----|----|
| 1 | a1 | b1 | c1 | d1 | e1 |
| 2 | a2 | b2 | c2 | d2 | e2 |
| 3 | a3 | b3 | c3 | d3 | e3 |
| 4 | a4 | b4 | c4 | d4 | e4 |

Figure 12: Una tabla en una hoja de cálculo

En el navegador la tabla se vería como:

| A | B | C | D | E |
|----|----|----|----|----|
| a1 | b1 | c1 | d1 | e1 |
| a2 | b2 | c2 | d2 | e2 |
| a3 | b3 | c3 | d3 | e3 |
| a4 | b4 | c4 | d4 | e4 |

Figure 13: Ejemplo de tabla

Fíjate también cómo en la cabecera de la página (etiqueta `<head>`) hemos añadido un elemento `<style>` con este contenido:

```
1 table, th, td {
2     border: 1px solid black;
3 }
4 th {
5     color:white;
6     background-color:black;
7 }
```

En estas líneas, mediante CSS le indicamos al navegador que queremos bordes alrededor de la tabla, de los `<th>` y `<td>` y que los `<th>` además tendrán la letra blanca con el fondo negro.

Ejercicio semana 2

Deberás crear todas las carpetas necesarias para ir haciendo la Web (css, js, img, snd...).

Dentro de la carpeta CSS crea un archivo estilos.css con el mismo contenido del ejemplo de las tablas.

Dentro de la carpeta JS crea un archivo index.js con el siguiente contenido:

```
1 console.log("Hola Mundo");
```

Hay que crear el fichero index.html con el siguiente contenido:

```
1 <!DOCTYPE html>
2 <html lang="es">
3     <head>
4         <meta charset="utf-8">
5         <meta name="viewport" content="width=device-width,
6             initial-scale=1.0">
7         <meta name="author" content="Juan Gualberto">
8         <meta name="copyright" content="GNU GPLv3">
9         <title>Buscaminas</title>
10        <link rel="stylesheet" href="css/estilos.css">
11    </head>
12    <body>
13        <nav>
14            <!-- Aquí irá el menú superior -->
15        </nav>
16        <section>
17            <!-- Aquí irán las vias, las minas y el smiley -->
18        </section>
```

```

18     <section>
19         <!-- Aquí estará el tablero de juego -->
20     </section>
21     <aside>
22         <!-- Esto no será necesario y lo borraremos -->
23     </aside>
24 </section>
25 <footer>
26     <!-- Aquí irá información adicional en el pié -->
27 </footer>
28 <script src="js/index.js"></script>
29 </body>
30 </html>

```

Para cargar las hojas de estilos, usamos el elemento `<link>` dentro la cabecera (elemento `<head>`) :

```

1 <link rel="stylesheet" href="css/estilos.css">

```

Para cargar el código JavaScript, fíjate cómo al final del código, justo antes del cierre de la etiqueta `</body>`, como hemos añadido un elemento `<script>`. Esto se hace así porque acelera la carga de la página en el navegador:

```

1 <script src="js/index.js"></script>

```

Recuerda cómo hemos hecho esto porque vamos a utilizarlo de ahora en adelante cada vez que hagamos una página Web.

PROGRAMACIÓN
HTML5 CSS3 Y JS

SEGUNDA SEMANA

Semana 3: CSS3 - Con un poco de estilo. Menús

Antes de avanzar en el tema de los estilos, queremos destacar que no se trata simplemente de “hacer bonitas” las páginas Webs o aplicaciones Web que diseñamos, también hay que trabajar la usabilidad y accesibilidad de las mismas.

UI vs UX

Cuando usamos una aplicación o página Web, queremos que sea intuitiva, fácil de usar, rápida, sin comportamientos extraños o errores. Esto tiene que ver con la interfaz de usuario y la experiencia de usuario, dos conceptos muy importantes que debes conocer.

En el área del *márketing digital*, la experiencia de usuario o **UX** hace referencia a las interacciones que hace un usuario con una marca, sitio Web o aplicación. Su objetivo es conseguir que la interacción entre el cliente y nuestro producto o servicio sea agradable.

En el área de la *ingeniería de la usabilidad*, la interfaz de usuario o **UI** cobra especial importancia porque el diseño que hagamos de nuestro producto o Web, debe estar centrado y atractivo para el usuario. Debe ser fácil de aprender, usar y robusto (sin fallos ni comportamientos extraños, no previstos).

Accesibilidad

Aunque se sale un poco del tema del curso, has de saber que existen una serie de normas tanto internacionales como locales sobre cómo hacer un sitio Web o una aplicación móvil accesibles a todo el mundo.

La **accesibilidad web** tiene como objetivo lograr que las páginas web sean utilizables por el máximo número de personas, independientemente de sus conocimientos o capacidades personales e independientemente de las características técnicas del equipo utilizado para acceder a la Web (Fuente: Universidad de Alicante). Esto enlaza con uno de los objetivos esenciales de la Web que planteó su creador:

“The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect” Tim Berners-Lee.

Una página Web accesible debería, al menos, de cumplir las siguientes pautas (fuente: Universidad de Alicante):

1. Imágenes y animaciones: Use el atributo alt para describir la función de cada elemento visual.
2. Mapas de imagen: Use el elemento map y texto para las zonas activas.
3. Multimedia: Proporcione subtítulos y transcripción del sonido, y descripción del vídeo.

4. Enlaces de hipertexto: Use texto que tenga sentido leído fuera de contexto. Por ejemplo, evite “pincha aquí”.
5. Organización de las páginas: Use encabezados, listas y estructura consistente. Use CSS para la maquetación donde sea posible.
6. Figuras y diagramas: Descríbalos brevemente en la página o use el atributo longdesc.
7. Scripts, applets y plug-ins: Ofrezca contenido alternativo si las funciones nuevas no son accesibles.
8. Marcos: Use el elemento noframes y títulos con sentido.
9. Tablas: Facilite la lectura línea a línea. Resuma.
10. Revise su trabajo: Verifique. Use las herramientas, puntos de comprobación y pautas de Guía de accesibilidad del consorcio w3.

El consorcio W3C es el encargado de redactar y revisar las **Pautas de Accesibilidad al Contenido en la Web (WCAG)**. Están dirigidas a los webmasters e indican cómo hacer que los contenidos del sitio web sean accesibles.

Las pautas WCAG 2.1 surgen con el objetivo de mejorar la accesibilidad principalmente de tres grupos de usuarios:

- Personas con discapacidad cognitiva o del aprendizaje
- Personas con baja visión
- Personas con discapacidad que acceden desde dispositivos móviles

Para más información:

- Introducción a las Pautas de Accesibilidad para el Contenido Web (WCAG)
- Guía de adaptación a WCAG 2.1 desde WCAG 2.0

Menús de aplicaciones

Preparando el esqueleto de nuestra APP/Web

Diseñar páginas Web atractivas es un proceso altamente creativo, lo mismo que ocurre con la programación. Es un proceso, en la mayoría de los casos, que nos llevará días, semanas, o incluso meses, por lo que es muy fácil perder en el horizonte el objetivo final.

Cuando diseñamos o bien pagamos para que nos hagan una aplicación o página Web, un recurso interesante para hacerse la idea de cómo queremos el producto final.

Diseño de prototipos Para decidir qué tipo de menú vamos a hacer, preparamos dos prototipos (pulsas en cada uno para ver el código fuente de cada uno). Para que te hagas una idea hemos preparado dos prototipos que puedes ver en vivo haciendo *clíc* en el enlace correspondiente.

Primero hicimos el prototipo 1, basado en BootStrap. Observa en las imágenes el menú y la pantalla de juego en modo fácil.



Figure 14: Imagen del menú con BootStrap

A continuación te mostramos el prototipo 2, basado en MaterializeCSS. De nuevo fíjate en el tipo de menú y el modo juego fácil.

En nuestro caso nos vamos a decantar por BootStrap cuyo uso está más extendido que Materialize.

El menú de nuestra aplicación

Ten a mano el esqueleto que hicimos la semana pasada donde teníamos la siguiente estructura de carpetas:

Vamos a cargar de internet, de un CDN, BootStrap (Bootstrap es una biblioteca multiplataforma para diseño de sitios y aplicaciones web). Abre el archivo **index.html** y vamos a modificar la cabecera para que quede así (cambia en el meta *author* tu nombre):

```
1 <head>
2   <meta charset="utf-8">
```

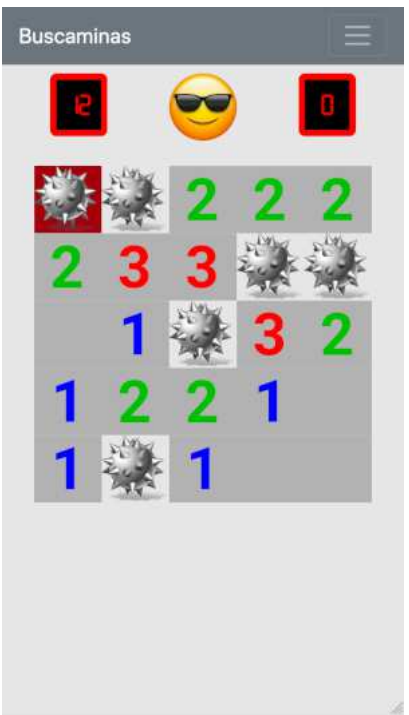



Figure 15: Imagen del juego con BootStrap



Figure 16: Imagen del menú con Materialize CSS

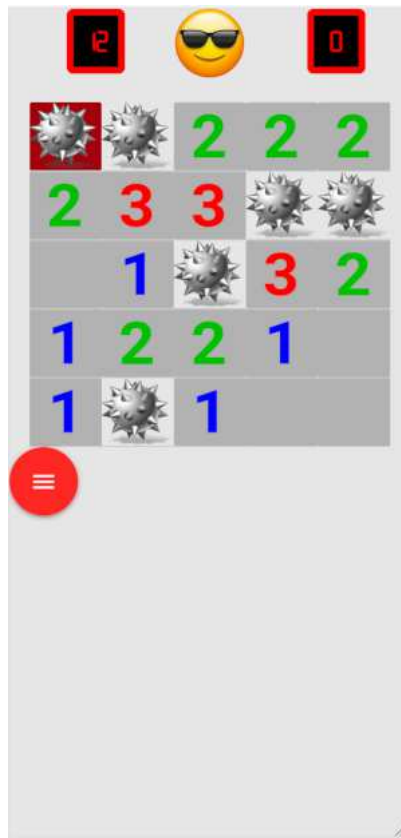


Figure 17: Imagen del juego con Materialize CSS

```
→ estructura git:(master) x tree
.
├── css
│   └── estilos.css
├── img
│   ├── 1.png
│   └── 2.png
├── index.html
├── js
│   └── index.js
└── snd
    └── aviso.wav
```

Figure 18: Estructura de carpetas de una página Web

```

3   <meta name="viewport" content="width=device-width,
      initial-scale=1">
4   <meta name="author" content="Juan Gualberto">
5   <meta name="copyright" content="GNU GPLv3">
6   <title>Buscaminas</title>
7   <link rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
8     integrity="sha384-Vkoo8x4CGs03+Hhvxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
      crossorigin="anonymous">
9   <link rel="stylesheet" href="css/estilos.css">
10 </head>

```

Fíjate bien en las dos etiquetas `<link>`, la primera es Bootstrap, la necesitamos para no tener que dar el aspecto que van a tener a los menús desde cero (colores, botones...). Simplemente incluyendo las clases de Bootstrap en nuestros programas, se van a ver mucho mejor. El segundo `<link>` es el archivo `css/estilos.css`, donde vamos a añadir nuestros estilos personales.

