alura
LATAM

# DESARROLLO WEB CON HTML, CSS Y JAVASCRIPT

# Sumário

Versão: 26.0.17

Capítulo 1

# SOBRE EL E-BOOK - EL COMPLEJO MUNDO DEL DESARROLLO WEB

*"Acción es la clave fundamental para todo éxito" -- Pablo Picasso*

Vivimos hoy en una era en que el Internet ocupa un espacio cada vez más importante en nuestras vidas personales y profesionales. El surgimiento constante de Aplicaciones Web, para las más diversas finalidades, es una señal clara de que ese mercado está en franca expansión y trae muchas oportunidades. Aplicaciones corporativas, comercio electrónico, redes sociales, películas, músicas, noticias y tantas otras áreas están presentes en el Internet, haciendo del navegador (el *browser*) el software más utilizado de nuestros computadores.

Este e-book pretende abordar el proceso de desarrollo de páginas que accedemos por medio del navegador de nuestros computadores, utilizando patrones actuales de desarrollo conociendo en profundidad sus características técnicas. Discutiremos la implantación de esas tecnologías en los diferentes navegadores, la adopción de *frameworks* que facilitan y aceleran nuestro trabajo, a parte de consejos técnicos que destacan a un profesional en el mercado. HTML y CSS serán vistos en profundidad a parte de eventos con JavaScript.

Aparte del acceso por medio del navegador de nuestros computadores, hoy el acceso a Internet a partir de dispositivos móviles representa un gran avance de la plataforma, pero también implica un poco más de atención en el trabajo de quien va a desarrollar. En el recorrer del curso, vamos conocer algunas de esas necesidades y cómo utilizar los recursos disponibles para atender también a esa gran necesidad.

## 1.1 EL CURSO

El contenido comienza con fundamentos de HTML y CSS, incluyendo tópicos relacionados con las novedades de las versiones HTML5 y CSS3, como por ejemplo, Flexbox y Grid. Después, es abordado el lenguaje de programación JavaScript enfocado en la parte de eventos, para crear interacciones entre el usuario y la página.

Te recomendamos que pongas en practica todo lo que aprendas, ni bien termines cada capitulo. Un buen ejercicio puede ser elegir un sitio web que te guste e intentar reproducirlo (comienza con uno simple y aumenta la complejidad con la práctica).

## 1.2 ACLARANDO LAS DUDAS

Recomendamos fuertemente buscar recursos y participación activa en la comunidad por medio de las listas de discusión relacionadas al contenido del curso.

Algunas referencias donde puedes buscar informaciones:

- MDN Web Docs - un sitio con la documentación oficial para desarrollo web de Mozilla (contenido en español)
- W3Schools - plataforma con tutoriales de los lenguajes y frameworks usados en el desarrollo web (contenido en ingles)

Alura también es un excelente lugar para aprender y aclarar tus dudas. Ella es una plataforma de cursos online que cuenta con diversos cursos dirigidos tanto para tecnología como para otras áreas. Posee una comunidad activa en los foros y nuestro equipo está siempre dispuesto a ayudar a todos los alumnos y alunas.

https://aluracursos.com

## 1.3 BIBLIOGRAFÍA

Aparte del conocimiento disponible en el Internet, existen muchos libros interesantes sobre el asunto. Algunas referencias:

- **HTML5 y CSS3: Domine la web del futuro (Portugués)** - Lucas Mazza, editora Casa del Código.
- **La Web Mobile: Design Responsivo y la parte para una Web adaptada al mundo moble (Portugués)** - Sérgio Lopes, editora Casa del Código.
- **The Art and Science of CSS** - Adams & Cols.
- **Pro JavaScript Techniques** - John Resig.
- **The Smashing Book** - smashingmagazine.com

## 1.4 ¿PARA DÓNDE IR DESPUÉS?

Este e-book es un complemento de la **Formación Front-end** de Alura que engloba cursos de HTML, CSS y JavaScript. Puedes obtener más informaciones aquí:

https://www.aluracursos.com/cursos-online-front-end

Si tu deseo es entrar más fondo en el desarrollo Web, incluyendo la parte *server-side*, ofrecemos cursos de Desarrollo con **Java**.

