

Departamento de Informática

Acceso a Datos: Apuntes de Repaso de HTML5 y JavaScript

Juan Gualberto Gutiérrez Marín

Octubre 2023

Este trabajo está liberado bajo licencia Creative Commons

Sólo el conocimiento nos hace libres (Sócrates)



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

Índice

1	Introducción	5
1.1	Estructura básica de una Web	5
2	HTML5	6
2.1	Lenguajes de marcas	6
2.2	Primer ejemplo	6
2.3	Etiquetas más comunes de HTML	6
2.3.1	Las nuevas etiquetas de HTML5	6
2.3.2	Estructura básica	6
2.3.3	Títulos y párrafos	8
2.3.4	Formato de palabras	9
2.3.5	Hiperenlaces o anclas	10
2.3.6	Imágenes	10
2.3.7	Uso de audio y video en HTML5	10
2.3.8	Formularios en HTML5	11
2.3.9	Listas	12
2.3.10	La etiqueta <div>	13
3	El modelo de cajas	14
3.1	Cómo añadir CSS a una Web	16
3.1.1	Añadiendo estilos en la misma etiqueta	16
3.1.2	Añadiendo estilos en la cabecera de la página	17
3.1.3	Añadiendo estilos desde un archivo externo	18
3.2	El modelo de cajas	18
3.2.1	Altura y anchura de una caja	19
3.2.2	Margen	20
3.2.3	Relleno	20
3.2.4	Borde	20
3.2.5	Laterales	21
3.2.6	Más opciones para los bordes	21
3.2.7	Esquinas redondeadas	22
3.2.8	Sombras	22
3.2.9	Elementos flotantes	22
3.3	Tablas con estilo	23
3.3.1	Tablas en HTML	23

3.4	Ejercicio propuesto	25
3.4.1	Editores de texto	26
4	Diseño de interfaces de usuario	27
4.1	UI vs UX	27
4.2	Accesibilidad	27
5	CSS - Cascade Style Sheet	29
5.1	Reglas de estilo	30
5.2	Selección de elementos	32
5.3	Propiedades y valores	34
5.4	Cascada	35
5.5	Heredabilidad	39
5.6	Media Queries	41
6	BootStrap	46
6.1	Añadiendo BootStrap a nuestra página	46
6.2	El modelo de rejilla	46
6.3	Componentes de BootStrap	48
6.4	Cargando BootStrap desde CDN	50
7	JavaScript	53
7.1	Primeros pasos	53
7.2	Sintaxis básica	55
7.2.1	Identificadores	55
7.2.2	JS es sensible a mayúsculas/minúsculas	55
7.2.3	Comentarios	55
7.2.4	Instrucciones o sentencias y bloques	56
7.2.5	Tipos de datos	57
7.3	Estructuras de control	62
7.3.1	Condicionales if-else	62
7.3.2	Switch..case...break	62
7.3.3	Bucle for	62
7.3.4	Bucle while	64
7.3.5	Bucle do..while	64
7.4	Variables, clases, métodos y funciones	64
7.5	Variables	65
7.6	Objetos	66
7.7	Funciones	67

7.8	Clases en JavaScript	69
7.9	Ámbito de una variable	71
7.9.1	Arrays (arreglos) y Objetos (diccionarios)	72

1 Introducción

Este manual es una introducción a la programación con JavaScript y HTML5/CSS3.

1.1 Estructura básica de una Web

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.

En el lado del servidor realmente tenemos una serie de archivos que comprenden lo que llamamos la estructura básica de una Web.

2 HTML5

2.1 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).

2.2 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>
```

2.3 Etiquetas más comunes de HTML

2.3.1 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>.

2.3.2 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  </footer>
28 </body>
29 </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 (**English**).
- `<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.

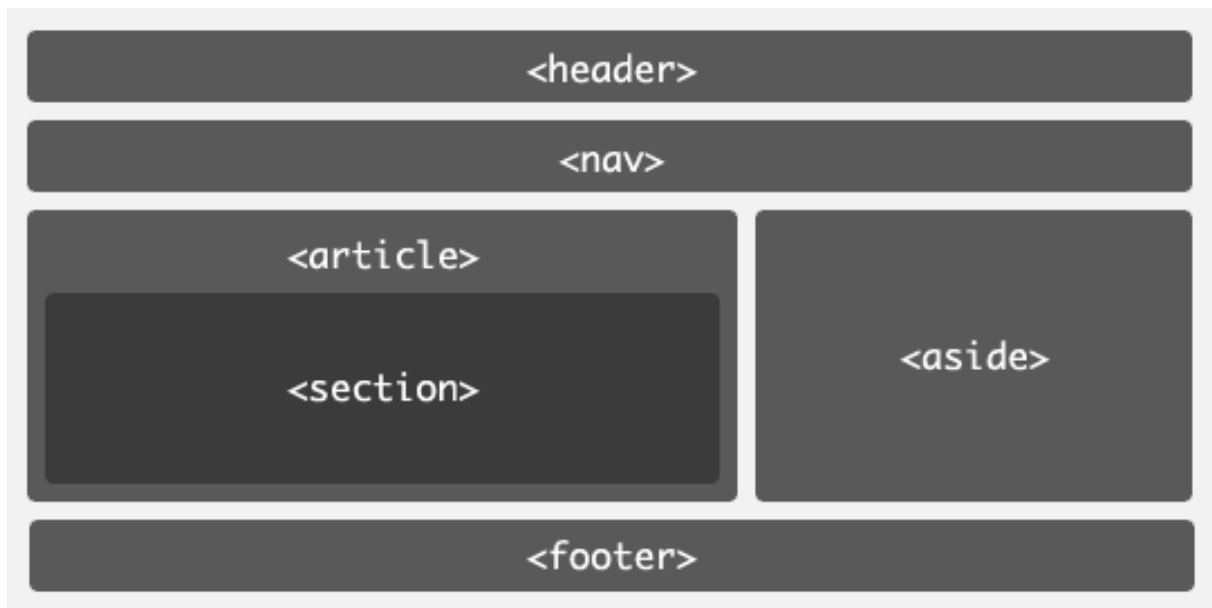


Figura 1: Esquema general de una página Web.

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.

2.3.3 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”.

2.3.3.1 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.

2.3.3.2 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.

2.3.4 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. </
   span>
4  </p>
5  <p>
6  Y esto otro párrafo con el siguiente texto en negrita
7  <span style="font-weight: bold;"> y esto un span dentro de un pá
   rrafo. </span>
8  </p>
9  <p>
10 Y esto otro párrafo con el siguiente texto en cursiva
11 <span style="font-style: italic;"> y esto un span dentro de un pá
   rrafo. </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.**

Figura 2: uso de span

2.3.5 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>
```

2.3.6 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 
```

2.3.7 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_película.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*”.

2.3.8 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:

Esto es un típico formulario Web

Nombre:

Apellido:

E-mail:

Edad:

Figura 3: Formulario información personal

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:

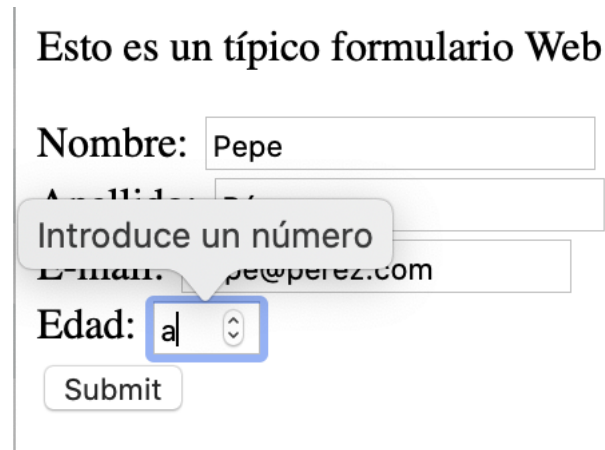


Figura 4: Formulario Restricción Edad

2.3.9 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

Figura 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

Figura 6: Lista ordenada

¿Has visto la diferencia entre usar y ?

2.3.10 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.

3 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):

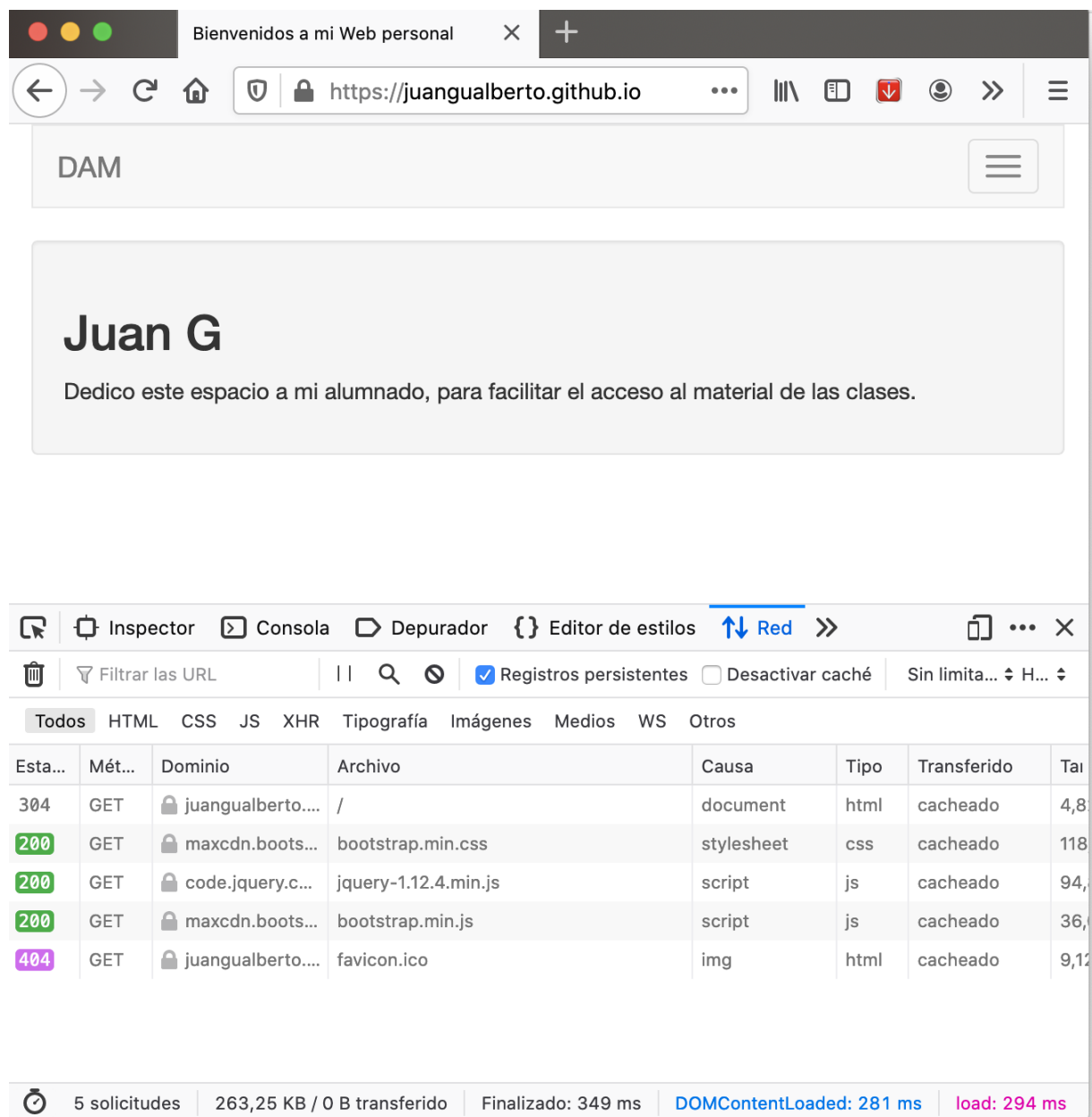


Figura 7: Firefox en modo Desarrollador Web

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**



Figura 8: Estructura de archivos

3.1 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.

3.1.1 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.

Figura 9: Estilos en elementos

3.1.2 Añadiendo estilos en la cabecera de la página

En la cabecera (dentro del elemento `<head></head>`) podemos indicar las reglas CSS que necesitemos, 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.

Figura 10: Estilos en elementos

3.1.3 Añadiendo estilos desde un archivo externo

Para añadir estilos desde un fichero externo, usamos la etiqueta **link** de este modo:

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

Donde el archivo `style.css` que está en la carpeta `css` contiene la hoja de estilo a cargar:

```
1 body {  
2     background-color: lightblue;  
3 }  
4  
5 h1,h2,h3 {  
6     color: white;  
7 }  
8  
9 p {  
10    font-family: verdana;  
11    font-size: 15px;  
12 }  
13  
14 li {  
15    font-family: sans-serif;  
16    font-size: 13px;  
17 }
```

3.2 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



Figura 11: modelo de cajas

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

3.2.1 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 contenido, 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 }
```

3.2.2 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; }
```

3.2.3 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; }
```

3.2.4 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.

3.2.5 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 }
```

3.2.6 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 estableceremos 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;}
```

3.2.7 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  }
```

3.2.8 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.

3.2.9 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.

3.3 Tablas con estilo

3.3.1 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.

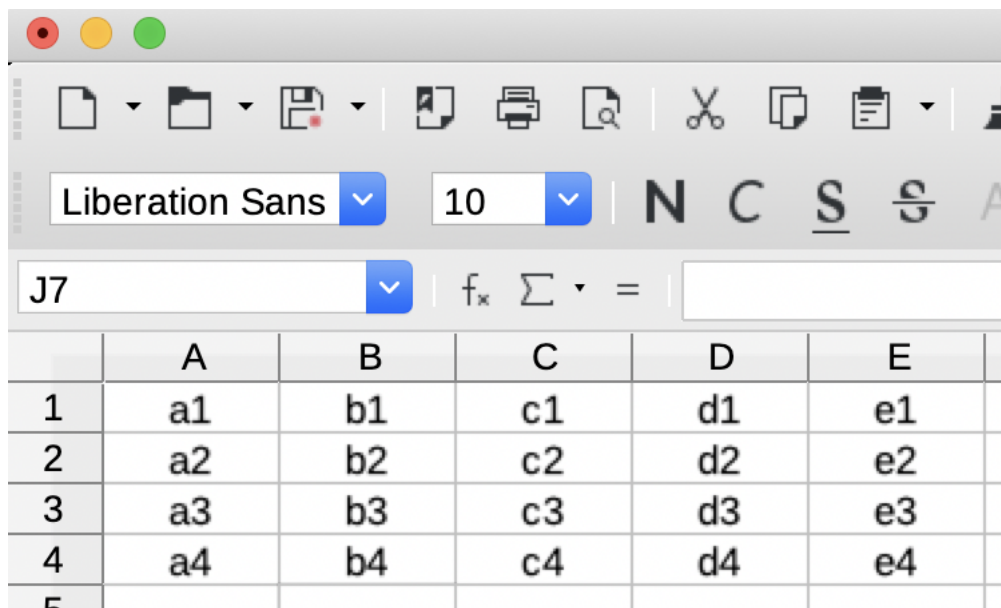
Ahora compara el contenido de la siguiente tabla HTML (busca lo que hay dentro de `<table>` y `</table>` en este ejemplo) con las siguientes imágenes (fíjate como le digo que haga una fila con las etiqueta `<tr>` y una celda con la etiqueta `<td>`):

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale
6     =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     </tr>
35   </tbody>
36 </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

Figura 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

Figura 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.

3.4 Ejercicio propuesto

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, initial-  
6             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 vías, las minas y el smiley -->  
18        </section>  
19        <section>  
20            <!-- Aquí estará el tablero de juego -->  
21        </section>  
22        <aside>  
23            <!-- Esto no será necesario y lo borraremos -->  
24        </aside>
```

```
24     </section>
25     <footer>
26         <!-- Aquí irá información adicional en el pie -->
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.

3.4.1 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 ése.

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 CodePen.

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

4 Diseño de interfaces de usuario

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.

4.1 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).

4.2 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

5 CSS - Cascade Style Sheet

CSS, que significa “Cascading Style Sheets” en inglés, es un lenguaje de hojas de estilo utilizado para controlar la presentación y el diseño de páginas web. CSS se utiliza junto con HTML (Hypertext Markup Language) y JavaScript para crear sitios web interactivos y visualmente atractivos.

Las hojas de estilo CSS se utilizan para definir cómo se debe mostrar el contenido HTML en un navegador web. Esto incluye aspectos como colores, fuentes, márgenes, tamaños de texto, espaciado entre párrafos, disposición de elementos en la página y muchas otras propiedades visuales.

Algunos conceptos clave de CSS incluyen:

1. **Reglas de Estilo:** Las reglas de estilo CSS consisten en un selector (que identifica los elementos HTML a los que se aplicará el estilo) y un bloque de declaración que contiene las propiedades de estilo y sus valores.

```
1 selector {  
2   propiedad: valor;  
3 }
```

2. **Selección de Elementos:** Los selectores pueden apuntar a elementos HTML específicos, como etiquetas, clases o identificadores. Por ejemplo, para aplicar estilos a todos los elementos de párrafo (<p>), puedes usar `p` como selector. Para aplicar estilos a elementos con una clase específica, puedes usar `.nombre-clase`.
3. **Propiedades y Valores:** Las propiedades son los atributos que se están configurando, como `color`, `font-size` o `margin`. Los valores son las opciones específicas que se asignan a esas propiedades, como `red`, `16px` o `10px 20px`.
4. **Cascada:** La “Cascada” en “Cascading Style Sheets” se refiere a la forma en que las reglas de estilo se aplican y se resuelven en una página web. Esto significa que las reglas pueden originarse en diferentes fuentes, como hojas de estilo externas, hojas de estilo internas o directamente en el HTML, y se aplican en un orden específico.
5. **Heredabilidad:** Los estilos pueden heredarse de elementos padres a elementos hijos en la jerarquía del documento. Esto significa que los estilos aplicados a un elemento padre pueden afectar a sus elementos hijos a menos que se anulen con reglas de estilo específicas.
6. **Media Queries:** Las “Media Queries” son una característica de CSS que permite definir estilos diferentes basados en las características del dispositivo, como el tamaño de la pantalla o la orientación. Esto facilita la creación de diseños responsivos que se adaptan a diferentes dispositivos y tamaños de pantalla.

5.1 Reglas de estilo

Las reglas de estilo CSS se utilizan para controlar la apariencia y el diseño de una página web, permitiendo a los desarrolladores y diseñadores web personalizar el aspecto de los elementos HTML de acuerdo con sus necesidades y requisitos de diseño.

Son instrucciones que se utilizan para definir cómo se debe presentar un elemento HTML en una página web. Cada regla de estilo consta de dos partes principales: el “selector” y el “bloque de declaración”. Veamos estos dos componentes con más detalle:

1. **Selector:** El selector es la parte de la regla de estilo que identifica el elemento HTML al que se aplicarán los estilos. Los selectores pueden ser muy específicos o más generales, y se utilizan para apuntar a uno o varios elementos en el documento HTML. Algunos ejemplos comunes de selectores incluyen:
 - **Selector de etiqueta:** Para aplicar un estilo a todos los elementos de una etiqueta HTML específica, como `p` para párrafos o `h1` para encabezados de nivel 1.
 - **Selector de clase:** Para aplicar un estilo a todos los elementos que tienen una clase específica, como `.destacado` para elementos con la clase “destacado”.
 - **Selector de ID:** Para aplicar un estilo a un elemento con un ID específico, como `#cabecera` para el elemento con el ID “cabecera”.
2. **Bloque de Declaración:** El bloque de declaración es donde se definen las propiedades de estilo y sus valores asociados. Estas propiedades determinan cómo se verá el elemento seleccionado. Un bloque de declaración se coloca entre llaves `{ }` y contiene una o más declaraciones de estilo. Cada declaración de estilo consiste en una propiedad y su valor, separados por dos puntos `(:)` y terminados con un punto y coma `(;)`. Por ejemplo:

```
1 selector {  
2   propiedad1: valor1;  
3   propiedad2: valor2;  
4 }
```

Aquí, `propiedad1` y `propiedad2` son ejemplos de propiedades de estilo, mientras que `valor1` y `valor2` son los valores asociados a esas propiedades.

Un ejemplo completo de una regla de estilo podría verse así:

```
1 p {  
2   color: blue;  
3   font-size: 16px;  
4 }
```

En este ejemplo, el selector es `p`, lo que significa que se aplicarán los estilos a todos los elementos de

párrafo (<p>) en el documento HTML. Las propiedades `color` y `font-size` se definen en el bloque de declaración, y los valores asociados son “blue” y “16px”, respectivamente. Esto resultará en que el texto dentro de los párrafos sea de color azul y tenga un tamaño de fuente de 16 píxeles.

Veamos algunos ejemplos de reglas de estilo CSS que demuestran diferentes propiedades y selectores:

1. Cambiar el color del texto de un encabezado:

```
1 h1 {  
2   color: red;  
3 }
```

En este caso, todos los elementos <h1> en la página tendrán su color de texto cambiado a rojo.

2. Estilo de enlace cuando el mouse pasa sobre él:

```
1 a:hover {  
2   text-decoration: underline;  
3   color: blue;  
4 }
```

Esta regla de estilo se aplicará a todos los enlaces (<a>) cuando el cursor del mouse pase sobre ellos. Cambia la decoración de texto a “subrayado” y el color del texto a azul.

3. Estilo de una clase específica:

```
1 .btn-destacado {  
2   background-color: yellow;  
3   color: black;  
4   padding: 10px;  
5 }
```

Esta regla se aplicará a todos los elementos que tienen la clase “btn-destacado”. Cambia el fondo a amarillo, el color de texto a negro y agrega un espacio de relleno de 10 píxeles.

4. Estilo de un ID específico:

```
1 #encabezado {  
2   font-size: 24px;  
3   text-align: center;  
4 }
```

Este estilo se aplicará a un elemento con el ID “encabezado”. Aumenta el tamaño de fuente a 24 píxeles y alinea el texto al centro.

5. Estilo de una lista desordenada:


```
1 ul {  
2   list-style-type: square;  
3   margin-left: 20px;  
4 }
```

Esta regla se aplicará a todas las listas desordenadas () en la página. Cambia el tipo de marcador de lista a cuadrado y agrega un margen a la izquierda de 20 píxeles.

6. Estilo de un elemento con múltiples selectores:

```
1 h2, h3 {  
2   font-weight: bold;  
3 }
```

Esta regla se aplicará tanto a los elementos <h2> como a los <h3> en la página, estableciendo el peso de la fuente en “negrita”.

Estos son solo ejemplos simples de reglas de estilo CSS. En el desarrollo web real, se utilizan muchas reglas de estilo para controlar la apariencia y el diseño de una página web de manera más compleja y detallada. Las propiedades y selectores en CSS son muy versátiles, lo que permite una amplia personalización de la presentación de un sitio web.

5.2 Selección de elementos

En CSS, los “selectores” son patrones que te permiten apuntar a elementos HTML específicos para aplicar reglas de estilo. Estos son algunos de los selectores CSS más comunes:

1. **Selector de Etiqueta:** Los selectores de etiqueta se utilizan para seleccionar todos los elementos con una etiqueta HTML específica. Por ejemplo, si deseas aplicar estilos a todos los párrafos en tu página, usarías el selector de etiqueta <p>.

```
1 p {  
2   color: blue;  
3   font-size: 16px;  
4 }
```

2. **Selector de Clase:** Los selectores de clase permiten seleccionar elementos que tienen una clase específica. Para aplicar estilos a elementos con una clase determinada, antepones un punto (.) al nombre de la clase.

```
1 .destacado {  
2   background-color: yellow;  
3   color: black;  
4 }
```

En el HTML, un elemento con la clase “destacado” se vería así: `<div class="destacado">Texto destacado</div>`.

3. **Selector de ID:** Los selectores de ID se utilizan para seleccionar un elemento con un ID específico. Para aplicar estilos a un elemento con un ID en particular, antepones un signo de almohadilla (#) seguido del nombre del ID.

```
1 #encabezado {
2   font-size: 24px;
3   text-align: center;
4 }
```

En el HTML, un elemento con el ID “encabezado” se vería así: `<h1 id="encabezado">Título del encabezado</h1>`.

4. **Selector Universal:** El selector universal (*) selecciona todos los elementos en la página. Puedes usarlo para aplicar estilos a todos los elementos, aunque se usa con precaución, ya que puede afectar a toda la página.

```
1 * {
2   margin: 0;
3   padding: 0;
4 }
```

5. **Selector de Atributo:** Los selectores de atributo permiten seleccionar elementos que tienen un atributo HTML específico. Puedes especificar el atributo y, opcionalmente, el valor que debe coincidir. Por ejemplo, `a[href]` seleccionaría todos los enlaces (`<a>`) con el atributo `href`, y `input[type="text"]` seleccionaría todos los elementos de entrada de texto (`<input>`) con el atributo `type` establecido en “text”.

```
1 a[href] {
2   text-decoration: underline;
3   color: blue;
4 }
```

6. **Combinación de Selectores:** Puedes combinar varios selectores en una sola regla para aplicar estilos a elementos que cumplen con más de una condición. Por ejemplo, `h1.destacado` seleccionaría todos los elementos `<h1>` con la clase “destacado”.

```
1 h1.destacado {
2   font-weight: bold;
3   color: red;
4 }
```

5.3 Propiedades y valores

Las propiedades y valores en CSS trabajan juntos para controlar la presentación de los elementos HTML en una página web. Combinando diferentes propiedades con valores adecuados, puedes personalizar el aspecto y el diseño de tus páginas web de manera efectiva y detallada.

A continuación veamos cómo funcionan:

1. **Propiedades:** Las propiedades en CSS son atributos que definen cómo se debe presentar un elemento HTML. Cada propiedad CSS especifica un aspecto visual o un comportamiento del elemento al que se aplica. Algunos ejemplos de propiedades CSS comunes incluyen:

- `color`: Define el color del texto.
- `font-size`: Establece el tamaño de la fuente.
- `background-color`: Define el color de fondo.
- `margin`: Controla el espacio exterior del elemento.
- `padding`: Controla el espacio interior del elemento.
- `border`: Establece el borde alrededor del elemento.

Aquí hay un ejemplo de cómo se utiliza una propiedad en una regla de estilo CSS:

```
1 p {  
2   color: blue; /* La propiedad 'color' establece el color del  
   texto en azul. */  
3 }
```

En este caso, la propiedad “color” se utiliza para definir el color del texto en todos los elementos de párrafo (<p>).

2. **Valores:** Los valores en CSS son las opciones específicas que se asignan a las propiedades. Cada propiedad espera un valor que debe indicar cómo se debe aplicar el estilo. Los valores pueden ser palabras clave, números, unidades de medida, colores, imágenes u otros valores admitidos por la propiedad en particular. Algunos ejemplos de valores CSS son:

- `"blue"`: Un valor de cadena que representa un color.
- `16px`: Un valor numérico seguido de una unidad de medida que establece el tamaño de la fuente.
- `#FF0000`: Un valor que representa un color en formato hexadecimal.
- `url('imagen.jpg')`: Un valor que especifica una imagen para usar como fondo.

Aquí hay un ejemplo de cómo se utiliza un valor en una regla de estilo CSS:

```
1 h1 {  
2   font-size: 24px; /* La propiedad 'font-size' se establece en 24  
   píxeles como valor. */  
3 }
```

```
3 }
```

En este caso, el valor “24px” se asigna a la propiedad “font-size” para definir el tamaño de la fuente de todos los encabezados de nivel 1 (<h1>).

5.4 Cascada

La “cascada” en CSS se refiere al proceso mediante el cual se determina cómo se aplicarán las reglas de estilo a los elementos HTML en una página web. La cascada se basa en un conjunto de reglas y prioridades que ayudan a resolver conflictos y determinar qué estilo prevalecerá cuando múltiples reglas afecten a un mismo elemento.

Veamos cómo funciona la cascada en CSS:

1. **Origen de las Reglas:** En una página web, las reglas de estilo CSS pueden provenir de diferentes fuentes o “origen”. Los principales orígenes de las reglas de estilo son:
 - **Hoja de estilo del usuario:** Estas son las preferencias de estilo definidas por el usuario en su navegador web, como el tamaño de la fuente predeterminado o la elección de colores de fondo y texto.
 - **Hoja de estilo del autor:** Estas son las reglas de estilo definidas por el desarrollador o diseñador web en el código CSS de la página. Pueden estar en hojas de estilo externas, internas o en línea en el documento HTML.
 - **Reglas de estilo de usuario importante:** Si el usuario especifica un estilo importante en sus preferencias, este tendrá prioridad sobre las reglas de estilo definidas por el autor de la página.
 - **Reglas de estilo de autor importante:** Si el desarrollador o diseñador web declara una regla como importante en el CSS, esta tendrá una alta prioridad.
2. **Especificidad:** Cuando varias reglas afectan a un mismo elemento, se utiliza la “especificidad” para determinar cuál regla prevalecerá. La especificidad se basa en el tipo de selector y la cantidad de selectores utilizados. Por ejemplo, una regla con un selector de ID (alta especificidad) tiene prioridad sobre una regla con un selector de clase (menos especificidad).
3. **Orden de Aparición:** Si las reglas tienen la misma especificidad, la última regla declarada prevalecerá. Esto significa que las reglas que aparecen más abajo en el archivo CSS o en el documento HTML anulan las reglas anteriores con la misma especificidad.
4. **Herencia:** Algunas propiedades CSS se heredan de elementos padres a elementos hijos. Por ejemplo, el tamaño de fuente definido en un elemento padre se puede heredar por los elementos hijos, a menos que se anule explícitamente en las reglas de estilo de los elementos hijos.

5. **Especificidad del Selector Universal:** El selector universal (*) tiene la menor especificidad, lo que significa que generalmente se sobrescribe con facilidad por reglas más específicas.

Lo mismo que con operaciones aritméticas tenemos la precedencia de operador, la cascada en CSS se encarga de resolver el orden y así determinar cómo se aplicarán las reglas de estilo a cada elemento en la página.

Veamos unos ejemplos a continuación: Supongamos que tienes un documento HTML simple con tres elementos `<p>` y has definido reglas de estilo CSS para ellos. Aquí está el HTML:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" type="text/css" href="estilos.css">
5 </head>
6 <body>
7   <p class="parrafo">Primer párrafo</p>
8   <p id="segundo">Segundo párrafo</p>
9   <p>Tercer párrafo</p>
10 </body>
11 </html>
```

Y aquí está el contenido de un archivo `estilos.css`:

```
1 /* Regla 1 */
2 p {
3   color: red;
4   font-size: 16px;
5 }
6
7 /* Regla 2 */
8 .parrafo {
9   color: blue;
10 }
11
12 /* Regla 3 */
13 #segundo {
14   font-size: 20px;
15   text-align: center;
16 }
17
18 /* Regla 4 */
19 p {
20   font-weight: bold;
21 }
```

En este ejemplo:

1. La **Regla 1** selecciona todos los elementos `<p>` y establece su color de texto en rojo y su tamaño de fuente en 16 píxeles.

2. La **Regla 2** selecciona elementos con la clase “parrafo” y cambia su color de texto a azul.
3. La **Regla 3** selecciona el elemento con el ID “segundo” y cambia su tamaño de fuente a 20 píxeles y su alineación de texto al centro.
4. La **Regla 4** nuevamente selecciona todos los elementos `<p>` y establece el peso de la fuente en negrita.

Ahora, veamos cómo se aplican estas reglas según el proceso de cascada:

- El primer párrafo tiene una clase “parrafo”, por lo que se ve afectado por la **Regla 2**. Su color de texto se establece en azul.
- El segundo párrafo tiene un ID “segundo”, por lo que la **Regla 3** tiene prioridad. Su tamaño de fuente se establece en 20 píxeles y su texto se alinea al centro. También hereda la propiedad `color` de la **Regla 1** y se mantiene en rojo.
- El tercer párrafo no tiene ninguna clase ni ID específicos, por lo que se aplica la **Regla 1** (color de texto en rojo y fuente en negrita).