Ahora, justo antes de cerrar la etiqueta `<body>` (fíjate, al final del archivo `index.html`), vamos a añadir nuestro código JavaScript:

```

1   <footer>
2   </footer>
3   <!-- Aquí cargamos los SCRIPTS para acelerar la carga de la
      página -->
4   <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
      integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
5     crossorigin="anonymous"></script>
6   <script
      src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
7     integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
8     crossorigin="anonymous"></script>
9   <script
      src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
10    integrity="sha384-wfSDF2E50Y2D1uUdj003uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
      crossorigin="anonymous"></script>
11   <script src="js/controller.js"></script>
12   <script src="js/index.js"></script>
13 </body>
14 </html>

```

Cargamos tres archivos de internet (jQuery, Popper y Bootstrap) que son necesarios para las animaciones que trae de serie Bootstrap. Los dos últimos archivos vamos a hacerlos nosotros: el archivo `js/index.js` (arranca o inicializa los componentes adicionales) y `js/controller.js` (gestiona que los menús funcionen). Estos últimos además contendrán el código JavaScript que vamos a programar nosotros.

Ahora añadimos los menús, recuerda que son elementos de navegación (tu etiqueta etiqueta <nav> debe quedar como la que ves a continuación) y que los hacemos con listas:

```

1  <nav class="navbar navbar-expand-lg navbar-dark bg-secondary">
2    <a class="navbar-brand" href="#">Buscaminas</a>
3    <button class="navbar-toggler" type="button"
4      data-toggle="collapse" data-target="#navbarNav"
5      aria-controls="navbarNav" aria-expanded="false"
6      aria-label="Toggle navigation">
7      <span class="navbar-toggler-icon"></span>
8    </button>
9    <div class="collapse navbar-collapse" id="navbarNav">
10     <ul class="navbar-nav">
11       <li class="nav-item">
12         <a class="nav-link" href="#">Inicio </a>
13       </li>
14       <li class="nav-item">
15         <a class="nav-link dropdown-toggle" href="#"
16           id="navbarDropdownPartida" role="button"
17           data-toggle="dropdown" aria-haspopup="true"
18           aria-expanded="false">
19           Partida
20         </a>
21         <div class="dropdown-menu"
22           aria-labelledby="navbarDropdownPartida">
23           <a id="menu_partida_facil"
24             class="dropdown-item" href="#">Fácil</a>
25           <a id="menu_partida_medio"
26             class="dropdown-item" href="#">Normal</a>
27           <a id="menu_partida_dificil"
28             class="dropdown-item"
29             href="#">Pesadilla</a>
30         </div>
31       </li>
32       <li class="nav-item">
33         <a id="menu_puntuaciones" class="nav-link"
34           href="#">Puntuaciones</a>
35       </li>
36       <li class="nav-item">
37         <a class="nav-link dropdown-toggle" href="#"
38           id="navbarDropdownAyuda" role="button"
39           data-toggle="dropdown" aria-haspopup="true"
40           aria-expanded="false">
41           Ayuda
42         </a>

```

```

31         <div class="dropdown-menu"
32             aria-labelledby="navbarDropdownAyuda">
33             <a id="menu_ayuda" class="dropdown-item"
34                 href="#">Cómo jugar</a>
35             <a id="menu_licencia" class="dropdown-item"
36                 href="#">Licencia</a>
37             <a class="dropdown-item"
38                 href="https://github.com/juanguialberto/tutorial-html5js"
39                 target="_blank">Versión</a>
40         </div>
    </li>
</ul>
</div>
</nav>

```

Guardamos todo y abrimos en un navegador. ¿Te ha funcionado?

Ejercicio horario de clase

Recuerda la semana pasada cómo hicimos una tabla. Esta semana te proponemos hacer el horario de clase que teníamos antes del confinamiento en formato Web para practicar y darle “estilo”. Como punto de partida, si quieres, puedes usar este modelo que te proponemos.

El diseño es libre. Puedes usar los colores que desees, se trata simplemente de explorar las opciones que nos ofrecen las hojas de estilo.

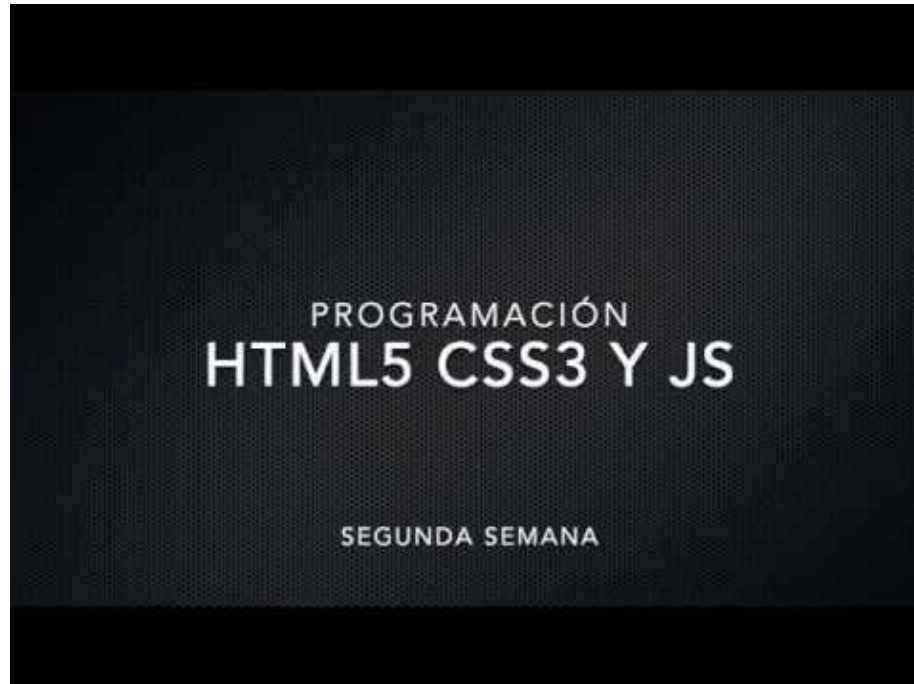
Crea una carpeta aparte “semana2” y hazlo ahí.

Horario de 2º Bachillerato Z

| | Lunes | Martes | Miércoles | Jueves | Viernes |
|---|-------------|------------|-----------|--------|---------|
| 1 | Matemáticas | Literatura | TIC 2 | Física | Inglés |
| 2 | Lengua | | | | |
| 3 | TIC 2 | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |

Wed Apr 22 2020 16:34:00 GMT+0200 (CEST)

Figure 19: Modelo de horario



Semana 4: JavaScript - ¡Hola Mundo!

JavaScript es uno de los lenguajes de programación más populares y usados mundialmente, además puede funcionar dentro de un navegador insertado en una página Web, lo que para nosotros lo hace muy atractivo. Sólo con un editor de texto sencillo (notepad, geany, vi...) y un navegador Web ya tenemos todo lo necesario para comenzar a ver resultados.

El desarrollo de aplicaciones

Crear aplicaciones para la Web, para móviles, ordenadores u otros dispositivos es un arte que, además de dotes de creatividad, requiere conocimientos de muchas disciplinas, como son:

- **Programación:** Cómo dar instrucciones al móvil o PC para que haga lo que quieres
- **Estructuras de datos:** Cómo almacenar, gestionar y manipular datos del mundo real en un programa
- **Ingeniería del software:** Nos ayuda a decidir los pasos a seguir en proyectos medianos y grandes
- **Algorítmica:** Resolver problemas reales con ayuda de la informática
- **Sistemas operativos:** Son el corazón de nuestros móviles y ordenadores. Nos proporcionan una interfaz amigable para interactuar con el hardware
- **Redes y comunicaciones:** Qué sería de nuestra vida hoy día sin estar conectados...
- **Bases de datos:** Los grandes almacenes de información

Y todo esto sin tener en cuenta las últimas tendencias en ciencia de datos (data science), inteligencia artificial (IA) o grandes volúmenes de datos (big data) o el internet de las cosas (IoT). Te animamos a explorar los enlaces anteriores para ir tomando conciencia del vasto universo de las ciencias de la computación.

Primeros pasos

La primera manera que veremos de insertar código JavaScript en HTML es usando la etiqueta `<script>` `</script>` y escribiendo dentro el código del programa que queramos correr.