Más informaciones en:

https://www.aluracursos.com/cursos-online-programacao/java

Capítulo 2

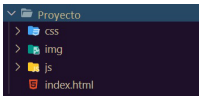
# LA ESTRUCTURA DE LOS ARCHIVOS DE UN PROYECTO

Como todo tipo de proyecto de software, existen algunas recomendaciones cuanto a la organización de los archivos de un sitio. No hay ningún rigor técnico cuanto a esa organización y, en la mayoría de las veces, puedes adaptar las recomendaciones de la forma que sea mejor para tu proyecto.

Como un sitio es un conjunto de páginas Web sobre un asunto, empresa, producto o cualquier otra cosa, es común que todos los archivos de un sitio estén dentro de una sola carpeta y, así como un libro, es recomendado que exista una "capa", una página inicial que pueda indicar para el visitante cuáles son las otras páginas que hacen parte de ese proyecto y como él puede acceder, como si fuese el **índice** del sitio.

Que haya ese índice, no es por coincidencia, es una convención adoptada por los servidores de páginas Web. Si deseamos que una determinada carpeta sea exhibida como una página Web y dentro de esa carpeta existe un archivo llamado **index.html**, ese archivo será la página inicial, o sea el índice, al menos que alguna configuración determine otra página para esa finalidad.

Dentro de la carpeta del proyecto, en el mismo nivel que el `index.html` , es recomendado que sean creadas otras carpetas para mantener separados los archivos de imágenes, las hojas de estilo y los scripts. Para iniciar un proyecto, tendríamos una estructura de carpetas como la demostrada en la siguiente imagen:




Muchas veces, un sitio Web es hospedado por medio de otra aplicación Web y, en esos casos, la estructura de los archivos depende de cómo la aplicación necesita de los recursos para funcionar correctamente. Pero, en general, las aplicaciones también siguen un patrón bien parecido al que estamos adoptando para nuestro proyecto.

## 2.1 SITIO WEB O APLICACIÓN WEB?

Cuando estamos comenzando en el mundo del desarrollo Web, acabamos por conocer muchos

términos nuevos, que por muchas veces no son claros o nos causan confusión. Vamos a entender un poco más ahora, cual es la diferencia entre un sitio Web y una aplicación Web.

### Sitio Web

Podemos considerar que un sitio Web es una colección de páginas HTML estáticas, o sea, que no interactúan con un banco de datos a través de un lenguaje de servidor Web. O sea, aquí todo el contenido del sitio está escrito directamente en el documento HTML, así como las imágenes y otras medias. Claro que, para que cualquier página Web sea ofrecida públicamente, la misma debe estar hospedada en un simple servidor Web (hospedaje de sitios).

### Aplicación Web

Una aplicación Web puede contener una colección de páginas, pero el contenido de estas páginas es montado en forma dinámica, o sea, es cargado a través de solicitaciones (petición) al banco de datos, que contendrá almacenado los textos y indicación de los caminos de las imágenes o recursos que la página necesita exhibir. Sin embargo, HTML no tiene acceso directo a un banco de datos, y esta comunicación debe ser hecha por un lenguaje de programación de servidor Web. Esta aplicación está escrita con un lenguaje de servidor que tiene el poder de acceder al banco de datos y montar la página HTML conforme lo solicitado por el navegador. Estas solicitudes pueden ser hechas de varias formas, inclusive utilizando JavaScript. Por lo tanto, una aplicación Web es más compleja porque necesita de un lenguaje de servidor para poder intermediar las solicitudes del navegador, un banco de datos, y muchas veces (pero no obligatoriamente) exhibir páginas HTML con estos contenidos.

Ejemplo de lenguajes de servidor Web: Java EE, PHP, Python, Ruby on Rails, NodeJS, etc.

## 2.2 EDITORES Y IDES

Los editores de texto son programas de computador livianos e ideales para escribir y editar las páginas de un sitio, como *Visual Studio Code* (https://code.visualstudio.com/), *Sublime* (https://www.sublimetext.com/), *Atom* (https://atom.io/) y *Notepad++* (https://notepad-plus-plus.org), que poseen realce de sintaxis y otras herramientas para facilitar el desarrollo de páginas.

Hay también **IDEs** (*Integrated Development Environment*) que son editores más robustos y tienen más facilidades para el desarrollo de aplicaciones Web, integrándose con otras funcionalidades. Algunos de ellos son: *WebStorm* (https://www.jetbrains.com/webstorm/) *Eclipse* (https://www.eclipse.org/) y *Visual Studio* (https://visualstudio.microsoft.com).