En este ejemplo, la cascada resolvió los estilos aplicados a cada elemento según su especificidad y el orden de aparición en las reglas. Las reglas más específicas prevalecen sobre las menos específicas, y las reglas que aparecen más abajo en el archivo CSS tienen prioridad sobre las que aparecen arriba. Esto demuestra cómo las reglas de estilo se aplican en función de la cascada en CSS. Claro, un ejemplo ayudará a ilustrar cómo funciona la cascada en CSS. Supongamos que tienes un documento HTML simple con tres elementos `<p>` y has definido reglas de estilo CSS para ellos. Aquí está el HTML:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" type="text/css" href="estilos.css">
5 </head>
6 <body>
7   <p class="parrafo">Primer párrafo</p>
8   <p id="segundo">Segundo párrafo</p>
9   <p>Tercer párrafo</p>
10 </body>
11 </html>
```

Y aquí está el contenido de un archivo `estilos.css`:

```
1 /* Regla 1 */
2 p {
3   color: red;
4   font-size: 16px;
5 }
6
7 /* Regla 2 */
```

```
8  .parrafo {
9    color: blue;
10 }
11
12 /* Regla 3 */
13 #segundo {
14   font-size: 20px;
15   text-align: center;
16 }
17
18 /* Regla 4 */
19 p {
20   font-weight: bold;
21 }
```

En este ejemplo:

1. La **Regla 1** selecciona todos los elementos `<p>` y establece su color de texto en rojo y su tamaño de fuente en 16 píxeles.
2. La **Regla 2** selecciona elementos con la clase “parrafo” y cambia su color de texto a azul.
3. La **Regla 3** selecciona el elemento con el ID “segundo” y cambia su tamaño de fuente a 20 píxeles y su alineación de texto al centro.
4. La **Regla 4** nuevamente selecciona todos los elementos `<p>` y establece el peso de la fuente en negrita.

Ahora, veamos cómo se aplican estas reglas según el proceso de cascada:

- El primer párrafo tiene una clase “parrafo”, por lo que se ve afectado por la **Regla 2**. Su color de texto se establece en azul.
- El segundo párrafo tiene un ID “segundo”, por lo que la **Regla 3** tiene prioridad. Su tamaño de fuente se establece en 20 píxeles y su texto se alinea al centro. También hereda la propiedad `color` de la **Regla 1** y se mantiene en rojo.
- El tercer párrafo no tiene ninguna clase ni ID específicos, por lo que se aplica la **Regla 1** (color de texto en rojo y fuente en negrita).

En este ejemplo, la cascada resolvió los estilos aplicados a cada elemento según su especificidad y el orden de aparición en las reglas. Las reglas más específicas prevalecen sobre las menos específicas, y las reglas que aparecen más abajo en el archivo CSS tienen prioridad sobre las que aparecen arriba. Esto demuestra cómo las reglas de estilo se aplican en función de la cascada en CSS.

5.5 Heredabilidad

La heredabilidad es un concepto fundamental en CSS que se refiere a la capacidad de ciertas propiedades de estilo para “heredarse” de un elemento padre a sus elementos hijos. Esto significa que si aplicas un estilo a un elemento HTML, algunos de esos estilos se transmitirán a los elementos contenidos en su interior, a menos que se anulen con reglas de estilo específicas para los elementos hijos.

Algunas de las propiedades CSS que suelen heredarse de un elemento padre a sus elementos hijos incluyen:

1. **Propiedad `font`:** Esta propiedad incluye detalles de la fuente como tamaño, estilo y peso. Si aplicas un estilo de fuente a un elemento padre, los elementos hijos heredarán automáticamente esos estilos de fuente a menos que se especifiquen otros estilos para ellos.
2. **Propiedad `color`:** El color del texto suele heredarse. Si cambias el color del texto en un elemento padre, los elementos hijos también tendrán ese color de texto, a menos que se anule de manera específica.
3. **Propiedad `line-height`:** Esta propiedad controla el espaciado entre líneas de texto. Si estableces un valor para `line-height` en un elemento padre, los elementos hijos heredarán ese valor.
4. **Propiedad `text-align`:** Esta propiedad controla la alineación del texto. Si estableces la alineación del texto en un elemento padre, los elementos hijos heredarán esa alineación a menos que se modifique para ellos.
5. **Propiedad `list-style`:** Para listas ordenadas y desordenadas, las propiedades como `list-style-type` y `list-style-position` suelen heredarse. Por ejemplo, si cambias el tipo de marcador en una lista (`list-style-type`) en un elemento padre, los elementos de lista hijos también heredarán ese tipo de marcador a menos que se anule.

Sin embargo, no todas las propiedades se heredan. Las propiedades que controlan aspectos más estructurales o posicionamiento, como márgenes (`margin`) o posiciones (`position`), generalmente no se heredan, ya que no tendría sentido que se aplicaran automáticamente a los elementos hijos. En su lugar, estas propiedades se aplican de manera independiente a cada elemento.

La heredabilidad es útil para mantener un cierto grado de coherencia en la apariencia de una página web. Si deseas que todos los párrafos de tu sitio tengan un tamaño de fuente y un color de texto consistentes, puedes aplicar estos estilos a un elemento padre (como un contenedor de párrafos, por ejemplo un *DIV*) y los elementos hijos heredarán esos estilos. Sin embargo, también es importante entender cómo y cuándo anular la heredabilidad, ya que a menudo querrás personalizar los estilos de elementos hijos de manera específica.

Veamos un ejemplo: Supongamos que tienes un elemento `div` que contiene varios elementos `p` con un elemento hijo `span` dentro de uno de los párrafos. Vamos a aplicar estilos a los elementos padre y observar cómo se heredan o anulan en los elementos hijos:

HTML:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" type="text/css" href="estilos.css">
5 </head>
6 <body>
7   <div class="contenedor">
8     <p>Este es un párrafo de ejemplo.</p>
9     <p>Este es otro párrafo.
10      <span>Este es un elemento span dentro del párrafo.</span>
11    </p>
12  </div>
13 </body>
14 </html>
```

CSS (en el archivo `estilos.css`):

```
1 /* Estilos para el elemento div (elemento padre) */
2 .contenedor {
3   font-size: 20px; /* Tamaño de fuente grande para el contenedor */
4   color: blue;     /* Color de texto azul para el contenedor */
5 }
6
7 /* Estilos para los elementos p dentro del contenedor (elementos hijos
   directos) */
8 .contenedor p {
9   font-weight: bold; /* Peso de fuente en negrita para párrafos dentro
   del contenedor */
10 }
11
12 /* Estilos para el elemento span (elemento hijo dentro de un párrafo)
   */
13 .contenedor p span {
14   color: red; /* Color de texto rojo para el elemento span */
15 }
```

En este ejemplo:

1. El elemento `div` con la clase “contenedor” actúa como el elemento padre.
2. Se aplican estilos al elemento padre `.contenedor`:
 - Se establece el tamaño de fuente en 20 píxeles.
 - Se establece el color de texto en azul.

3. Se aplican estilos a los elementos `p` dentro del elemento padre (elementos hijos directos):
 - Se establece el peso de fuente en negrita.
4. Se aplican estilos al elemento `span` que se encuentra dentro de un párrafo (`p`) dentro del elemento padre:
 - Se establece el color de texto en rojo.

La heredabilidad y la especificidad de los selectores hacen que los estilos se apliquen de la siguiente manera:

- Los párrafos (elementos `p`) heredan el tamaño de fuente (20px) y el color de texto azul del elemento padre `.contenedor`. También heredan el peso de fuente en negrita de su propia regla de estilo.
- El elemento `span` dentro del segundo párrafo (con el texto “Este es un elemento span dentro del párrafo”) hereda el color de texto rojo de su regla de estilo específica. El tamaño de fuente y el color de texto azul del elemento padre no se aplican a este elemento.

En resumen, la heredabilidad permite que los elementos hijos hereden ciertos estilos de sus elementos padres, pero estos estilos pueden anularse o especificarse con reglas de estilo más específicas según sea necesario. Esto proporciona flexibilidad en la aplicación de estilos en una página web.

5.6 Media Queries

Las “media queries” (consultas de medios) permiten adaptar el diseño y la presentación de una página web en función de las características del dispositivo o el medio en el que se muestra. Las media queries son esenciales para la creación de **diseños web responsivos** (responsive web design), lo que significa que un sitio web puede adaptarse de manera inteligente a diferentes tamaños de pantalla y dispositivos, como computadoras de escritorio, tabletas y teléfonos móviles.

Las media queries se utilizan para aplicar estilos de CSS específicos cuando se cumplen ciertas condiciones. Las condiciones pueden estar relacionadas con aspectos como el ancho de la ventana del navegador, la orientación del dispositivo, la resolución de pantalla y otras características del dispositivo o del medio. Algunos ejemplos de consultas de medios comunes incluyen:

- Cambiar el tamaño del texto o el diseño de la página cuando la pantalla es más estrecha (p. ej., para dispositivos móviles).
- Ocultar o mostrar elementos específicos en función del tamaño de la pantalla.
- Adaptar la disposición de columnas en un diseño de cuadrícula según el ancho de la pantalla.

- Cambiar la fuente o los colores para una mejor legibilidad en pantallas pequeñas.

Las media queries se definen en CSS utilizando la regla `@media`. Aquí tienes un ejemplo simple de cómo se ve una media query en CSS:

```
1  /* Estilos generales para todos los dispositivos */
2  p {
3      font-size: 16px;
4  }
5
6  /* Media query: Cambiar el tamaño de fuente para pantallas pequeñas */
7  @media (max-width: 768px) {
8      p {
9          font-size: 14px;
10     }
11 }
```

En este ejemplo, los párrafos (`<p>`) tienen un tamaño de fuente de 16 píxeles en dispositivos en general. Sin embargo, cuando el ancho de la pantalla es igual o inferior a 768 píxeles (como en dispositivos móviles o pantallas pequeñas), se aplica una media query para cambiar el tamaño de fuente a 14 píxeles. Esto garantiza que el texto sea legible en pantallas más pequeñas.

Las media queries son esenciales para proporcionar una experiencia de usuario consistente y optimizada en diversos dispositivos y tamaños de pantalla. Al utilizar media queries de manera efectiva, los diseñadores web podemos crear diseños web que se adapten de forma dinámica y respondan a las necesidades de los usuarios en diferentes contextos.

Veamos ahora un ejemplo algo más complejo: Vamos a hacer un diseño de página sencillo que se adapta a dispositivos de escritorio, tabletas y teléfonos móviles.