Crea un archivo llamado **holajavascript.html** en tu editor favorito. Genera un esqueleto html5 y añade en el `<body>` el siguiente código JavaScript entre etiquetas `<script>` para que quede así:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
```

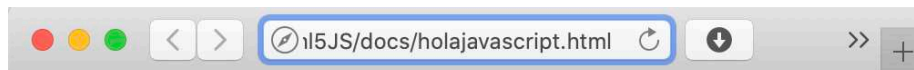


```

5   <meta name="viewport" content="width=device-width,
      initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>Mi primer programa JavaScript</h1>
10  <script>
11    document.write("<p>¡Hola Mundo!</p>");
12  </script>
13 </body>
14 </html>

```

Guarda el archivo y ábrelo en un navegador. Deberías ver algo como en la imagen.



Mi primer programa JavaScript

¡Hola Mundo!

Figure 20: Hola Mundo en JavaScript

Concretamente fíjate en la línea ¹

```
1 document.write("<p>Hola Mundo</p>");
```

esto es código JavaScript, es una instrucción o sentencia en este lenguaje. Las instrucciones las daremos con **palabras reservadas** especiales. Lo mismo que en HTML tenemos etiquetas que significan algo (ej. <html>, <head>, <body>, <p>, <h1>), en JavaScript (y en general en cualquier lenguaje de programación) hay una serie de palabras que el ordenador traducirá a instrucciones e intentará llevar a cabo la tarea que con ellas le hemos encomendado.

¹Si ya conoces JavaScript seguramente estarás pensando que esto podría ser más sencillo con un comando **console.log()**, pero para no liar más al lector con el modo desarrollo y la consola del navegador de momento usaremos esta opción aunque es peligroso escribir directamente en el árbol DOM.

Otra manera de cargar código JavaScript es en un fichero externo, para lo cual usaremos el atributo **src**; en el siguiente ejemplo nuestro código JavaScript estará en el fichero **index.js** que está en la carpeta o directorio **js**:

```
1 <script src="js/index.js">
```

Por norma general esta será la manera preferente de hacerlo.

Pero, ¿qué ocurre si el navegador no soporta JavaScript o el usuario lo tiene desactivado? Para eso tenemos la etiqueta `<noscript>` con la que podemos informar al usuario que debe activar JavaScript o actualizar su navegador para poder ver el contenido de nuestra página:

```
1 <noscript>
2   <p>Esta página necesita JavaScript que tenga habilitado.</p>
3 </noscript>
```

Sintaxis básica

Identificadores

Un identificador es el nombre que le podemos dar a:

- *una variable*: una variable es un pequeño espacio de la memoria del ordenador donde vamos a guardar un dato, una información concreta. Por ejemplo, si te digo que sumes 3 y 4 y que uses tus manos, tu mano derecha es una variable que ahora mismo almacena el dato 3 y tu mano izquierda es otra variable que guarda temporalmente el 4 hasta que finalmente haces la suma y devuelves 7. En un ordenador se pueden guardar tantas *manos* que hay que ponerles nombres...
- *una función*: creamos funciones para automatizar operaciones que repetimos continuamente. Imagina que quieres aplicar un descuento del X% al precio de un producto, tendrás que restar al precio inicial el resultado de multiplicar dicho precio inicial por el descuento y dividido por cien. Si esto tengo que hacerlo para muchos precios voy a tener que escribir mucho. Si creo una función sólo tengo que llamar a la función, por ejemplo *descuento(200)* y hará a 200 el descuento predeterminado.
- *una propiedad*: Imagina los descuentos de antes. Una propiedad (ya veremos más adelante cómo se hace) sería el valor del descuento. Cambiando la propiedad *descuento*, obtendré diferentes resultados.
- *un argumento de función*: En el ejemplo de los descuentos, cuando llamábamos la función *descuento(200)*, ese **200** será un argumento, un parámetro que le paso a la función (igual que en matemáticas).

JS es sensible a mayúsculas/minúsculas

JavaScript es sensible a mayúsculas y minúsculas (en inglés *case sensitive*) lo que quiere decir que no es lo mismo escribir *unavariabale* que *unaVariable*. Hay que tener mucho cuidado en no bailar letras ni cambiar mayúsculas por minúsculas (y viceversa).

Comentarios

Un comentario es una serie de palabras o líneas en el programa que no deben ejecutarse, simplemente están ahí para facilitar la vida del/la programador/a.

Pueden ser de una sólo línea o multilínea.

```
1 // esto es un comentario de una línea
2
3 /*
4 Esto es un
5 comentario que
6 ocupa más de
7 una línea.
8 */
```

Instrucciones o sentencias y bloques

Las instrucciones o sentencias en JavaScript **deben terminar siempre en punto y coma**, aunque se puede omitir y se tomará como siguiente instrucción la siguiente línea, pero esto puede dar lugar a resultados inesperados y no deberíamos hacerlo.

Ejemplo:

```
1 let a = 3; // definimos la variable "a" y guardamos en ella un 3
2 let b = 4; // definimos la variable "b" y guardamos en ella un 4
3 let suma = a+b; // definimos la variable "suma" y guardamos en ella
  un 7
```

En el ejemplo anterior hemos creado tres sentencias. Si quisiéramos agrupar sentencias en un bloque, usaremos las llaves: { y }. Ejemplo:

```
1 let test=true;
2 if ( test ) {
3     test = false;
4     console.log("Ahora test vale:"+test);
5 }
```

En el ejemplo anterior, si la variable *test* es verdadera (que lo es), entonces se ejecuta el bloque completo (las dos líneas que hay debajo de la condición). Aquí ya empezamos a ver palabras reservadas y que las variables pueden ser de varios tipos.

| Palabras | reservadas | en | JavaScript |
|----------|------------|------------|------------|
| await | do | import | throw |
| break | else | in | try |
| case | enum | instanceof | typeof |
| catch | export | interface | var |
| class | extends | let | void |
| const | finally | new | while |
| continue | for | return | with |
| debugger | function | super | yield |
| default | if | switch | |
| delete | implements | this | |

Fuente: Lista de palabras reservadas en JavaScript. Mozilla.

Las variables pueden ser definidas de tres maneras:

- con **var**, ejemplo: *var saludo='hola'*, la variable *saludo* guarda la cadena de caracteres *hola*.
- con **let**: define variables a nivel de bloque. Es la que te recomendamos usar en vez de *var*.
- con **const**: define una variable con un valor que no vamos a cambiar, es decir, una constante.
- sin poner nada: aunque se puede hacer, no es recomendable.

Tipos de datos

Copia y pega el siguiente código JavaScript en tu editor favorito. Llama al fichero *tiposjavascript.html* y guárdalo. Vamos a aprender los tipos de datos **number** (para números de cualquier tipo), **string** (cadenas de caracteres) y **boolean** (para booleanos: verdadero/falso). Fíjate en las operaciones matemáticas, en que hay diferentes tipos de datos...

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <title>Probando JS</title>
7   <meta name="description" content="">
```

```

8     <meta name="viewport" content="width=device-width,
        initial-scale=1">
9     <script src="js/index.js">
10    </script>
11 </head>
12 <body>
13     <h1> Hola Mundo JS </h1>
14     <!-- Comentario HTML -->
15     <script>
16         /* Comentario JavaScript */
17         /* llamada bloqueante. JS es no bloqueante salvo esto */
18         let nombre = prompt("Dime tu nombre:");
19         // esto imprime un párrafo donde nos saluda
20         document.write("<p>1) Hola "+nombre+" ¿Qué tal todo?</p>");
21     </script>
22     <h2> Vamos a ver más ejemplos </h2>
23     <script>
24         // VARIABLES TIPO NÚMERO
25         let numero = 2 + 3;
26         document.write("<p>2) 2+3=" + numero + " que es de tipo: " +
            typeof(numero) + "</p>");
27         numero = 2.3 + 5.9;
28         document.write("<p>3) 2.3+5.9=" + numero + " que es de tipo:
            " + typeof(numero) + "</p>");
29         numero = 2.3 + "5.9";
30         document.write("<p>4) 2.3+'5.9'=" + numero + " que es de
            tipo: " + typeof(numero) + "</p>");
31         numero = 2.3 + +'5.9';
32         document.write("<p>5) 2.3+ +'5.9'=" + numero + " que es de
            tipo: " + typeof(numero) + "</p>");
33         numero = 2.3 + -'5.9';
34         document.write("<p>6) 2.3+ -'5.9'=" + numero + " que es de
            tipo: " + typeof(numero) + "</p>");
35         // VARIABLES TIPO CADENA DE CARACTERES
36         let cadena = "Hola";
37         cadena += " Mundo";
38         // se pueden anidar unas comillas dobles dentro de simples y
            viceversa
39         document.write('<p>7) cadena = "' + cadena + '", que es de
            tipo: ' + typeof(cadena) + '</p>');
40         document.write("<p>8) cadena = '" + cadena + "', que es de
            tipo: " + typeof(cadena) + "</p>");
41         document.write('<p>9) cadena = \'\' + cadena + \'\'', que es de
            tipo: ' + typeof(cadena) + '</p>');
42         let variable = numero + cadena;

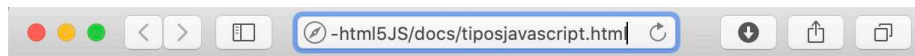
```

```

43     document.write('<p>10) variable= numero+cadena = \'' +
        variable + '\', que es de tipo: ' + typeof(variable) +
        '</p>');
44     // BOOLEANOS verdadero/falso
45     let booleano = true;
46     document.write("&<p>11) boolano= " + booleano + ", que es de
        tipo: " + typeof(booleano)+ '</p>');
47     booleano = "pepe";
48     document.write("<p>12) boolano= " + booleano + ", que es de
        tipo: " + typeof(booleano)+ '</p>');
49 </script>
50 </body>
51 </html>

```

¿Lo has probado? Debería verse algo como en la figura en tu navegador (tras contestar a la pregunta).



Hola Mundo JS

1) Hola Juan ¿Qué tal todo?

Vamos a ver más ejemplos

2) 2+3=5 que es de tipo: number

3) 2.3+5.9=8.2 que es de tipo: number

4) 2.3+'5.9'=2.35.9 que es de tipo: string

5) 2.3+ +'5.9'=8.2 que es de tipo: number

6) 2.3+ -'5.9'=-3.6000000000000005 que es de tipo: number

7) cadena = "Hola Mundo", que es de tipo: string

8) cadena = 'Hola Mundo', que es de tipo: string

9) cadena = 'Hola Mundo', que es de tipo: string

10) variable= numero+cadena = '-3.6000000000000005Hola Mundo', que es de tipo: string

11) booleano= true, que es de tipo: boolean

12) booleano= pepe, que es de tipo: string

Figure 21: Tipos básicos JavaScript

Ahora fíjate en la figura e intenta responder a las siguientes cuestiones :

1. ¿Para qué sirve la función *prompt()*?
2. ¿Qué tipo de dato usa JavaScript para los números enteros?
3. ¿Qué tipo de dato se usa para números decimales?
4. ¿Qué ocurre si ponemos entre comillas un número? ¿En qué se convierte? Ya no suma ahora, ¿qué ha pasado con los números?
5. ¿Qué le ocurre a una cadena de caracteres si le ponemos delante un símbolo más? (ojo, sólo si la cadena contiene números en el texto)
6. ¿Y si le ponemos un símbolo negativo?
7. ¿Qué pasa si ponemos comillas dobles dentro de comillas simples?
8. ¿Qué pasa si ponemos comillas simples dentro de comillas dobles?
9. ¿Te has fijado en cómo ahora pone comillas simples dentro de comillas simples? Eso se llama **escapar** las comillas (usando un carácter de escape, en este caso la barra invertida)
10. ¿Se pueden mezclar números con cadenas de caracteres? ¿Qué es el resultado final?
11. Los booleanos, la lógica, son muy importantes en programación
12. Aunque hemos definido una variable con otro tipo, al asignar se puede cambiar el tipo. Esto es muy peligroso