CAPÍTULO 3

# INTRODUCCIÓN AL HTML

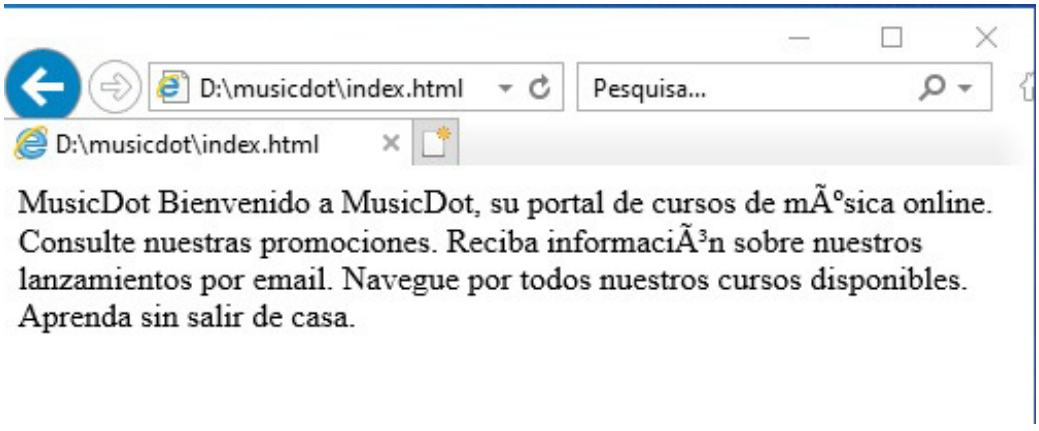
*"Cuanto más nos elevamos, menores parecemos a los ojos de aquellos que no saben volar." -- Friedrich Wilhelm Nietzsche*

## 3.1 EXHIBIENDO INFORMACIONES EN LA WEB

El único lenguaje que un navegador Web consigue interpretar para la exhibición de contenido es HTML. Para iniciar la exploración de HTML, vamos a imaginar el siguiente caso: el navegador realizó un pedido y recibió como cuerpo de la respuesta el siguiente contenido:

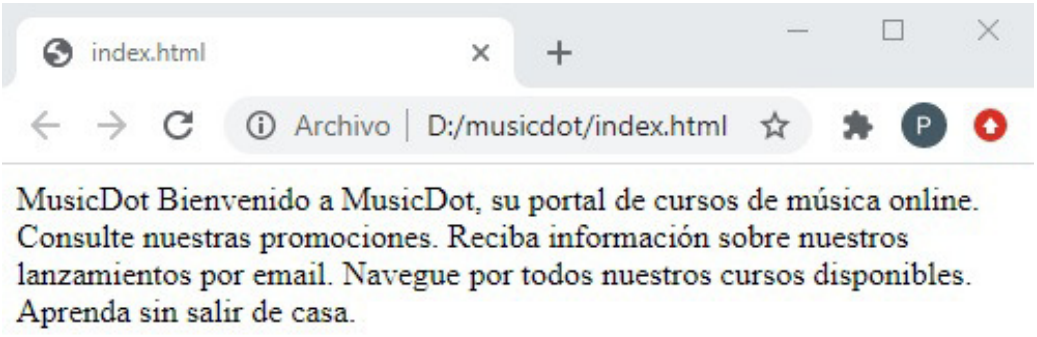
```
MusicDot

Bienvenido a MusicDot, su portal de cursos de música online.

Consulte nuestras promociones.
Reciba información sobre nuestros lanzamientos por email.
Navegue por todos nuestros cursos disponibles.
Aprenda sin salir de casa.
```

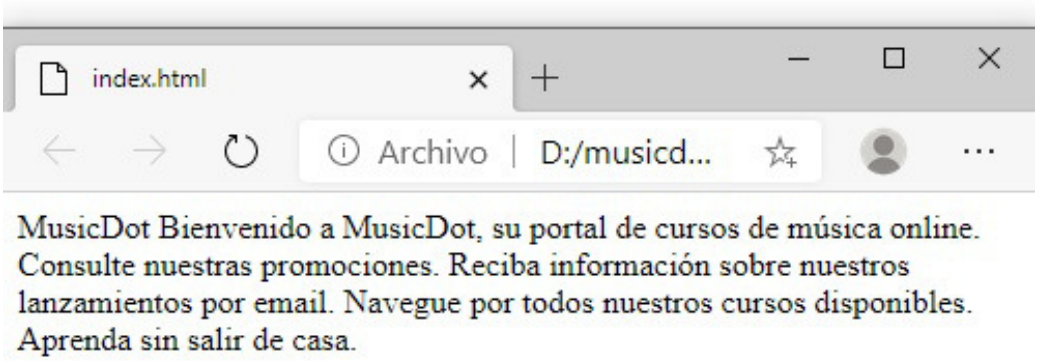
Para conocer el comportamiento de los navegadores en cuanto al contenido descrito antes, vamos a reproducir este contenido en un archivo de texto común, que puede ser creado con cualquier editor de texto puro. Guarde el archivo como **index.html** y ábralo en el navegador de su preferencia.