HTML:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <link rel="stylesheet" type="text/css" href="estilos.css">
5  </head>
6  <body>
7      <header>
8          <h1>Mi Sitio Web</h1>
9          <nav>
10             <ul>
11                 <li><a href="#">Inicio</a></li>
12                 <li><a href="#">Acerca de</a></li>
13                 <li><a href="#">Servicios</a></li>
14                 <li><a href="#">Contacto</a></li>
15             </ul>
16          </nav>
17      </header>
```

```
18 <section class="contenido">
19   <article>
20     <h2>Artículo 1</h2>
21     <p>Este es el primer artículo de mi sitio web.</p>
22   </article>
23   <article>
24     <h2>Artículo 2</h2>
25     <p>Este es el segundo artículo de mi sitio web.</p>
26   </article>
27 </section>
28 <aside>
29   <h2>Barra lateral</h2>
30   <p>Contenido de la barra lateral.</p>
31 </aside>
32 </body>
33 </html>
```

CSS (en el archivo `estilos.css`):

```
1  /* Estilos generales para todos los dispositivos */
2  body {
3    font-family: Arial, sans-serif;
4  }
5
6  header {
7    background-color: #333;
8    color: white;
9    padding: 10px;
10   text-align: center;
11 }
12
13 nav ul {
14   list-style: none;
15   padding: 0;
16 }
17
18 nav li {
19   display: inline;
20   margin-right: 20px;
21 }
22
23 .contenido {
24   display: flex;
25   justify-content: space-between;
26 }
27
28 article {
29   width: 48%;
30   padding: 10px;
31   border: 1px solid #ccc;
32 }
```

```
33
34 aside {
35     width: 30%;
36     background-color: #f4f4f4;
37     padding: 10px;
38 }
39
40 /* Media query: Diseño para tabletas */
41 @media (max-width: 768px) {
42     header {
43         padding: 5px;
44     }
45     nav ul {
46         text-align: center;
47     }
48     .contenido {
49         flex-direction: column;
50     }
51     article {
52         width: 100%;
53     }
54     aside {
55         width: 100%;
56     }
57 }
58
59 /* Media query: Diseño para teléfonos móviles */
60 @media (max-width: 480px) {
61     nav li {
62         display: block;
63         margin: 5px 0;
64     }
65 }
```

En este ejemplo:

- El diseño base se aplica a todos los dispositivos y se define en las reglas generales de CSS. Se establece la fuente, el fondo del encabezado, los estilos de navegación y la disposición de contenido.
- Se utilizan dos media queries para adaptar el diseño a tamaños de pantalla más pequeños. Cuando el ancho de la pantalla es igual o inferior a 768 píxeles (como en tabletas), se aplican estilos específicos para tabletas, como centrar la navegación y cambiar la disposición de los elementos de contenido. Cuando el ancho de la pantalla es igual o inferior a 480 píxeles (como en teléfonos móviles), se aplican estilos adicionales, como convertir los elementos de navegación en bloques.

Este es solo un ejemplo básico de cómo las media queries se utilizan en el diseño web receptivo. En un

proyecto real, las consultas de medios se personalizarían aún más para adaptarse a las necesidades específicas de diseño y presentación en diferentes dispositivos y tamaños de pantalla.

6 Bootstrap

Bootstrap es un popular framework de diseño web de código abierto que se utiliza para desarrollar sitios web y aplicaciones web de manera más rápida y eficiente. Bootstrap proporciona una colección de herramientas, componentes y estilos predefinidos que facilitan la creación de interfaces de usuario atractivas y receptivas.

Algunas de las características más destacadas de Bootstrap incluyen:

1. **Grid System:** Bootstrap utiliza un sistema de rejilla (grid system) que facilita la creación de diseños de páginas web responsivos. Esto permite que los elementos de la página se adapten de manera automática a diferentes tamaños de pantalla, como en dispositivos móviles y tablets.
2. **Componentes Reutilizables:** Bootstrap ofrece una amplia gama de componentes predefinidos, como botones, formularios, paneles, alertas, barras de navegación, entre otros. Estos componentes son fáciles de integrar en tus proyectos y ahorran tiempo de desarrollo.
3. **Estilos y Temas:** Bootstrap incluye estilos CSS predefinidos que hacen que los sitios web sean visualmente atractivos. Además, puedes personalizar los temas y estilos para que se adapten a tus necesidades específicas.
4. **JavaScript Interactivo:** Bootstrap también proporciona una serie de complementos de JavaScript, como modales, carruseles, pestañas y más, que permiten agregar interactividad a tu sitio web sin tener que escribir código JavaScript desde cero.
5. **Documentación Completa:** Bootstrap ofrece una documentación exhaustiva con ejemplos y guías detalladas que facilitan su aprendizaje y uso.

Bootstrap es ampliamente utilizado en la industria del desarrollo web y es especialmente popular para la creación de sitios web receptivos y amigables para dispositivos móviles. Al utilizar Bootstrap, los desarrolladores pueden acelerar el proceso de desarrollo y garantizar una apariencia y funcionalidad consistentes en sus proyectos web.

6.1 Añadiendo Bootstrap a nuestra página

6.2 El modelo de rejilla

¡Claro! El modelo de rejilla (grid system) de Bootstrap es uno de los componentes clave de este framework y se utiliza para crear diseños de páginas web receptivos y flexibles. Permite organizar y distribuir elementos en una página web en una estructura de filas y columnas, lo que facilita la adaptación de la página a diferentes tamaños de pantalla, como en dispositivos móviles, tablets y ordenadores de escritorio.

A continuación, se describen los conceptos básicos del sistema de rejilla de Bootstrap:

1. **Filas (Rows):** En Bootstrap, una fila es el contenedor principal que alberga las columnas. Se utiliza la clase `.row` para definir una fila. Cada fila se divide en 12 columnas en total. Las filas se utilizan para agrupar contenido relacionado y asegurarse de que se ajusten correctamente en diferentes tamaños de pantalla.
2. **Columnas (Columns):** Las columnas son los elementos que se utilizan para dividir una fila. Puedes especificar cuántas columnas ocupa un elemento utilizando las clases como `.col-`, `.col-sm-`, `.col-md-`, `.col-lg-`, etc. El número después de “col-” indica cuántas de las 12 columnas totales de la fila debe ocupar el elemento. Por ejemplo, `col-6` significa que el elemento ocupará la mitad del ancho de la fila.
3. **Tamaños de Pantalla (Breakpoints):** Bootstrap ofrece clases para especificar el número de columnas que un elemento debe ocupar en diferentes tamaños de pantalla. Estos tamaños de pantalla se denominan “breakpoints” y se corresponden con los tamaños comunes de dispositivos, como extra pequeño (`.col-xs-`), pequeño (`.col-sm-`), mediano (`.col-md-`), y grande (`.col-lg-`).
4. **Anidamiento de Columnas:** Puedes anidar columnas dentro de otras columnas para crear diseños más complejos. Esto te permite organizar contenido en múltiples niveles de profundidad.
5. **Offsets:** Bootstrap también permite establecer desplazamientos u “offsets” en columnas para crear márgenes y alinear elementos de manera precisa.

Aquí tienes un ejemplo simple de cómo se ve el sistema de rejilla de Bootstrap en el código HTML:

```
1 <div class="container">
2   <div class="row">
3     <div class="col-12 col-sm-6 col-md-4 col-lg-3">Columna 1</div>
4     <div class="col-12 col-sm-6 col-md-4 col-lg-3">Columna 2</div>
5     <div class="col-12 col-sm-6 col-md-4 col-lg-3">Columna 3</div>
6     <div class="col-12 col-sm-6 col-md-4 col-lg-3">Columna 4</div>
7   </div>
8 </div>
```

En este ejemplo, hemos creado una fila con cuatro columnas que ocupan diferentes cantidades de espacio según el tamaño de la pantalla. La clase `.container` se utiliza para contener el contenido y establecer un ancho máximo. Las clases `.col-` determinan el ancho de las columnas en función del tamaño de la pantalla.

Puedes ampliar información haciendo clic sobre este enlace.

6.3 Componentes de Bootstrap

Bootstrap ofrece una amplia variedad de componentes reutilizables que facilitan el desarrollo de sitios web y aplicaciones. Estos componentes están diseñados para proporcionar una apariencia y funcionalidad coherentes en tus proyectos. A continuación, te presento algunos de los principales componentes reutilizables de Bootstrap:

1. **Botones (Buttons):** Bootstrap ofrece estilos de botones predefinidos que puedes aplicar a elementos `<button>`, `<a>` y `<input>`. Puedes personalizarlos con diferentes tamaños, colores y estilos. Estos estilos predefinidos incluyen tamaños, colores y estilos.
2. **Formularios (Forms):** Bootstrap facilita la creación de formularios atractivos y funcionales. Esto incluye campos de entrada, áreas de texto, casillas de verificación, botones y grupos de botones de opción.
3. **Barras de Navegación (Navbar):** Las barras de navegación de Bootstrap son altamente personalizables y se utilizan para crear menús de navegación. Puedes agregar elementos de menú, logotipos y ajustar el comportamiento de la barra de navegación en dispositivos móviles.
4. **Tarjetas (Cards):** Las tarjetas son componentes versátiles que se utilizan para mostrar contenido en un formato similar a una tarjeta. Puedes incluir imágenes, texto y botones en las tarjetas.
5. **Modales (Modals):** Los modales son ventanas emergentes que se utilizan para mostrar contenido adicional o realizar acciones específicas. Bootstrap proporciona estilos y funcionalidades para la creación de modales.
6. **Carrouseles (Carousels):** Los carrouseles son útiles para crear presentaciones de imágenes o contenido que se deslizan automáticamente o mediante navegación. Bootstrap incluye un componente de carrusel con opciones de personalización.
7. **Pestañas (Tabs):** Bootstrap te permite crear pestañas para organizar contenido en secciones. Los usuarios pueden alternar entre pestañas para ver información específica.
8. **Alertas (Alerts):** Las alertas son cuadros de mensaje utilizados para mostrar información importante o mensajes de error. Bootstrap proporciona estilos para alertas de éxito, información, advertencia y error.
9. **Botones de Grupo (Button Groups):** Puedes agrupar botones relacionados en un conjunto y aplicar estilos de grupo, lo que es útil para crear grupos de botones de opción, por ejemplo.
10. **Listas (List Groups):** Bootstrap ofrece estilos de lista que te permiten crear listas simples o listas interactivas con elementos que se pueden hacer clic y seleccionar.
11. **Dropdowns:** Los dropdowns son menús desplegables que permiten a los usuarios seleccionar una opción de una lista. Bootstrap proporciona estilos y funcionalidades para crear dropdowns

interactivos.

12. **Íconos (Icons):** Bootstrap incorpora iconos de FontAwesome y Glyphicons que puedes utilizar en tus proyectos para agregar símbolos gráficos a tus elementos.
13. **Acordeones (Accordions):** Los acordeones son componentes colapsables que te permiten mostrar información de manera organizada y colapsar secciones para ahorrar espacio.
14. **Botones de Acción (Action Buttons):** Bootstrap ofrece estilos de botones de acción que son ideales para acciones específicas, como agregar, eliminar, editar, guardar, etc.
15. **Badges:** Los badges son etiquetas informativas que se pueden agregar a elementos para mostrar información adicional, como recuentos o estados.
16. **Jumbotron:** El componente Jumbotron se utiliza para resaltar contenido o mensajes importantes en la parte superior de una página web. Puedes agregar títulos, texto descriptivo y botones de llamada a la acción en un diseño atractivo.
17. **Popovers:** Los popovers son elementos emergentes que aparecen al hacer clic en un elemento y pueden mostrar información adicional, como descripciones o notas.
18. **Tooltips:** Los tooltips son pequeñas ventanas emergentes que aparecen al pasar el cursor sobre un elemento y proporcionan información adicional o descripciones breves.
19. **Barra de Progreso (Progress Bar):** Bootstrap permite agregar barras de progreso para mostrar visualmente el avance de tareas o procesos.
20. **Tipografía (Typography):** Bootstrap incluye estilos tipográficos predefinidos para títulos, párrafos, listas y otros elementos de texto.
21. **Navegación por Breadcrumb:** Los breadcrumbs se utilizan para mostrar la ubicación de una página dentro de la estructura del sitio web.
22. **Grupos de Entradas (Input Groups):** Puedes agrupar elementos de entrada (como campos de texto o botones) con texto adicional o botones de acción.
23. **Imágenes (Images):** Bootstrap proporciona clases para controlar la alineación, tamaño y formas de las imágenes.
24. **Medios Embebidos (Embedded Media):** Puedes insertar medios embebidos, como videos de YouTube o mapas de Google, en tus páginas web de manera sencilla.
25. **Utilidades de Espaciado (Spacing Utilities):** Bootstrap

incluye clases de utilidad para controlar el espaciado y el margen entre elementos.

26. **Utilidades de Texto (Text Utilities):** Puedes aplicar clases de utilidad para controlar el alineamiento, el color y otros estilos de texto.

Estos son los principales componentes de Bootstrap. Puedes ver ejemplos de uso y código haciendo clic en este enlace.

6.4 Cargando BootStrap desde CDN

Recuerda el esqueleto de una Web, que responde a la siguiente estructura de carpetas:



Figura 14: Estructura de carpetas de una página Web

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 <!DOCTYPE html>
2 <html lang="es">
3
4   <head>
5     <meta charset="UTF-8">
6     <meta author="Alumnos de Desarrollo de Aplicaciones
7       Multiplataforma">
8     <meta name="viewport"
9       content="width=device-width, initial-scale=1.0">
10    <title>Lector de Noticias</title>
11    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
12      bootstrap@5.2.3/dist/css/bootstrap.min.css" integrity="
13      sha384-
      rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65
      " crossorigin="anonymous">
14    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
15      family=Material+Icons">
16
17    <script type="module" src="js/index.js"></script>
```

```
14     <link rel="stylesheet" href="css/estilos.css"/>
15   </head>
16   <body>
17     <div id="main" class="container">
18       <h1>Hola Mundo</h1>
19
20     </div>
21   </body>
22
23   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/
    bootstrap.bundle.min.js" integrity="sha384-
    C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL
    " crossorigin="anonymous"></script>
24
25   <script type="module" src="js/index.js"></script>
26 </html>
```

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** de arriba), cómo hemos añadido nuestro código JavaScript. Ahí, al final, cargamos un archivo JavaScript de internet (Bootstrap) y finalmente nuestro *index.js*. Este último además contendrá el código JavaScript que vamos a programar nosotros.

Habrás notado el atributo `type=module` que tiene nuestro *index.js* al final. Cargar varios archivos JavaScript desde el encabezado (`<head>`) de tu documento HTML o cargarlos como módulos desde un archivo *index.js* que los procesa tiene diferencias significativas en cuanto a rendimiento, mantenibilidad y organización de tu código. Aquí hay algunas diferencias clave:

1. Rendimiento:

- **Carga Sincrónica vs. Asincrónica:** Cuando cargas varios archivos JavaScript desde el encabezado, se descargan y ejecutan de manera sincrónica, lo que significa que uno debe cargarse y ejecutarse antes de que se inicie la carga del siguiente. Esto puede ralentizar la carga de la página, especialmente si los archivos son grandes. En cambio, cuando utilizas módulos y los importas desde un archivo *index.js*, puedes aprovechar la carga asincrónica, lo que permite que los navegadores descarguen y ejecuten los archivos de manera más eficiente, sin bloquear la carga de la página.
- **Caché y Reducción de Solicitudes:** Al utilizar módulos desde un solo archivo, es más probable que los navegadores almacenen en caché el archivo *index.js*, lo que puede reducir la necesidad de descargar archivos JavaScript cada vez que un usuario visita tu sitio. Además,

reduce la cantidad de solicitudes HTTP necesarias para cargar los recursos, lo que puede mejorar significativamente el rendimiento de la página.

2. Mantenibilidad:

- **Organización del Código:** Utilizar módulos y un archivo `index.js` centralizado puede facilitar la organización y la gestión del código. Puedes agrupar funcionalidades relacionadas en módulos individuales y mantener un archivo principal que importa y coordina estos módulos. Esto facilita la división de tu código en partes más pequeñas y manejables.
- **Reducción de Conflictos:** Al cargar múltiples archivos desde el encabezado, existe un mayor riesgo de conflictos entre variables y funciones con nombres similares. Al utilizar módulos, los nombres de las variables y funciones se aíslan dentro de los módulos y no entran en conflicto con otros elementos globales.

3. Reutilización de Código:

- **Reutilización de Módulos:** Al utilizar módulos, puedes reutilizar fácilmente funcionalidades en diferentes partes de tu aplicación. Importar un módulo en varios lugares te permite aprovechar el mismo código sin duplicarlo.

4. Debugging y Testing:

- **Depuración y Pruebas:** Utilizar módulos puede facilitar la depuración y las pruebas. Puedes aislar y probar módulos individualmente, lo que simplifica la identificación y corrección de errores.

Recuerda: cargar varios archivos JavaScript como módulos desde un archivo `index.js` suele ser una práctica más eficiente y organizada en términos de rendimiento y mantenibilidad en comparación con cargarlos desde el encabezado. Sin embargo, la elección depende de tus necesidades y del entorno de desarrollo. Si tienes una pequeña aplicación web, cargar archivos directamente en el encabezado puede ser más simple, pero a medida que el proyecto crece en complejidad, la gestión de módulos suele ser la mejor opción.

7 JavaScript

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.

7.1 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.



Figura 15: 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 intrucciones e intentará llevar a cabo la tarea que con ellas le hemos encomendado.

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 qe debe activar JavaScript o actualizar su navegador para poder ver el contenido de nuestra página:

```
1 <noscript>
```

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.

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

7.2 Sintaxis básica

7.2.1 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).

7.2.2 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).

7.2.3 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  */
```

7.2.4 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

Palabras	reservadas	en	JavaScript
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.

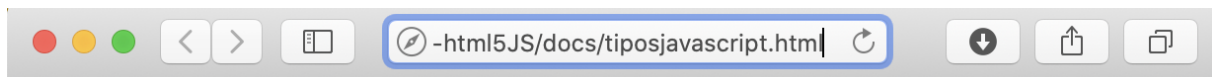
7.2.5 Tipos de datos

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   >
10  <script src="js/index.js">
11  </script>
12 </head>
13 <body>
14   <h1> Hola Mundo JS </h1>
15   <!-- Comentario HTML -->
16   <script>
17     /* Comentario JavaScript */
18     /* 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: " +
27         typeof(numero) + "</p>");
28     numero = 2.3 + 5.9;
29     document.write("<p>3) 2.3+5.9=" + numero + " que es de tipo: "
30         + typeof(numero) + "</p>");
31     numero = 2.3 + "5.9";
32     document.write("<p>4) 2.3+'5.9'=" + numero + " que es de tipo:
33         " + typeof(numero) + "</p>");
34     numero = 2.3 + +'5.9';
35     document.write("<p>5) 2.3+ +'5.9'=" + numero + " que es de tipo
36         : " + typeof(numero) + "</p>");
37     numero = 2.3 + -'5.9';
38     document.write("<p>6) 2.3+ -'5.9'=" + numero + " que es de tipo
39         : " + typeof(numero) + "</p>");
40     // VARIABLES TIPO CADENA DE CARACTERES
41     let cadena = "Hola";
42     cadena += " Mundo";
43     // se pueden anidar unas comillas dobles dentro de simples y
44     // viceversa
45     document.write('<p>7) cadena = "' + cadena + '", que es de tipo
46         : ' + typeof(cadena) + '</p>');
47     document.write("<p>8) cadena = '" + cadena + "', que es de tipo
48         : " + typeof(cadena) + "</p>");
49     document.write('<p>9) cadena = \'\' + cadena + \'\'', que es de
50         tipo: ' + typeof(cadena) + '</p>');
51     let variable = numero + cadena;
52     document.write('<p>10) variable= numero+cadena = \'\' + variable
53         + \'\'', que es de tipo: ' + typeof(variable) + '</p>');
54     // BOOLEANOS verdadero/falso
55     let booleano = true;
56     document.write("<p>11) booleano= " + booleano + ", que es de
57         tipo: " + typeof(booleano)+ '</p>');
58     booleano = "pepe";
59     document.write("<p>12) booleano= " + booleano + ", que es de
60         tipo: " + typeof(booleano)+ '</p>');
61 </script>
62 </body>
63 </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) boolano= true, que es de tipo: boolean

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

Figura 16: 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   );
25   } else
26   if (dato1 >= dato2) {
27     document.write('El primero es mayor o igual que el segundo<br>'
28   );
29   } else {
30     document.write('Ninguno de los casos<br>');
```

7.3 Estructuras de control

7.3.1 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 }
```

7.3.2 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
7     ejecutara el siguiente bloque
8     console.log('cero')
9     // NOTA: el "break" olvidado debería estar aquí
10    case 1: // No hay sentencia "break" en el 'case 0:', por lo tanto
11      este caso también será ejecutado
12      console.log('uno');
13      break; // Al encontrar un "break", no será ejecutado el 'case 2:'
14    case 2:
15      console.log('dos');
16      break;
17    default:
18      console.log('default');
19 }
```

7.3.3 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", "Sábado", "Domingo"];
2
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", "Sábado", "Domingo"];
2
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>');
```


7.3.4 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 }
```

7.3.5 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) ;
```

7.4 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.

7.5 Variables

En JavaScript, una variable es un contenedor que se utiliza para almacenar datos. Estos datos pueden ser números, cadenas de texto, objetos, funciones y más. Las variables son fundamentales en la programación, ya que te permiten almacenar y manipular información de manera dinámica en tu código.

Para declarar una variable en JavaScript, puedes utilizar tres palabras clave:

1. **var:** Antes de ECMAScript 6 (ES6), **var** era la única forma de declarar variables en JavaScript. Las variables declaradas con **var** tienen un alcance de función o global, lo que significa que están disponibles en toda la función en la que se declaran o en todo el ámbito global, dependiendo de dónde se declaren.