Operadores de comparación:

No es lo mismo el comparador estricto (`===` ó `!==`) que el regular (`==` ó `!=`). ¡Cuidado con los cambios de tipo!

```
1 // OPERADORES de COMPARACIÓN
2 var numero1 = 5;
3 var numero2 = '5';
4
5 document.write('numero1=5, numero2="5" ');
6
7 // COMPARACION REGULARES
8 var resultado = numero1 >= numero2;
9 document.write('<br>');
10 document.write('numero1 >= numero2 = ' + resultado);
11
12 resultado = numero1 <= numero2;
13 document.write('<br>');
14 document.write('numero1 <= numero2 = ' + resultado);
15
16 resultado = numero1 == numero2;
17 document.write('<br>');
18 document.write('numero1 == numero2 = ' + resultado);
19
20 resultado = numero1 != numero2;
21 document.write('<br>');
22 document.write('numero1 != numero2 = ' + resultado);
```

```

23
24 // COMPARACIÓN Estricta
25 var resultado = numero1 >= numero2;
26 document.write('<br>');
27 document.write('numero1 >= numero2 = ' + resultado);
28
29 resultado = numero1 <= numero2;
30 document.write('<br>');
31 document.write('numero1 <= numero2 = ' + resultado);
32
33 resultado = numero1 === numero2;
34 document.write('<br>');
35 document.write('numero1 === numero2 = ' + resultado);
36
37 resultado = numero1 !== numero2;
38 document.write('<br>');
39 document.write('numero1 !== numero2 = ' + resultado);

```

Conversiones de tipos de datos: Usamos `parseInt()` y `parseFloat()`. Cuidado que no redondean, sólo truncan (cortan) el resultado.

```

1 // EJEMPLO DE CONVERSION: "PARSEINT"
2 numero2 = '5.2 enanitos de blancanieves';
3 document.write('<br>numero1=' + numero1 + ', numero2="' + numero2 +
  '" ');
4 resultado = numero1 === parseInt(numero2);
5 document.write('<br>');
6 document.write('Usando parseInt (sólo coge los primeros enteros que
  encuentra) <br>');
7 document.write('numero1 === parseInt(numero2) = ' + resultado);
8
9 resultado = numero1 !== parseInt(numero2);
10 document.write('<br>');
11 document.write('numero1 !== parseInt(numero2) = ' + resultado);
12
13
14 // Introducción a ESTRUCTURA DE CONTROL if
15 document.write("<h1>Estructura de control IF-ELSE</h1>");
16 var dato1 = parseInt(prompt('Dame un número:'));
17 var dato2 = parseInt(prompt('Dame otro número:'));
18
19 if (isNaN(dato1) || isNaN(dato2)) {
20     document.write('Eso no era un número<br>');
21 } else {
22     if (dato1 <= dato2) {
23         document.write('El primero es menor o igual que el

```



```

        segundo<br>');
24   } else
25   if (dato1 >= dato2) {
26       document.write('El primero es mayor o igual que el
        segundo<br>');
27   } else {
28       document.write('Ninguno de los casos<br>');
29   }
30 }

```

Estructuras de control

Condicionales if-else

Si la expresión que hay entre paréntesis es cierta se ejecuta el código que hay entre los paréntesis (bloque), si existe un “else” y la condición es falsa, se ejecutará el código del segundo bloque.

```

1 let edad = prompt('Dime tu edad');
2 if (edad<18) {
3     document.write('<p>Eres menor de edad</p>');
4 } else {
5     document.write('<p>Eres un adulto</p>')
6 }

```

Switch..case...break

La estructura de control **switch** nos ayuda cuando tenemos múltiples opciones a evaluar. Por ejemplo, cuando llamas a un servicio de atención telefónica nos dice la locución: pulse 1 para compras, pulse 2 para ventas, pulse 3 para administración, etc. ¿Cómo se programa esto? Así:

```

1 let dato = 0;
2 switch (dato) {
3     case -1:
4         console.log('uno negativo');
5         break;
6     case 0: // foo es 0, por lo tanto se cumple la condición y se
              ejecutara el siguiente bloque
7         console.log('cero')
8         // NOTA: el "break" olvidado debería estar aquí
9     case 1: // No hay sentencia "break" en el 'case 0:', por lo tanto
              este caso también será ejecutado
10        console.log('uno');

```

```

11     break; // Al encontrar un "break", no será ejecutado el 'case 2:'
12   case 2:
13     console.log('dos');
14     break;
15   default:
16     console.log('default');
17 }

```

Bucle for

Bucle **for**: Repetimos un bloque de código tantas veces como indicamos en la variable *i* indicamos, en este caso la longitud del array (en el siguiente tema veremos qué son) Ejemplo:

```

1 let semana = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
2               "Sábado", "Domingo"];
3 document.write('<h1> Los días de la semana (bucle FOR..OF) </h1>');
4 console.log('Los días de la semana son:');
5 document.write('<br>');
6 for (let i=0; i<semana.length; i++) {
7   document.write("<p>" + semana[i] + "</p>");
8   console.log(semana[i]);
9 }
10 document.write('<br>');

```

Bucle **for..of**: para cada elemento de un objeto especial que llamamos *iterable* repetimos tantas veces como elementos (una por cada elemento) de dicho objeto iterable. Ejemplo:

```

1 let semana = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
2               "Sábado", "Domingo"];
3 document.write('<h1> Los días de la semana (bucle FOR..OF) </h1>');
4 console.log('Los días de la semana son:');
5
6 document.write('<br>');
7 for (let dia of semana) {
8   document.write("<p>" + dia + "</p>");
9   console.log(dia);
10 }
11 document.write('<br>');

```

Bucle **for..in**: Para un conjunto de atributos y su propiedad repetimos tantas veces como elementos tiene el conjunto. Ejemplo:

```

1 let semana = {l:"Monday", m:"Tuesday", x:"Wednesday", j:"Thursday",
  v:"Friday", s:"Saturday", d:"Sunday"};
2
3 document.write('<h1> Los días de la semana (bucle FOR) </h1>');
4 console.log('Los días de la semana son:');
5
6 document.write('<br>');
7 for (let dia in semana) {
8   document.write("<p>" + dia + " = " + semana[dia] + "</p>");
9   console.log(dia);
10 }
11 document.write('<br>');

```

Bucle while

Repite un bloque de código mientras la condición sea cierta. Puede ser que nunca lo sea, que nunca entremos. Ejemplo:

```

1 let countdown = 10;
2 document.write("<p> Empieza la cuenta atrás: </p>");
3 while(countdown>0) {
4   countdown = countdown - 1;
5   document.write("<p> Empieza la cuenta atrás: </p>");
6 }

```

Bucle do..while

Repite un bloque de código mientras la condición sea cierta. Como mínimo se repite una vez. Ejemplo:

```

1 let countdown = 10;
2 document.write("<p> Empieza la cuenta atrás: </p>");
3 do {
4   countdown = countdown - 1;
5   document.write("<p> Empieza la cuenta atrás: </p>");
6 }while(countdown>0) ;

```

Ejercicio tablas de multiplicar

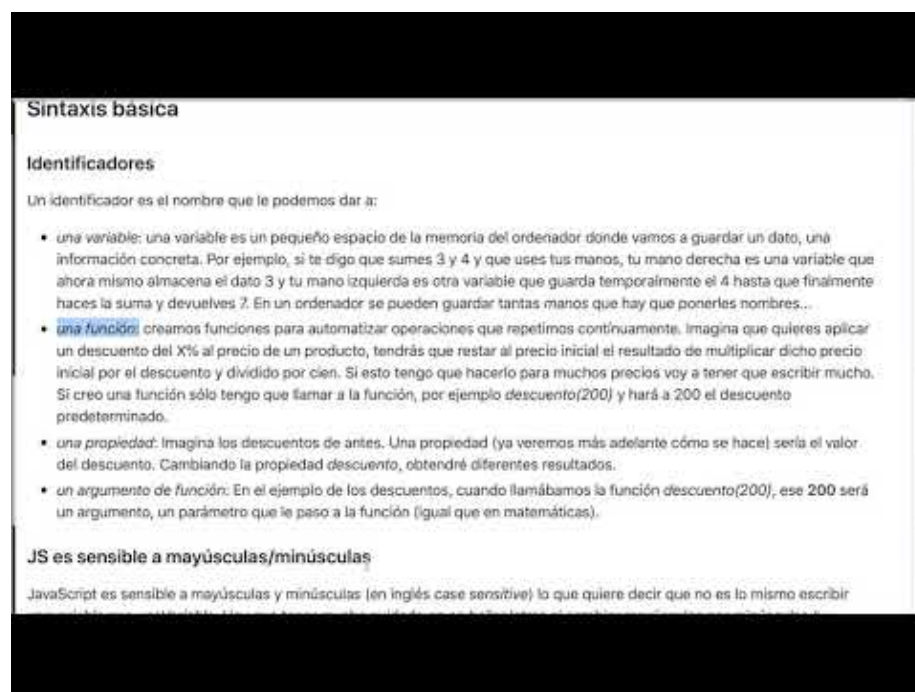
Hacer un programa que pregunte por un número del 1 al 10 (recuerda el ejemplo de la edad con el `if`). Guárdalo en una variable que vas a usar después para mostrar la tabla de multiplicar de ese número, usando para ello la estructura de control `for` (recuerda el ejemplo del primer *for* y los días de la semana).

Puedes mostrar la salida (`document.write`) en la página o en la consola (`console.log`).

Ejercicios resueltos y videotutorial de la semana

En Codepen.io tenéis los ejercicios resueltos.

En Youtube está el tutorial de la semana.



Sintaxis básica

Identificadores

Un identificador es el nombre que le podemos dar a:

- **una variable:** una variable es un pequeño espacio de la memoria del ordenador donde vamos a guardar un dato, una información concreta. Por ejemplo, si te digo que sumes 3 y 4 y que uses tus manos, tu mano derecha es una variable que ahora mismo almacena el dato 3 y tu mano izquierda es otra variable que guarda temporalmente el 4 hasta que finalmente haces la suma y devuelves 7. En un ordenador se pueden guardar tantas cosas que hay que ponerles nombres...
- **una función:** creamos funciones para automatizar operaciones que repetimos continuamente. Imagina que quieres aplicar un descuento del X% al precio de un producto, tendrás que restar al precio inicial el resultado de multiplicar dicho precio inicial por el descuento y dividido por cien. Si esto tengo que hacerlo para muchos precios voy a tener que escribir mucho. Si creo una función sólo tengo que llamar a la función, por ejemplo `descuento(200)` y hará a 200 el descuento predeterminado.
- **una propiedad:** Imagina los descuentos de antes. Una propiedad (ya veremos más adelante cómo se hace) sería el valor del descuento. Cambiando la propiedad `descuento`, obtendré diferentes resultados.
- **un argumento de función:** En el ejemplo de los descuentos, cuando llamábamos la función `descuento(200)`, ese 200 será un argumento, un parámetro que le paso a la función (igual que en matemáticas).