Parece que obtuvimos un resultado un poco diferente del esperado, ¿no? A pesar de ser capaz de exhibir texto puro en su área principal, algunas reglas deben ser seguidas caso deseemos que ese texto sea exhibido con algún formato, principalmente para facilitar la lectura por el usuario final.

Una nota de atención es que la imagen de arriba fue tomada del navegador: **Microsoft Internet Explorer 11**. Vea lo que pasa cuando obtenemos la misma imagen pero con navegadores más actuales:




La imagen de arriba fue tomada de los navegadores: **Google Chrome y Microsoft Edge**.

*Obs: existe la posibilidad de que mismo en esos navegadores, si utilizamos una versión más antigua, puede ser que el texto sea mostrado igual que el de la foto del navegador Internet Explorer*.

Usando los resultados de arriba podemos concluir que los navegadores más antiguos por defecto:

- Pueden no exhibir correctamente caracteres con acentuaciones;

Pero hasta en los navegadores más nuevos:

- No se exhiben los saltos de fila.

Para que podamos exhibir las informaciones con el formateo deseado, es necesario que cada trecho de texto tenga una **marcación** indicando cuál es su significado. Esa marcación también influye en la forma con que cada trecho del texto será exhibido. A continuación es listado el texto con esta marcación esperada por el navegador:

```
<!DOCTYPE html>
<html>
  <head>
    <title>MusicDot</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>MusicDot</h1>
    <h2>Bienvenido a MusicDot, su portal de cursos de música online.</h2>
    <ul>
      <li>Consulte nuestras promociones.</li>
      <li>Reciba información sobre nuestros lanzamientos por email.</li>
      <li>Navegue por todos nuestros cursos disponibles.</li>
      <li>Aprenda sin salir de casa.</li>
    </ul>
  </body>
</html>
```

El texto con las debidas marcaciones, comúnmente es llamado de "código". Reproduzca entonces el código anterior en un nuevo archivo de texto puro y guárdalo como **index2.html**. No se preocupe con la sintaxis, vamos a conocer detalladamente cada característica de estas marcaciones en los próximos capítulos. Abra el archivo en los tres navegadores diferentes.


Ahora, una página mucho más agradable y legible es exhibida. Para eso, tuvimos que añadir las marcaciones que son pertenecientes a HTML. Esas marcaciones son llamadas de **tags o etiquetas**, y ellas básicamente dan una **representación** al texto, contenido entre su apertura y cierre.

A pesar de estar correctamente marcadas, las informaciones presentan poco o ningún atractivo estético y, en esa deficiencia del HTML, reside el primer y mayor desafío para las personas que trabajan con desarrollo Web.

HTML (*Hypertext Markup Language*) o lenguaje de marcación de hipertexto fue desarrollado para suplir la necesidad de exhibición de documentos científicos disponibles por una red de Internet. Pero, con el tiempo y la evolución de la Web y de su potencial comercial, fue necesaria la exhibición de información con gran riqueza de elementos gráficos y de interacción.

Comenzaremos por partes, primero entenderemos como funciona HTML, para después aprender estilos, elementos gráficos e interacciones.

## 3.2 SINTAXIS DEL HTML

HTML es un conjunto de **etiquetas** responsables por la marcación del contenido de una página en el navegador. En el código que vimos antes, las etiquetas son los elementos extras que escribimos usando la sintaxis `<nombredelaetiqueta>`. Diversas etiquetas están disponibles para el lenguaje HTML y cada una posee una funcionalidad específica.

En el código de antes, vimos por ejemplo el uso del etiqueta `<h1>`. Esta representa el título principal de la página.

```
<h1>MusicDot</h1>
```

Note la sintaxis. Una etiqueta es definida con caracteres `<` y `>`, y su nombre (**h1** en el caso). Muchas etiquetas poseen contenido, como el texto del título (*"MusicDot"*). En este caso, para determinar donde acaba el contenido, usamos una *etiqueta de cierre* con la barra antes del nombre: `</h1>`.