```
1 var nombre = "Juan";
```

2. **let:** A partir de ES6, se introdujo la palabra clave **let**. Las variables declaradas con **let** tienen un alcance de bloque, lo que significa que solo están disponibles dentro del bloque en el que se declaran. Esto hace que **let** sea útil para evitar problemas de alcance.

```
1 let edad = 25;
```

3. **const:** La palabra clave **const** se utiliza para declarar variables cuyo valor no cambiará después de su asignación inicial. Las variables **const** también tienen un alcance de bloque.

```
1 const PI = 3.1416;
```

Para asignar un valor a una variable, simplemente utiliza el operador de asignación `=`:

```
1 let nombre = "Ana";
```

Puedes utilizar variables para realizar cálculos, almacenar información, interactuar con el usuario y mucho más. También puedes cambiar el valor de una variable en cualquier momento, lo que le da a JavaScript su naturaleza dinámica. Por ejemplo:

```
1 let numero = 5;  
2 numero = numero * 2; // Cambiar el valor de la variable 'numero' a 10
```

Las variables en JavaScript son esenciales para la programación y son ampliamente utilizadas en la creación de aplicaciones web y en el desarrollo de scripts del lado del cliente.

7.6 Objetos

En JavaScript, un objeto es una estructura de datos que agrupa valores y funciones relacionados en un solo lugar. Los objetos son una parte fundamental del lenguaje y se utilizan para representar conceptos del mundo real de manera estructurada. Un objeto puede contener propiedades (también llamadas atributos) que almacenan valores y métodos (funciones) que realizan acciones asociadas a esos valores. Esto sigue el paradigma de programación orientada a objetos.

Un objeto en JavaScript se define utilizando llaves `{ }` y puede estar vacío o contener uno o más pares clave-valor. Cada clave (o nombre) se asocia a un valor, y el valor puede ser de cualquier tipo de dato, incluyendo números, cadenas, otros objetos, funciones, etc.

Aquí tienes un ejemplo simple de un objeto JavaScript:

```
1 const persona = {  
2   nombre: "Juan",  
3   edad: 30,  
4   saludar: function() {  
5     console.log("Hola, soy " + this.nombre + " y tengo " + this.edad +  
6       " años.");  
7   }  
};
```

En este ejemplo, `persona` es un objeto que tiene tres propiedades: `nombre`, `edad` y `saludar`, y `saludar` es un método. Puedes acceder a las propiedades y métodos de un objeto utilizando la notación de punto:

```
1 console.log(persona.nombre); // Accede a la propiedad 'nombre' (imprime  
   "Juan")  
2 console.log(persona.edad); // Accede a la propiedad 'edad' (imprime 30)  
3 persona.saludar(); // Llama al método 'saludar' (imprime "Hola, soy  
   Juan y tengo 30 años.")
```

Los objetos en JavaScript son muy flexibles y pueden ser anidados, lo que significa que pueden contener otros objetos como propiedades. Esto permite la creación de estructuras de datos complejas y la representación de relaciones entre datos de manera natural.

Los objetos en JavaScript y el formato JSON (JavaScript Object Notation) están estrechamente relacionados, ya que JSON se deriva de la notación de objetos en JavaScript y comparten una estructura de datos similar. Aquí tienes una descripción de su relación:

1. **Notación Similar:** La notación utilizada para definir objetos en JavaScript y para crear datos en formato JSON es muy similar. Ambos utilizan llaves `{ }` para denotar objetos, y la estructura de clave-valor es compartida. En ambos casos, se pueden anidar objetos dentro de otros objetos.
2. **JSON es un Subconjunto de Objetos JavaScript:** JSON es una forma de notación de objetos

que se originó en JavaScript, pero a lo largo del tiempo se convirtió en un formato de datos independiente y se utiliza en una variedad de lenguajes de programación. JSON es un subconjunto de la notación de objetos de JavaScript, lo que significa que todos los datos válidos en JSON son también datos válidos en JavaScript. Sin embargo, JavaScript admite más características que no son compatibles con JSON, como funciones y propiedades no enumerables.

3. **Interoperabilidad:** JSON es un formato de datos ampliamente utilizado para el intercambio de información en aplicaciones web y servicios web. Es independiente del lenguaje y es compatible con muchos lenguajes de programación. Esto facilita la comunicación entre diferentes partes de una aplicación o entre diferentes aplicaciones que utilizan tecnologías diversas.
4. **Serialización y Deserialización:** En JavaScript, puedes convertir objetos en formato JSON (serialización) y viceversa (deserialización) utilizando las funciones `JSON.stringify()` y `JSON.parse()`. La función `JSON.stringify()` convierte un objeto JavaScript en una cadena JSON, y `JSON.parse()` convierte una cadena JSON en un objeto JavaScript.

Ejemplo de serialización de un objeto a JSON:

```
1 const persona = { nombre: "Juan", edad: 30 };  
2 const personaJSON = JSON.stringify(persona);
```

Ejemplo de deserialización de JSON a objeto:

```
1 const personaDesdeJSON = JSON.parse(personaJSON);
```

5. **Uso Común:** JSON se utiliza ampliamente para transmitir datos entre un servidor y un cliente en aplicaciones web, para almacenar configuraciones y datos en archivos, y para intercambiar datos en servicios web RESTful. La simplicidad y la interoperabilidad de JSON lo hacen muy popular en el desarrollo de software.

RECUERDA: JSON es una notación de objetos basada en JavaScript que se ha convertido en un formato de datos ampliamente adoptado y que se utiliza para la comunicación y el intercambio de datos en aplicaciones web y servicios web. Su relación con los objetos de JavaScript es fundamental, ya que comparten una estructura de datos similar y se pueden convertir fácilmente entre sí.

7.7 Funciones

Las funciones en JavaScript son bloques de código que realizan una tarea específica o una serie de tareas. Las funciones son una parte fundamental del lenguaje y se utilizan para modular y organizar el código, reutilizar lógica y permitir la ejecución de acciones en respuesta a eventos o llamadas de otros fragmentos de código.

Aquí hay algunos conceptos clave relacionados con las funciones en JavaScript:

1. **Declaración de Funciones:** Puedes declarar una función en JavaScript utilizando la palabra clave `function`, seguida por un nombre de función y un conjunto de paréntesis `()`. El código de la función se coloca entre llaves `{}`. Por ejemplo:

```
1 function saludar() {  
2   console.log("Hola, mundo!");  
3 }
```

2. **Invocación de Funciones:** Para ejecutar una función, debes invocarla utilizando su nombre seguido de paréntesis. Por ejemplo:

```
1 saludar(); // Llama a la función 'saludar' y muestra "Hola, mundo!" en  
   la consola.
```

3. **Parámetros y Argumentos:** Las funciones pueden aceptar parámetros, que son valores que se pasan a la función cuando se invoca. Estos parámetros se definen entre los paréntesis en la declaración de la función. Los valores reales que se pasan a la función se denominan argumentos. Por ejemplo:

```
1 function saludar(nombre) {  
2   console.log("Hola, " + nombre + "!");  
3 }  
4  
5 saludar("Juan"); // Llama a la función 'saludar' con el argumento "Juan"  
   " y muestra "Hola, Juan!" en la consola.
```

4. **Valor de Retorno:** Las funciones pueden devolver un valor utilizando la palabra clave `return`. El valor de retorno puede ser de cualquier tipo de dato. Por ejemplo:

```
1 function sumar(a, b) {  
2   return a + b;  
3 }  
4  
5 const resultado = sumar(3, 5); // Llama a la función 'sumar' y almacena  
   el resultado en la variable 'resultado'.  
6 console.log(resultado); // Muestra 8 en la consola.
```

5. **Funciones Anónimas:** Puedes definir funciones sin un nombre y asignarlas a variables o utilizarlas directamente en contexto. Estas son conocidas como funciones anónimas. Por ejemplo:

```
1 const duplicar = function(numero) {  
2   return numero * 2;  
3 };  
4  
5 const resultado = duplicar(4); // Llama a la función anónima y almacena  
   el resultado en 'resultado'.
```

6. **Funciones Flecha:** A partir de ES6, se introdujeron las funciones flecha (`=>`) como una forma más concisa de definir funciones anónimas. Son especialmente útiles para funciones simples. Por ejemplo:

```
1 const cuadrado = (numero) => numero * numero;
```

Al igual que en cualquier lenguaje de programación, las funciones son utilizadas por tanto para encapsular lógica, dividir programas en partes más pequeñas y reutilizar código. Puedes llamar a funciones en respuesta a eventos, desde otras funciones o en cualquier parte de tu código donde necesites realizar una tarea específica.

7.8 Clases en JavaScript

Las clases en JavaScript son una característica introducida en ECMAScript 6 (ES6) que proporciona una sintaxis más orientada a objetos para definir objetos y su comportamiento. Las clases son una forma de definir plantillas para crear objetos que compartan propiedades y métodos comunes. Aunque JavaScript no es un lenguaje orientado a objetos puro como Java o C++, las clases en JavaScript simplifican la programación orientada a objetos en el lenguaje.

A continuación, se muestra cómo definir y utilizar una clase en JavaScript:

Definición de una Clase:

Para definir una clase en JavaScript, utiliza la palabra clave **class**, seguida por el nombre de la clase y el cuerpo de la clase, que contiene propiedades y métodos. Aquí tienes un ejemplo de una clase **Persona**:

```
1 class Persona {
2   constructor(nombre, edad) {
3     this.nombre = nombre;
4     this.edad = edad;
5   }
6
7   saludar() {
8     console.log(`Hola, soy ${this.nombre} y tengo ${this.edad} años.`);
9   }
10 }
```

En este ejemplo, la clase **Persona** tiene un constructor que inicializa las propiedades **nombre** y **edad**, y un método **saludar** que muestra un saludo.

Creación de Objetos a partir de una Clase:

Para crear un objeto a partir de una clase, utiliza la palabra clave **new** seguida del nombre de la clase. Luego puedes llamar a los métodos y acceder a las propiedades del objeto resultante. Aquí tienes un

ejemplo:

```
1 const juan = new Persona("Juan", 30);
2 juan.saludar(); // Muestra "Hola, soy Juan y tengo 30 años."
```

Herencia de Clases:

JavaScript también admite la herencia de clases. Puedes crear una clase hija que herede propiedades y métodos de una clase padre utilizando la palabra clave **extends**. Aquí hay un ejemplo:

```
1 class Estudiante extends Persona {
2   constructor(nombre, edad, curso) {
3     super(nombre, edad);
4     this.curso = curso;
5   }
6
7   presentarse() {
8     console.log(`Hola, soy ${this.nombre}, tengo ${this.edad} años y
9       estudio ${this.curso}.`);
10  }
```

En este ejemplo, la clase **Estudiante** hereda de la clase **Persona**. El método **super()** se utiliza para llamar al constructor de la clase padre y luego se agregan propiedades adicionales y métodos específicos de la clase hija.

Métodos Estáticos:

Puedes definir métodos estáticos en una clase que son accesibles directamente desde la clase, en lugar de desde instancias de la clase. Para hacerlo, utiliza la palabra clave **static**. Por ejemplo:

```
1 class Utilidades {
2   static duplicar(numero) {
3     return numero * 2;
4   }
5 }
6
7 const resultado = Utilidades.duplicar(5); // Llama al método estático
   sin crear una instancia.
```

En resumen, las clases en JavaScript permiten crear plantillas para objetos, simplifican la programación orientada a objetos y fomentan la reutilización de código. Puedes definir propiedades y métodos en una clase, crear objetos a partir de ella y heredar propiedades y métodos en clases hijas. Las clases son una característica importante de ES6 y son ampliamente utilizadas en el desarrollo de aplicaciones web y otras aplicaciones en JavaScript.

7.9 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 reassignable y *no* permite hoisting; **const** declara una variable de ámbito global, local para la función o de bloque; no es reassignable, 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 creamos 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*.

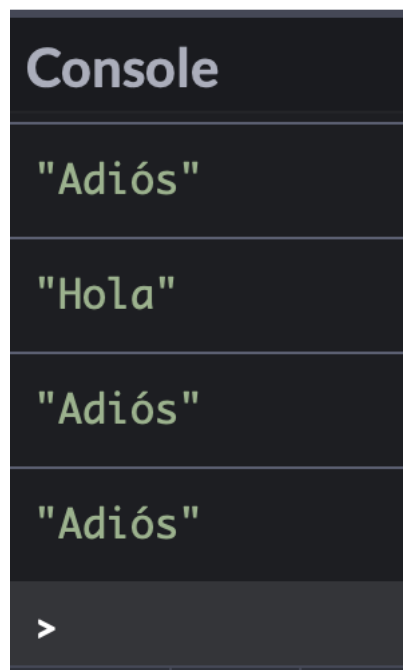


Figura 17: Ámbito de una variable con let y var: diferencias

En JavaScript, las declaraciones (de variables o funciones) se mueven al principio de su ámbito. Este comportamiento se conoce como *hoisting* o *leantamiento* 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”.

7.9.1 Arrays (arreglos) y Objetos (diccionarios)

JavaScript no tiene un tipo de dato específico llamado “lista”. En lugar de eso, JavaScript utiliza principalmente dos estructuras de datos para almacenar colecciones de elementos: arrays (arreglos) y

objetos. Ambos tienen sus propias características y se utilizan de manera diferente.

1. **Arrays (Arreglos):** Los arrays en JavaScript son colecciones ordenadas de elementos. Los elementos de un array pueden ser de diferentes tipos, como números, cadenas, objetos u otros arrays. Los elementos en un array se almacenan en un índice numérico y se pueden acceder y modificar mediante ese índice. Aquí hay un ejemplo de cómo crear y trabajar con un array en JavaScript:

```
1 const miArray = [1, 2, 3, "cuatro", { nombre: "Juan" }];
2 console.log(miArray[0]); // Accede al primer elemento del array (1)
3 miArray.push(5); // Agrega un elemento al final del array
4 miArray.length; // Devuelve la longitud del array (5)
```

2. **Objetos:** Los objetos en JavaScript son colecciones no ordenadas de pares clave-valor. Cada clave (nombre de propiedad) se asigna a un valor. Los valores pueden ser de cualquier tipo, incluyendo números, cadenas, funciones y otros objetos. Aquí tienes un ejemplo de cómo crear y trabajar con un objeto en JavaScript:

```
1 const miObjeto = {
2   nombre: "Juan",
3   edad: 30,
4   ciudad: "Ejemploville"
5 };
6 console.log(miObjeto.nombre); // Accede a la propiedad 'nombre' del
  objeto ("Juan")
7 miObjeto.profesion = "Desarrollador"; // Agrega una nueva propiedad al
  objeto
8 delete miObjeto.edad; // Elimina la propiedad 'edad' del objeto
```

En resumen, en JavaScript, no existe una estructura de datos específica llamada “lista”. En su lugar, puedes usar arrays para crear colecciones ordenadas de elementos o objetos para crear colecciones de pares clave-valor. La elección entre utilizar arrays u objetos depende de la naturaleza de los datos que estás manejando y de cómo planeas acceder y manipular esos datos en tu código. Los arrays son ideales para secuencias ordenadas de elementos, mientras que los objetos son útiles para representar datos con propiedades descriptivas.

En JavaScript, el término “**diccionario**” se utiliza coloquialmente para referirse a objetos. En JavaScript, los objetos se asemejan a los diccionarios en otros lenguajes de programación, ya que son colecciones no ordenadas de pares clave-valor. Cada propiedad de un objeto actúa como una clave que se asigna a un valor asociado.

Aquí hay otro ejemplo de un “diccionario” en JavaScript utilizando un objeto:

```
1 const persona = {
2   nombre: "Juan",
```

```
3   edad: 30,  
4   ciudad: "Ejemploville"  
5 };
```

En este caso, `persona` es un objeto que actúa como un diccionario. Las propiedades (`nombre`, `edad`, `ciudad`) son las claves, y los valores asociados a esas claves son los datos correspondientes. Puedes acceder a los valores utilizando las claves, por ejemplo:

```
1 console.log(persona.nombre); // Accede al valor de la clave 'nombre' ("Juan")  
2 console.log(persona.edad); // Accede al valor de la clave 'edad' (30)
```

Puedes agregar, modificar y eliminar propiedades en un objeto como lo harías en un diccionario. Por ejemplo:

```
1 persona.profesion = "Desarrollador"; // Agrega una nueva propiedad  
2 persona.edad = 31; // Modifica el valor de una propiedad  
3 delete persona.ciudad; // Elimina una propiedad
```

Además, puedes utilizar bucles `for...in` para recorrer todas las propiedades de un objeto y realizar operaciones en cada una de ellas.

Sabiendo esto, ahora no te extrañará que se pueda acceder a las propiedades de un objeto en JavaScript de dos formas: utilizando la notación de punto (`objeto.propiedad`) o utilizando la notación de corchetes (`objeto['propiedad']`). Ambas formas son equivalentes y te permiten acceder a las propiedades de un objeto. Aquí tienes el ejemplo anterior ampliado para mostrar cómo acceder a la propiedad “nombre” utilizando ambas notaciones:

```
1 const persona = {  
2   nombre: "Juan",  
3   edad: 30,  
4   ciudad: "Ejemploville"  
5 };  
6  
7 // Acceso a propiedades con notación de punto  
8 console.log(persona.nombre); // Accede al valor de la propiedad 'nombre'  
9   ' ("Juan")  
9 console.log(persona.edad); // Accede al valor de la propiedad 'edad'  
10   (30)  
10  
11 // Acceso a propiedades con notación de corchetes  
12 console.log(persona['nombre']); // Accede al valor de la propiedad '  
13   nombre' ("Juan")  
13 console.log(persona['edad']); // Accede al valor de la propiedad 'edad'  
14   (30)
```

Ambas notaciones son intercambiables, y puedes usar la que te resulte más cómoda o adecuada en un contexto particular. La notación de corchetes es especialmente útil cuando necesitas acceder a

propiedades utilizando variables o cuando las claves contienen caracteres especiales o espacios. Por ejemplo:

```
1 const propiedad = 'nombre';
2 console.log(persona[propiedad]); // Accede al valor de la propiedad '
  nombre' usando una variable
3
4 const claveConEspacios = 'nombre completo';
5 console.log(persona[claveConEspacios]); // Accede al valor de una
  propiedad con espacios en su nombre
```

La notación de corchetes también es útil para acceder a propiedades dinámicas o para manipular objetos de manera más dinámica en tu código.

Como hemos dicho antes, se puede conocer las propiedades de un objeto en JavaScript utilizando varios métodos y técnicas. Ejemplo:

1. **Método `Object.keys()`**: El método `Object.keys()` te permite obtener un array de las claves (propiedades) de un objeto. Puedes luego recorrer este array para conocer todas las propiedades. Aquí tienes un ejemplo:

```
1 const objeto = { nombre: "Juan", edad: 30, ciudad: "Ejemploville" };
2 const propiedades = Object.keys(objeto);
3
4 for (let i = 0; i < propiedades.length; i++) {
5   console.log(propiedades[i]); // Muestra las propiedades una por una
6 }
```

2. **Bucle `for...in`**: El bucle `for...in` te permite recorrer todas las propiedades enumerables de un objeto. Puedes utilizarlo para acceder a las claves y sus valores. Aquí hay un ejemplo:

```
1 const objeto = { nombre: "Juan", edad: 30, ciudad: "Ejemploville" };
2
3 for (const propiedad in objeto) {
4   if (objeto.hasOwnProperty(propiedad)) {
5     console.log(propiedad + ": " + objeto[propiedad]);
6   }
7 }
```

4. **Método `Object.getOwnPropertyNames()`**: El método `Object.getOwnPropertyNames()` devuelve un array de todas las propiedades (enumerables y no enumerables) de un objeto. Esto te proporcionará una lista completa de las propiedades del objeto.

```
1 const objeto = { nombre: "Juan", edad: 30, ciudad: "Ejemploville" };
2 const propiedades = Object.getOwnPropertyNames(objeto);
3
4 for (let i = 0; i < propiedades.length; i++) {
```

```
5 console.log(propiedades[i]); // Muestra las propiedades una por una
6 }
```

Ten en cuenta que en el ejemplo anterior se verifica si una propiedad existe en el objeto utilizando el método `hasOwnProperty()` para asegurarte de que estás trabajando solo con las propiedades propias del objeto, no con propiedades heredadas de prototipos.

Cada uno de estos métodos tiene sus ventajas y desventajas, y la elección de cuál utilizar depende de tus necesidades específicas. En la mayoría de los casos, `Object.keys()` o el bucle `for...in` son las formas más comunes de conocer las propiedades de un objeto.