JS es sensible a mayúsculas/minúsculas

JavaScript es sensible a mayúsculas y minúsculas (en inglés case sensitive) lo que quiere decir que no es lo mismo escribir

Semana 5: JavaScript - Variables, clases, métodos y funciones

En esta semana vamos a aprender un poco más sobre variables, clases, métodos y funciones.

Conceptos clave:

1. **variable:** una zona de memoria donde almacenamos información de manera temporal. Se pueden definir usando *var*, *let* o *const*. Échale un ojo al apartado de los ámbitos para más información.
2. **objeto:** un objeto es una versión ampliada de una variable, es la representación interna en el ordenador de la información referente a algo que existe en el mundo real.
3. **función:** al igual que una sentencia o instrucción podría ser la mínima unidad de cómputo, una función debería ser una serie de bloques de código que engloban siempre las mismas instrucciones pero a los que podemos dar diferentes valores y así obtener diferentes resultados (ojo, las instrucciones o sentencias son siempre las mismas).
4. **clase:** De momento basta con que pensemos que será el esqueleto o patrón con el que vamos a crear objetos.
5. **método:** Para comunicarnos con los objetos, les pasamos mensajes. Para enviar o recibir un mensaje de un objeto, usaremos métodos.

Introducción

Ámbito de una variable

Podemos definir una variable con *var*, *let* y *const*. El último básicamente se usa cuando no vamos a cambiar el valor almacenado nunca, ejemplo:

```
1 const pi = 3.14159
```

Ahora bien, no es lo mismo usar *var* que *let*: **var** declara una variable de ámbito global o local para la función sin importar el ámbito de bloque, además permite levantamiento o hoisting; **let** declara una variable de ámbito global o local para la función o de bloque, es reasignable y *no* permite hoisting; **const** declara una variable de ámbito global, local para la función o de bloque; no es reasignable, pero es mutable y no permite hoisting.

| definición | ámbito | cambios | hoisting |
|------------|-----------------------------|---------|----------|
| var | global, bloque y subbloques | | sí |
| let | global, bloque y subbloques | | sí |
| const | global, bloque y subbloques | | no |

```

1 function test() {
2     let saludo = "hola"; // local variable
3 }
4 test();
5 console.log(saludo); // ¡error!

```

En el código anterior, la variable *saludo* sólo existen dentro de la función *test* (fíjate cómo está dentro de unas llaves, de un bloque). La variable *saludo* sólo existirá dentro del bloque donde fue definida y, de haber subbloques dentro, también ahí (salvo que creemos otra que se llame igual).

```

1 function comprueba1(booleano){
2     let saludo = "Hola";
3     if(booleano) {
4         let saludo = "Adiós";
5         console.log(saludo);
6     }
7     console.log(saludo);
8 }
9
10 function comprueba2(booleano){
11     var saludo = "Hola";
12     if(booleano) {
13         var saludo = "Adiós";
14         console.log(saludo);
15     }
16 }
17
18 comprueba1(true);
19 comprueba2(true);

```

En la función *comprueba1()* dentro del *if* definimos una nueva variable que no afecta para nada a la anterior. Sin embargo en *comprueba2()* como está definido con **var** cuando dentro del *if* queremos definir una nueva variable, no es así, estamos machacando la anterior *saludo*.

En JavaScript, las declaraciones (de variables o funciones) se mueven al principio de su ámbito. Este comportamiento se conoce como *lecantamiento* o *hoisting* y es muy importante tenerlo en cuenta a la hora de programar para prevenir posibles errores.

Teniendo en cuenta cómo funciona el *hoisting*, podemos llamar a una función y definirla más abajo, porque automáticamente JS la “subirá”.

Listas

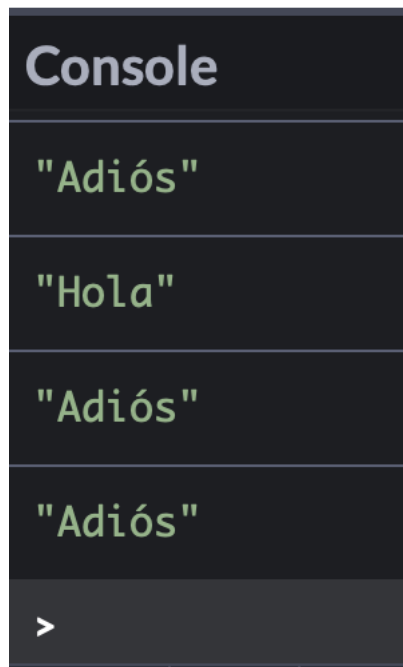


Figure 22: Ámbito de una variable con let y var: diferencias

```

1 let coches = ['VW', 'Seat', 'Audi', 'Porsche', 'Mercedes']
2
3 console.log(coches.length);
4 // 5
5 console.log(coches[0]);
6 // VW
7 console.log(coches[4]);
8 // Mercedes

```

Diccionarios

```

1 let persona = {
2   "nombre": "John",
3   "apellidos": "Doe",
4   "edad" : 18
5 };
6
7 console.log(persona["nombre"]);
8 // John

```

Ejemplo de función: creando el tablero de juego

Entre los programadores de aplicaciones y páginas Web, hoy día es habitual que parte de la ventana que estamos viendo no cambia nunca y otra parte se va mostrando, ocultando o incluso variando su contenido. Técnicamente estaríamos hablando de una aplicación en una página (*single applicaton page o SAP*) que se comunica asíncronamente con un servidor y va modificando dinámicamente lo que vemos con funciones JavaScript.

Eso es lo que vamos a hacer nosotros en este ejemplo. Recuperamos nuestro *index.html* del BuscaMinas y añadimos estas cajas:

```

1 // esto va justo después del <nav> </nav>
2 <div class="container">
3   <div id="panel_inicio" class="panel">
4     <p>Inicio</p>
5   </div>
6   <div id="panel_partida_facil" class="panel">
7     <p>Partida Fácil</p>
8   </div>
9   <div id="panel_partida_medio" class="panel">
10    <p>Partida Medio</p>
11  </div>
12  <div id="panel_partida_dificil" class="panel">
13    <p>Partida Dificil</p>
14  </div>
15  <div id="panel_ayuda" class="panel">

```



```

16     <p>Ayuda</p>
17 </div>
18 <div id="panel_licencia" class="panel">
19     <p>Licencia</p>
20 </div>
21 <div id="panel_puntuaciones" class="panel">
22     <p>Puntuaciones</p>
23 </div>
24 </div>

```

Añadimos el controlador. Este código es un objeto que se encarga de ir cambiando qué panel se ve en cada momento.

```

1 // fichero js/controlador.js
2 /**
3  * Biblioteca "casera" para hacer el "binding" del
4  * os menús con las diferentes vistas de la APP.
5  * Para usarla, basta con poner el mismo id a la entrada
6  * del menú que a su vista asociada, pero cambiando el prefijo,
7  * el el menú debe ser menu_AAA y en la vista panel_AAA.
8  */
9 $.controller = {};
10
11 /**
12  * Esta función gestiona qué panel está activo en cada momento (sólo
13  * puede
14  * haber uno)
15  * @param {type} panel_name el nombre del panel a activar
16  */
17 $.controller.activate = function (panel_name) {
18     // console.log("cambio old:"+$.controller.active_panel+"
19     // new:"+panel_name);
20     $($.controller.active_panel).hide();
21     $(panel_name).show();
22     $.controller.active_panel = panel_name;
23 };
24
25 /**
26  * Función para crear todos los "onClick" de los menús y
27  * asociarlos con cada panel correspondiente.
28  */
29 $.controller.active_panel = "";
30
31 $.controller.init = function (panel_inicial) {
32     console.log("Panel inicial="+panel_inicial);
33     $('[id^="menu_"]').each(function () {

```

```

32     var $this = $(this);
33     var menu_id = $this.attr('id');
34     var panel_id = menu_id.replace('menu_', 'panel_');
35
36     $("##" + menu_id).click(function () {
37         $.controller.activate("#" + panel_id);
38     });
39     // console.log("id_menu:" + menu_id + " id_panel" +
        panel_id);
40 });
41 $(".panel").hide();
42 $(panel_inicial).show();
43 $.controller.active_panel = panel_inicial;
44
45 }

```

Como los componentes, valores por defecto, etc. los ponemos en el fichero *js/index.js* hay que decirle que arranque el controlador:

```

1 /**
2  * Cuando la página se ha cargado entera, comenzamos:
3  *   - Inicializamos componentes
4  *   - Configuramos opciones
5  *   - Cargamos valores por defecto
6  *   - Etc...
7  */
8 $(function() {
9     /**
10     * Código para hacer que se cierre sólo el menú al pulsar sobre
        él
11     */
12     $('.navbar-nav li a').on('click', function(){
13         if(!$( this ).hasClass('dropdown-toggle')){
14             $('.navbar-collapse').collapse('hide');
15         }
16     });
17
18     /**
19     * Iniciamos el controlador que gestiona los eventos de
20     * los menús y activa/desactiva las vistas
21     */
22     $.controller.init("#panel_inicio");
23 });