Algunas etiquetas pueden recibir algún tipo de información extra dentro de su definición llamadas de **atributos**. Son parámetros que usan la sintaxis de `atributo="valor"`. Para definir una imagen, por ejemplo, usamos el etiqueta `<img>` y, para indicar la ruta de la imagen, usamos el atributo `src`:

```
<img src="../imagenes/casa_de_playa.jpg">
```

Observe que la etiqueta `img` no posee contenido por sí sola, lo que contiene es un archivo externo (la imagen). En estos casos, **no** es necesario usar una etiqueta de cierre como antes en el `h1`.

## 3.3 ETIQUETAS HTML

El HTML está compuesto de diversas etiquetas, cada una con su función y significado. Desde 2013, con la actualización del lenguaje para el HTML 5, muchos nuevas etiquetas fueron agregadas, que veremos a lo largo del curso.

En este momento, nos vamos a enfocar en etiquetas que representan **títulos**, **párrafos** y **énfasis**.

### Títulos

Cuando queremos indicar que un texto es un título en nuestra página, utilizamos los etiquetas de ***heading*** en su marcación:

```
<h1>MusicDot</h1>
<h2>Bienvenido a MusicDot, su portal de cursos de música online.</h2>
```

Las etiquetas de *heading* son para exhibir contenido de texto y contiene 6 niveles, o sea de `<h1>` a

`<h6>` , siguiendo un orden de importancia, siendo `<h1>` el título principal, el más importante, y `<h6>` el título de menor importancia.

Utilizamos, por ejemplo, la etiqueta `<h1>` para el nombre, título principal de la página, y la etiqueta `<h2>` como subtitulo o como título de secciones dentro del documento.

*Obs: la etiqueta `<h1>` solo puede ser utilizada una vez en cada página porque es utilizado para marcar el contenido más importante de la página.*

El orden de importancia tiene impacto en las herramientas que procesan HTML. Las herramientas de indexación de contenido para búsquedas, como Google, Bing o Yahoo llevan en consideración ese orden y relevancia. Los navegadores especiales para accesibilidad también interpretan el contenido de esas etiquetas de forma a diferenciar su contenido y facilitar la navegación del usuario por el documento.

## Párrafos

Cuando exhibimos cualquier texto en nuestra página, es recomendado que este sea siempre contenido de alguna etiqueta hija de la etiqueta `<body>` . La marcación más indicada para textos comunes es la etiqueta de **párrafos**:

```
<p>
  MusicDot es la mayor escuela online de música en todo el mundo.
</p>
```

Si tenemos varios párrafos de texto, usamos varias de esas etiquetas `<p>` para separarlos:

```
<p>
  MusicDot es la mayor escuela online de música en todo el mundo.
</p>
<p>
  Nuestra matriz queda en Mafra, en Santa Catarina. De allá, salen  gran parte de las grabaciones de nuestros cursos.
</p>
```

## Marcaciones de énfasis

Cuando queremos dar énfasis diferente a un trecho de texto, podemos utilizar las marcaciones de énfasis. Podemos dejar un texto "más fuerte o con negrita" con la etiqueta `<strong>` o dejar el texto con un "énfasis acentuada o itálico" con la etiqueta `<em>` . De la misma forma que la etiqueta `<strong>` deja la etiqueta "más fuerte", tenemos también el etiqueta `<small>` , que disminuye el "peso" del texto.

Por defecto, los navegadores exhiben el texto dentro del etiqueta `<strong>` en negrita y el texto dentro del etiqueta `<em>` en itálica. Existen aún las etiquetas `<b>` y `<i>` , que alcanzan el mismo resultado visualmente, pero los etiquetas `<strong>` y `<em>` son más indicadas por definir nuestra intención de significado al contenido, más que una simple indicación visual. Vamos a discutir mejor la

cuestión del significado de las etiquetas más adelante.

```
<p>Aprenda de una forma rápida y barata en <strong>MusicDot</strong>.</p>
```

## 3.4 IMÁGENES

La etiqueta `<img>` indica para el navegador que una imagen debe ser "renderizada" (mostrada/diseñada) en aquel lugar y necesita dos atributos: `src` y `alt`. El primero es un atributo obligatorio para exhibir la imagen y apunta para su localización (puede ser un local de su computador o una dirección en la Web), y el segundo es un texto alternativo que aparece caso la imagen no pueda ser cargada o visualizada.