```

Fichero Matriz.js

Esta semana tenemos que añadir un nuevo fichero a nuestro directorio “js” llamado “matriz.js” con el siguiente contenido:

```
1 /**
2  * Librería JavaScript para el manejo de Matrices 2D
3  * Ejemplo de uso:
4  * let mi_matriz = new Matriz(10, 10);
5  * mi_matriz.inicializa();
6  * mi_matriz.ponMinas(20);
7  * mi_matriz.ponContadores();
8  */
9
10 class Matriz {
11     /**
12      * Constructor, aloja espacio en memoria para
13      * una matriz de filas * columnas.
14      *
15      * @param {number} filas número de filas de la matriz.
16      * Si no se pasa valor (undefined) por defecto es 20.
17      * @param {number} columnas número de columnas de la matriz.
18      * Si no se pasa valor (undefined) por defecto es 20.
19      */
20
21     constructor(filas=20, columnas=20){
22         this._filas = filas;
23         this._columnas = columnas;
24         this._data = [];
25         for (let i=0; i<filas;i++){
26             this._data.push(new Array(columnas).fill(0));
27         }
28     }
29
30     /**
31      * Método para (re)inicializar la matriz con un valor
32      * determinado.
33      * Para el Buscaminas debe ser 0. Si no le pasamos nada por
34      * defecto
35      * será 0.
36      *
37      * @param {number} valor El dato con el que vamos a rellenar
38      * toda la matriz
39      */
40     inicializa(valor){
41         for(var i=0; i<this._data.length; i++) {
```

```

39         // for (var j=0; j<this._data[i].length; j++) {
40             this._data[i].fill(0);
41         // }
42     }
43 }
44
45 /**
46  * Coloca n_minas (con valor -1) en la matriz.
47  *
48  * @param {number} n_minas número de minas a colocar.
49  */
50 ponMinas(n_minas){
51     // Como mínimo que tengamos que poner una mina
52     let max_minas = Math.floor( (this._columnas * this._filas) /
53         3);
54     if ( (n_minas >= 1) && (n_minas < max_minas) ) {
55         this._minas=n_minas;
56         do {
57             // con LET estas variables sólo tienen como ámbito
58             // este bloque
59             let pos_fil = this.dado(this._filas);
60             let pos_col = this.dado(this._columnas);
61             // console.log("Matriz::ponMinas: intentando colocar
62             // mina en: "+pos_fil+","+pos_col);
63             // compruebo si no había mina previa
64             if (this._data[pos_fil][pos_col]!==(1)){
65                 this._data[pos_fil][pos_col]=(1);
66                 n_minas--;
67             }
68         } while (n_minas>0);
69     } else {
70         // enviar mensaje de error
71         throw new Error("Matriz::ponMinas:: número de minas no
72         válido");
73     }
74 }
75
76 /**
77  * Este método genera un aleatorio entre 0 y valor-1
78  *
79  * @param {number} valor genera un aleatorio entre 0 y valor-1
80  */
81 dado(valor) {
82     var tmp;
83     tmp = Math.floor(Math.random()*valor);
84     // console.log("DADO::"+tmp);

```

```

81     return tmp;
82 }
83
84 /**
85  * Métodos para hacer las comprobaciones acerca de las minas
86  */
87 miraNorte(i,j){
88     return this._data[i-1][j]==-1?1:0;
89 }
90 miraNO(i,j){
91     return this._data[i-1][j-1]==-1?1:0;
92 }
93 miraNE(i,j){
94     return this._data[i-1][j+1]==-1?1:0;
95 }
96 miraEste(i,j){
97     return this._data[i][j+1]==-1?1:0;
98 }
99 miraSE(i,j){
100     return this._data[i+1][j+1]==-1?1:0;
101 }
102 miraSur(i,j){
103     return this._data[i+1][j]==-1?1:0;
104 }
105 miraSO(i,j){
106     return this._data[i+1][j-1]==-1?1:0;
107 }
108 miraOeste(i,j){
109     return this._data[i][j-1]==-1?1:0;
110 }
111
112 /**
113  * Este método pone los contadores de las minas
114  * que hay alrededor de cada casilla.
115  */
116 ponContadores(){
117     for (let i=0; i<this._data.length; i++){
118         for(let j=0; j<this._data[i].length; j++){
119             if (this._data[i][j]!=-1) { // sólo si no hay mina
120                 hacemos las cuentas...
121                 if (i==0) { // estamos en la primera fila
122                     if (j==0) { // estamos en la primera columna
123                         // miramos sólo a la derecha y abajo
124                         this._data[i][j]+=this.miraEste(i,j);
125                         this._data[i][j]+=this.miraSE(i,j);
126                         this._data[i][j]+=this.miraSur(i,j);

```

```

126         } else {
127             if (j==(this._columnas-1)) { // estamos
                en la última columna
128                 // miramos abajo y a izquierda
129                 this._data[i][j]+=this.miraOeste(i,j);
130                 this._data[i][j]+=this.miraSO(i,j);
131                 this._data[i][j]+=this.miraSur(i,j);
132             } else { // estamos en las columnas
                centrales
133                 // miramos a derecha, izquierda y
                abajo
134                 this._data[i][j]+=this.miraOeste(i,j);
135                 this._data[i][j]+=this.miraEste(i,j);
136                 this._data[i][j]+=this.miraSO(i,j);
137                 this._data[i][j]+=this.miraSE(i,j);
138                 this._data[i][j]+=this.miraSur(i,j);
139             }
140         }
141     } else {
142         if (i==(this._filas-1)) { // estamos en la
            última fila
143             if (j==0) { // estamos en la primera
                columna
144                 // miramos sólo arriba y derecha
145                 this._data[i][j]+=this.miraNorte(i,j);
146                 this._data[i][j]+=this.miraNE(i,j);
147                 this._data[i][j]+=this.miraEste(i,j);
148             } else {
149                 if (j==(this._columnas-1)) { //
                    estamos en la última columna
150                     // miramos a la izquierda y
                    arriba
151                     this._data[i][j]+=this.miraNorte(i,j);
152                     this._data[i][j]+=this.miraNO(i,j);
153                     this._data[i][j]+=this.miraOeste(i,j);
154                 } else { // estamos en las columnas
                    centrales
155                     // miramos a derecha, izquierda
                    y arriba
156                     this._data[i][j]+=this.miraNorte(i,j);
157                     this._data[i][j]+=this.miraNO(i,j);
158                     this._data[i][j]+=this.miraOeste(i,j);
159                     this._data[i][j]+=this.miraEste(i,j);
160                     this._data[i][j]+=this.miraNE(i,j);
161                 }
162             }
        }
    }

```

```

163         } else { // estamos en las filas centrales
164             if (j==0) { // estamos en la primera
                columna
165                 // arriba, abajo y a la derecha
166                 this._data[i][j] += this.miraNorte(i,j);
167                 this._data[i][j] += this.miraSur(i,j);
168                 this._data[i][j] += this.miraSE(i,j);
169                 this._data[i][j] += this.miraEste(i,j);
170                 this._data[i][j] += this.miraNE(i,j);
171             } else {
172                 if (j==(this._columnas-1)) { //
                    estamos en la última columna
173                     // arriba, abajo y a la izquierda
174                     this._data[i][j] += this.miraNorte(i,j);
175                     this._data[i][j] += this.miraNO(i,j);
176                     this._data[i][j] += this.miraOeste(i,j);
177                     this._data[i][j] += this.miraSO(i,j);
178                     this._data[i][j] += this.miraSur(i,j);
179
180                 } else { // estamos en las columnas
                    centrales
181                     // arriba, abajo, derecha e
                        izquierda
182                     this._data[i][j] += this.miraNorte(i,j);
183                     this._data[i][j] += this.miraNO(i,j);
184                     this._data[i][j] += this.miraOeste(i,j);
185                     this._data[i][j] += this.miraSO(i,j);
186                     this._data[i][j] += this.miraSur(i,j);
187                     this._data[i][j] += this.miraSE(i,j);
188                     this._data[i][j] += this.miraEste(i,j);
189                     this._data[i][j] += this.miraNE(i,j);
190                 }
191             }
192         }
193     }
194 }
195
196 }
197
198
199 /**
200  * Método para consultar la posición [i,j]
201  * de la matriz del tablero del Buscaminas.
202  * @param {number} i Posición de la fila en el tablero.
203  * @param {number} j Posición de la columna en el tablero.
204  */

```

```

205     get(i,j){
206         if (i>=0 && i<this._filas && j>=0 && j<this._columnas) {
207             return this._data[i][j];
208         } else {
209             throw new Error("Matriz::get: Ha intentado acceder a una
                posición no válida.");
210         }
211     }
212
213     /**
214     * Método para almacenar un dato en la posición [i,j]
215     * de la matriz del tablero del Buscaminas. Si no se
216     * pasa el parámetro dato por defecto será 0.
217     * @param {number} i posición (fila)
218     * @param {number} j posición (columna)
219     * @param {number} dato Si no se indica, por defecto será 0.
220     */
221     set(i,j,dato=0) {
222         if (i>=0 && i<this._filas && j>=0 && j<this._columnas) {
223             this._data[i][j]=dato;
224         } else {
225             throw new Error("Matriz::set: Ha intentado acceder a una
                posición no válida.");
226         }
227     }
228
229     getFilas(){
230         return this._filas;
231     }
232
233     getColumnas(){
234         return this._columnas;
235     }
236
237     getMinas(){
238         return this._minas;
239     }
240
241     imprimeMatriz(){
242         let texto="";
243         for (let i=0; i<this._data.length; i++){
244             for(let j=0; j<this._data[i].length; j++){
245                 texto += "\t"+this._data[i][j];
246             }
247             texto += "\n";
248         }

```



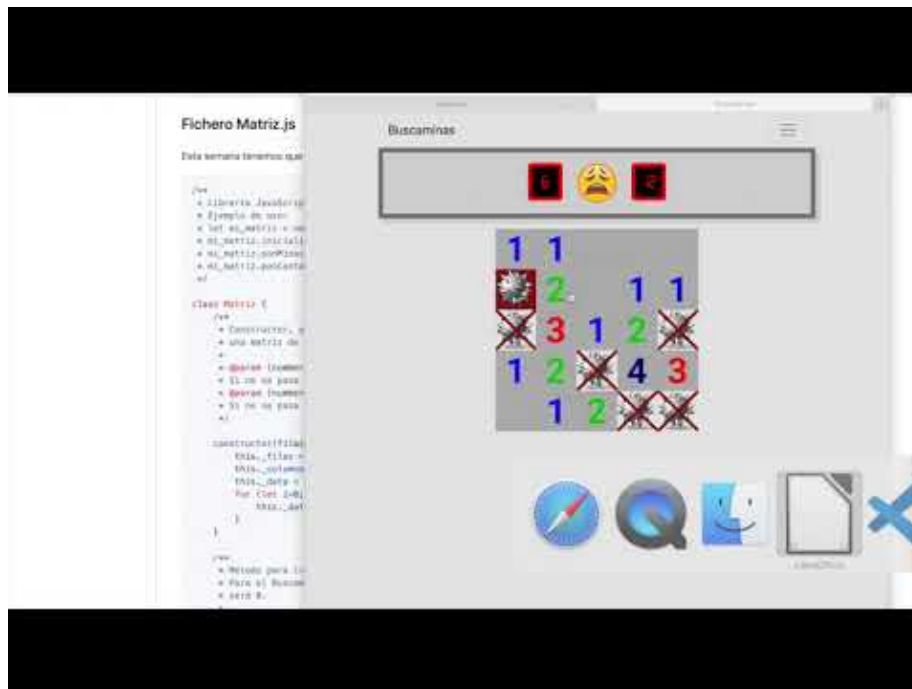
```

249     console.log(texto);
250   }
251 };

```

Ejercicio: Adivina en qué número pienso

Haz un programa que que nos proponga averiguar el número en el que estamos pensando. Cada vez que le demos un número, si no acertamos, nos avisará si es mayor o menor.



Semana 6: JavaScript - El árbol DOM

El Modelo de Objetos del Documento (del inglés Document Object Model) es una interfaz de programación de aplicaciones (lo que los programadores llamamos una API) para documentos bien formados y válidos. Define la estructura lógica de los documentos y el modo en que se accede y manipula.

Una página Web es un documento que internamente un ordenador *lo ve* como si fuese un árbol. Sea el siguiente código fuente (puedes acceder al original en esta Web):

```
1 <html>
2   <head>
3     <title>Trickier nesting, still</title>
4   </head>
5 <body>
6   <div id="main">
7     <div id="contents">
8       <table>
9         <tr>
10          <th>
11            Steps
12          </th>
13          <th>
14            Process
15          </th>
16        </tr>
17        <tr>
18          <td>
19            1
20          </td>
21          <td>
22            Figure out the
23            <em>root element</em>.
24          </td>
25        </tr>
26        <tr>
27          <td>
28            2
29          </td>
30          <td>
31            Deal with the
32            <span id="code">head</span>
33            first,
34            as 'its usually easy.
35          </td>
```

```

36         </tr>
37         <tr>
38             <td>
39                 3
40             </td>
41             <td>
42                 Work through the
43                 <span id="code">body</span>.
44                 Just <em>take your time</em>.
45             </td>
46         </tr>
47     </table>
48 </div>
49 <div id="closing">
50     This link is <em>not</em> active,
51     but if it were, the answers to this
52     <a href="answers.">html</a>
53     </a>
54     would be there. But
55     <em>do the exercise anyway!</em>
56 </div>
57 </div>
58 </body>
59 </html>

```

La representación **gráfica** que podemos hacer del mismo sería algo parecido a esto:

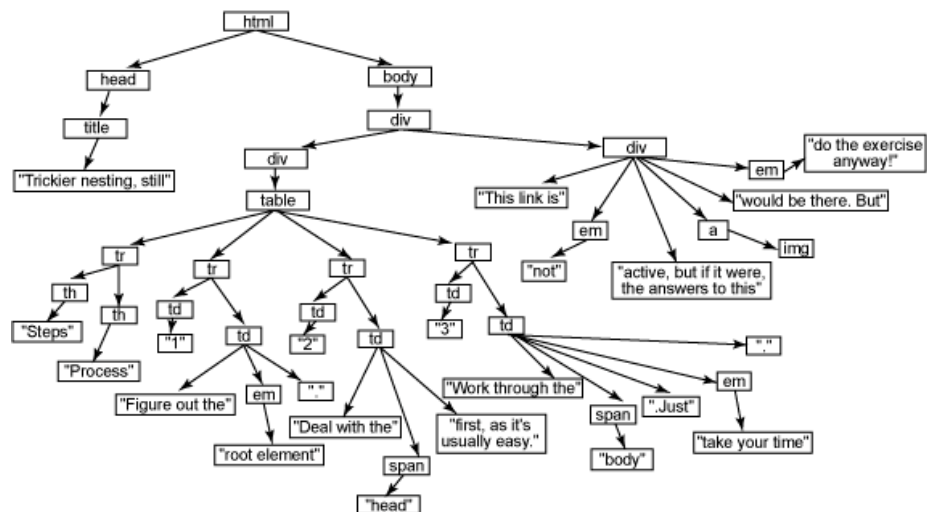


Figure 23: El árbol DOM

Es decir, tiene lo que en programación se llama *estructura de árbol*.

JavaScript nos proporciona herramientas para preguntar a cada nodo por sus hijos, sus padres, sus hermanos, el tipo de elemento, el contenido del mismo... A eso lo llamamos la API DOM.

jQuery

Para manipular el árbol DOM, aunque ya hay otras tecnologías más modernas, existe una biblioteca multiplataforma de JavaScript que permite hacerlo de una manera más sencilla.

Recuerda lo que vimos las semanas pasadas: tenemos una serie de “cajas” que llamábamos paneles, a los que ponemos un nombre gracias al atributo **id** que se puede poner a cualquier elemento HTML (no puede haber jamás dos elementos con el mismo *id*, es como el DNI de las etiquetas HTML) y le asignábamos una clase gracias al atributo **class** (esto nos ayudaba con los CSS u hojas de estilo). Veamos cómo funciona *jQuery* con unos ejemplos sobre manipulación de esta pieza de HTML:

```
1 <div class="container">
2   <div id="panel_inicio" class="panel">
3     <p>Inicio</p>
4   </div>
5   <div id="panel_partida_facil" class="panel">
6     <p>Partida Fácil</p>
7   </div>
8   <div id="panel_partida_medio" class="panel">
9     <p>Partida Medio</p>
10  </div>
11  <div id="panel_partida_dificil" class="panel">
12    <p>Partida Dificil</p>
13  </div>
14  <div id="panel_ayuda" class="panel">
15    <p>Ayuda</p>
16  </div>
17  <div id="panel_licencia" class="panel">
18    <p>Licencia</p>
19  </div>
20  <div id="panel_puntuaciones" class="panel">
21    <p>Puntuaciones</p>
22  </div>
23 </div>
```

Quiero seleccionar la caja con **id** “panel_inicio”, pues en jQuery sería (fíjate como ponemos el *id* entre comillas y le ponemos delante el carácter almohadilla):

```
1 let caja_inicio = $("#panel_inicio");
```

Quiero seleccionar todos los <div> de la página Web:

```
1 let todos_los_div = $("div");
```

Quiero seleccionar todos los <div> que sean de clase “panel”:

```
1 let divs_paneles = $("div.panel");
```

Quiero seleccionar cualquier elemento que sea de clase “panel”:

```
1 let paneles = $(".panel");
```

Quiero seleccionar todos los elementos con **id** que comienza por la palabra “panel_”:

```
1 let elementos = $('[id^="panel_"]');
```

Quiero ocultar todos los “paneles” (nuestros *div* con clase “panel”):

```
1 $(".div.panel").hide();
```

Quiero mostrar sólo el “panel de inicio”:

```
1 $(".div.panel").hide();  
2 $("#panel_inicio").show();
```

Quiero generar una tabla de 5x5 para la partida fácil y colgarla dentro del panel “partida_fácil”:

```
1 // almacenamos temporalmente en la variable "panel"  
2 // la caja o panel de partida fácil:  
3 let panel = $("#panel_partida_facil");  
4 // vaciamos el contenido del panel  
5 panel.empty();  
6 let mi_tabla = $("<table/>");  
7 // vamos a generar una tabla de 5 por 5  
8 for (let i=0; i<5; i++){  
9     // cada iteración de la i, se añade una fila (append)  
10    let fila = $("<tr></tr>");  
11    for (let j=0; j<5; j++){  
12        // cada iteración de la j, se añade una celda (append)  
13        let celda = $("<td id=\"celda_\"+i+\"_\"+j+\"\\>\"+\"</td>\");  
14        celda.addClass("vacio");  
15        fila.append(celda);  
16    }  
17    mi_tabla.append(fila);  
18 }  
19 // se añade la tabla al panel partida fácil (con append)  
20 panel.append(mi_tabla);
```

Fichero juego.js

Esta semana creamos el fichero **juego.js** en el directorio **js**:

```
1 // var $ = require('jQuery'); con ECMA7
2
3 /**
4  * Esta clase gestiona la partida de Buscaminas
5  */
6 class Juego {
7
8     /**
9      * Constructor, crea el objeto encargado del gestionar el juego
10      y las partidas
11      *
12      * @param {string} id_tiempo El identificador de la caja que va
13      a contener
14      * el tiempo en segundos de la partida
15      * @param {string} id_minas El identificador donde está la caja
16      que
17      * contiene el número de minas
18      * @param {string} id_carita El identificador donde está el
19      *smiley* que
20      * va cambiando conforme evoluciona la partida
21      * @param {string} id_tablero El identificador de la caja que va
22      a contener la tabla
23      * de las minas donde hacer clic
24      */
25     constructor(id_tiempo="#tiempo", id_minas="#minas",
26                 id_carita="#carita", id_tablero="#tablero"){
27         this.caja_tiempo=$(id_tiempo);
28         this.caja_minas=$(id_minas);
29         this.caja_carita=$(id_carita);
30         this.caja_tablero=$(id_tablero);
31         this.marcadores = new Marcadores();
32         this.timer=null;
33     }
34
35     pintaTablero(){
36         this.caja_tablero.empty();
37         let mi_tabla = $("<table/>");
38
39         // vamos a generar una tabla de n_filas por n_columnas
40         for (var i=0; i<this.matriz.getFilas(); i++){
41             // cada iteración de la i, se añade una fila
42             let fila = $("<tr></tr>");
```

```

38         for (var j=0; j<this.matriz.getColumnas(); j++){
39             // cada iteración de la j, se añade una celda
40             let celda = $("<td
                id=\"celda_\"+i+\"_\"+j+\">\"+\"</td>");
41             celda.addClass("vacio");
42             // celda.attr("onclick","alert(\"disparo en:
                \"+i+\", \"+j+\"'\");");
43             fila.append(celda);
44         }
45         mi_tabla.append(fila);
46     }
47     this.caja_tablero.append(mi_tabla);
48
49 }
50
51 partida(filas=10, columnas=10, minas=20){
52     this.caja_carita.removeAttr('class');
53     this.caja_carita.addClass('caraBanderita');
54     this.matriz = new Matriz(filas, columnas);
55     this.matriz.ponMinas(minas);
56     this.minas = minas;
57     this.filas = filas;
58     this.columnas = columnas;
59     this.aciertos = filas*columnas-minas;
60     this.matriz.ponContadores();
61
62     this.caja_minas.html(minas);
63     this.caja_tiempo.html(0);
64
65     this.pintaTablero();
66     // this.resuelve();
67
68     clearInterval(this.timer);
69     this.timer = setInterval(function(data){
70         data.html( +(data.html())+ 1);
71     }, 1000, this.caja_tiempo);
72
73     this.matriz.imprimeMatriz();
74 }
75
76 resuelve(perder){
77     if (this.matriz!==undefined) {
78         for (let i=0; i<this.matriz.getFilas();i++) {
79             for (let j=0; j<this.matriz.getColumnas();j++){
80                 if (this.matriz.get(i,j)==-1) {
81                     let td = $("#celda_\"+i+\"_\"+j);

```

```

82         td.removeAttr("class");
83         if (perder) td.addClass("bombaAnulada");
84         else td.addClass("bomba");
85     } else {
86         this.cambiaClase(i,j);
87     }
88 }
89 }
90 }
91 }
92
93 destapa(fila,columna){
94     if (fila>=0 && fila<this.filas && columna>=0 &&
95         columna<this.columnas) {
96         console.log("destapando "+fila+", "+columna);
97         // this.cambiaClase(fila,columna);
98         this.destapa(fila-1,columna-1);
99         this.destapa(fila+1,columna+1);
100     }
101 }
102
103 resuelveCelda(i,j){
104     let td = $("#celda_"+i+"_"+j);
105     if (this.aciertos>0){
106         this.aciertos--;
107         this.cambiaClase(i,j);
108     }
109     // hemos ganado!!
110     if (this.aciertos==0) {
111         this.resuelve(false);
112         clearInterval(this.timer);
113         this.caja_carita.removeAttr('class');
114         this.caja_carita.addClass('caraGanar');
115         // alert("Has ganado!!!");
116         let filas = this.filas;
117         let columnas = this.columnas;
118         let tiempo = (+this.caja_tiempo.html());
119         let puntos = Math.floor(
120             (this.minas*this.minas)/(filas*columnas*tiempo)*100000);
121         $('#puntosJugador').html(puntos);
122         $('#tiempoJugador').html(tiempo);
123         $("#modalPuntuacion").modal();
124     }
125 }

```



```

126 guardarPuntos(){
127     let filas = this.filas;
128     let columnas = this.columnas;
129     let tiempo = (+this.caja_tiempo.html());
130     let puntos = Math.floor(
131         (this.minas*this.minas)/(filas*columnas*tiempo)*100000);
132     this.marcadores.addMarcador($('#nombreJugador').val(),
133         puntos, tiempo, filas,
134         columnas, this.minas);
135     $('#puntosJugador').html(puntos);
136     $('#tiempoJugador').html(tiempo);
137     return (this.marcadores.getTabla());
138 }
139
140 cambiaClase(i,j){
141     if (this.matriz.get(i,j)==0) {
142         console.log("destapando "+i+","+j);
143         // this.destapa(i,j);
144     }
145     let td = $("#celda_"+i+"_"+j);
146     switch(this.matriz.get(i,j)){
147         case -1:
148             this.caja_carita.removeAttr('class');
149             this.caja_carita.addClass('caraPerder');
150             td.removeAttr("class");
151             td.addClass("bombaExplotada");
152             this.aciertos=-1;
153             clearInterval(this.timer);
154             // alert("Has perdido!!!");
155             this.matriz.set(i,j,-11);
156             this.resuelve(true);
157             break;
158         case 1:
159             td.removeAttr("class");
160             td.addClass("oneCell");
161             this.matriz.set(i,j,11);
162             break;
163         case 2:
164             td.removeAttr("class");
165             td.addClass("twoCell");
166             this.matriz.set(i,j,12);
167             break;
168         case 3:
169             td.removeAttr("class");
170             td.addClass("threeCell");

```

```

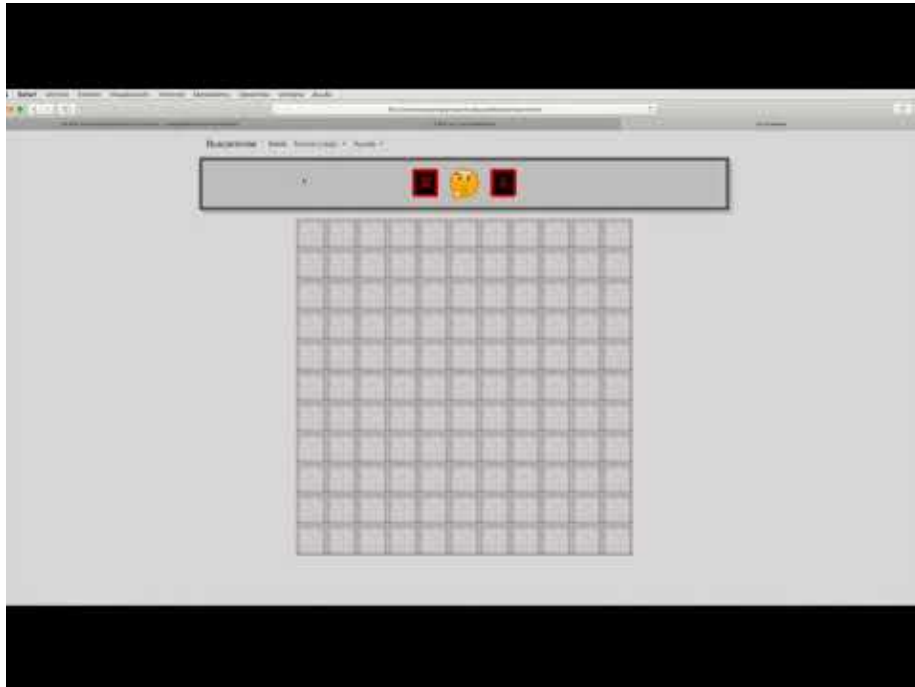
171         this.matriz.set(i,j,13);
172         break;
173     case 4:
174         td.removeAttr("class");
175         td.addClass("fourCell");
176         this.matriz.set(i,j,14);
177         break;
178     case 5:
179         td.removeAttr("class");
180         td.addClass("fiveCell");
181         this.matriz.set(i,j,15);
182         break;
183     case 6:
184         td.removeAttr("class");
185         td.addClass("sixCell");
186         this.matriz.set(i,j,16);
187         break;
188     case 7:
189         td.removeAttr("class");
190         td.addClass("sevenCell");
191         this.matriz.set(i,j,17);
192         break;
193     case 8:
194         td.removeAttr("class");
195         td.addClass("eighthCell");
196         this.matriz.set(i,j,18);
197         break;
198     case 0:
199         td.removeAttr("class");
200         td.addClass("nothingCell");
201         this.matriz.set(i,j,10);
202         break;
203     default:
204         this.aciertos++;
205         // no hacer nada, ya está resuelta...
206     }
207 }
208
209 disparo(caja) {
210     console.log("disparo");
211     let cadena = caja.id;
212     let pos = cadena.split("_");
213     this.resuelveCelda(+pos[1],+pos[2]);
214 }
215
216 limpiaTablero() {

```

```

217         if(this.caja_tablero!=undefined) {
218             this.caja_tablero.empty();
219             clearInterval(this.timer);
220             this.caja_minas.html(this.matriz.getMinas());
221         }
222     }
223 }

```



Semana 7: JavaScript - Pensamiento computacional. Algoritmos

Cálculo del dígito de control del NIF/NIE

El artículo 11 del Real Decreto 1553/2005, de 23 de diciembre, establece que el Documento Nacional de Identidad recogerá el número personal del DNI y carácter de verificación correspondiente al número de identificación fiscal.

• Para verificar el NIF de españoles residentes mayores de edad, el algoritmo de cálculo del dígito de control es el siguiente:

Se divide el número entre 23 y el resto se sustituye por una letra que se determina por inspección mediante la siguiente tabla:

Handwritten notes:

$12345678 \div 23 = 14$

$\begin{array}{r} D \\ \hline 14 \\ \hline R \end{array}$

| RESTO | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|
| LETRA | T | R | W | G | M | Y | P | D | X | B | | |

| RESTO | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|-------|----|----|----|----|----|----|----|----|----|----|----|
| LETRA | N | J | S | Q | V | H | L | C | K | E | |

Ejercicio: La letra del DNI

Semana 8: JavaScript - Almacenando datos

Fichero marcadores.js:

Esta semana añadimos el fichero marcadores.js a nuestro directorio js:

```
1 /**
2  * Esta clase gestiona los marcadores de un juego en LocalStorage
3  */
4 class Marcadores{
5
6     /**
7      * Crea un objeto de tipo Marcadores
8      * y carga la configuración inicial.
9      */
10    constructor(){
11        this.lista = [];
12        this.load();
13    }
14
15    /**
16     *
17     * @param {string} nombre
18     * @param {number} puntos
19     * @param {number} tiempo
20     * @param {number} filas
21     * @param {number} columnas
22     * @param {number} minas
23     */
24    addMarcador(nombre, puntos, tiempo, filas, columnas, minas){
25        let marcador = {
26            "nombre": nombre, // $("#nombre_jugador").val() lee del
27                           // formulario este input => <input id="nombre_jugador"
28                           // type="text />
29            "filas": filas, // (dentro de juego):
30                           // this.matriz.getFilas()
31            "columnas": columnas, // (dentro de juego):
32                           // this.matriz.getComlunas()
33            "minas": minas, // this.minas
34            "tiempo": tiempo, // +this.caja_tiempo.html()
35            "puntos": puntos // fórmula que calcula los puntos
36        };
37        this.lista.push(marcador);
38        this.lista.sort(function(a, b){return b.puntos-a.puntos});
39        this.save();
40    }
41 }
```

```

37
38  /**
39   * Carga de localStorage los marcadores.
40   */
41   load(){
42       if(localStorage!==undefined) {
43           let listado =
44               JSON.parse(localStorage.getItem('puntuaciones'));
45           if (listado!==null) {
46               // sobra esta ordenación, porque ya lo hacemos al
47               añadir
48               listado.sort(function(a, b){return
49                   b.puntos-a.puntos});
50               this.lista = listado;
51           }
52       } else {
53           throw new Error("Marcadores::load:: Almacenamiento no
54               disponible");
55       }
56   }
57
58   /**
59   * Guarda a localStorage los marcadores
60   */
61   save(){
62       if (localStorage!==undefined) {
63           localStorage.setItem('puntuaciones',
64               JSON.stringify(this.lista)) ;
65       } else {
66           throw new Error("Marcadores::save:: Almacenamiento no
67               disponible");
68       }
69   }
70
71   /**
72   * Devuelve el código HTML de una tabla con los puntos
73   */
74   getTabla() {
75       let html="<table>";
76       html+="<thead><th>#</th><th>Nombre</th><th>Puntos</th><th>Tiempo</th></thead>";
77       html+="<tbody>";
78       // Pintamos sólo las 20 puntuaciones más altas
79       let limite = this.lista.length > 20 ? 20 : this.lista.length;
80       for (let i=0; i<limite; i++) {
81           html+="<tr>";
82           html+="<td>"+(i+1)+"</td>";

```

```
77         html+="<td>"+this.lista[i].nombre+"</td>";
78         html+="<td>"+this.lista[i].puntos+"</td>";
79         html+="<td>"+this.lista[i].tiempo+"</td>";
80         html+="</tr>";
81     }
82     html+="</tbody>";
83     html+="</table>";
84     return html;
85 }
86
87 }
```