El atributo `alt` no es obligatorio, pero es considerado un error caso sea omitido, pues este provee el entendimiento de la imagen para personas con deficiencia que necesitan el uso de lectores de pantalla para acceder al computador, y también ayuda en la indexación de la imagen para motores de busca, como Google.

HTML 5 introdujo dos nuevas etiquetas específicas para imagen: `<figure>` y `<figcaption>`. La etiqueta `<figure>` define una imagen en conjunto con la etiqueta `<img>`. A parte de eso, permite añadir una leyenda para la imagen por medio de la etiqueta `<figcaption>`.

```
<figure>
  <img src="img/matriz-musicdot.png" alt="Foto de la matriz de musicdot">
  <figcaption>Matriz de MusicDot</figcaption>
</figure>
```

## 3.5 PRIMERA PÁGINA

La primera página que desarrollaremos para *MusicDot* será la página *Sobre*, que explica detalles sobre la empresa, presenta fotos y la historia.

Recibimos el diseño listo, así como los textos. Nuestro trabajo, como personas desarrolladoras de front-end, es codificar el HTML y CSS necesarios para ese resultado.

**BUENA PRÁCTICA - INDENTACIÓN**

Una práctica siempre recomendada, asociada a la organización y utilizada para facilitar la lectura del código, es el uso correcto de **sangría**, o **indentación**. En HTML acostumbramos alinear los elementos "hermanos" en el mismo margen y añadir algunos espacios o un *tab* para elementos "hijos".

La mayoría de los ejercicios de este tutorial utiliza un patrón recomendado de indentación.

**BUENAS PRÁCTICAS - COMENTARIOS**

Cuando comenzamos nuestro proyecto, utilizamos pocas etiquetas HTML. En seguida agregaremos una cantidad razonable de elementos, lo que puede generar una cierta confusión. Para mantener el código más legible, es recomendado agregar comentarios antes de la apertura y después del cierre de las etiquetas estructurales (que contendrán otras etiquetas). De esa forma, nosotros podemos identificar claramente cuando un elemento está **dentro** de esa estructura o **después** de ella.

```
<!-- inicio del encabezado-->
<header>
  <p>Este párrafos está <strong>dentro</strong> del encabezado.</p>
</header>
<!-- fin del encabezado-->

<p>Este párrafo está <strong>después</strong> del encabezado.</p>
```

Capítulo 4

# ESTRUCTURA DE UN DOCUMENTO HTML

Un documento HTML válido necesita seguir obligatoriamente la estructura compuesta por las etiquetas (o tags) `<html>`, `<head>` y `<body>` y la instrucción `<!DOCTYPE>`. Esta estructura está informada en una documentación que describe todos los detalles del HTML, en el caso las etiquetas y los atributos, y como los navegadores deben considerar y interpretar estas etiquetas. Esa documentación es llamada de "especificación del HTML", y a través del cual es posible entender si un documento HTML es válido o no. Cuando un documento HTML es inválido, este es cargado por el navegador, pero en un "*modo de compatibilidad*",vamos entender mejor sobre eso más adelante.

A seguir, vamos a conocer en detalles cada un de las **tags** o etiquetas estructurales obligatorias:

## 4.1 LA ETIQUETA <HTML>

En la estructura de nuestro documento, antes de comenzar a colocar el contenido, insertamos una etiqueta `<html>`. Dentro de esa etiqueta, es necesario declarar otras dos tags: `<head>` y `<body>`. Esas dos tags son "hermanas", pues están en el mismo nivel jerárquico en relación a su tag "madre", que es el `<html>`.

```
<html> <!-- madre-->
  <head></head> <!-- hijo-->
  <body></body> <!-- hijo-->
</html>
```

## 4.2 LA ETIQUETA <HEAD>

La etiqueta `<head>` contiene informaciones sobre el documento HTML que son de interés solamente del navegador y para otros servicios de la web, y no para las personas que van a acceder a nuestro sitio. Son informaciones que no serán exhibidas directamente en el navegador, también podemos considerar un local donde informamos los metadatos sobre la página.

La especificación del HTML obliga la presencia de la etiqueta de contenido `<title>` dentro del `<head>`, permitiendo definir el título del documento, que puede ser visto en la *barra de título* o *pestaña* de la ventana del navegador. Caso contrario, la página no será un documento HTML válido.

Otra configuración muy importante, principalmente en documentos HTML cuyo contenido es escrito en un idioma como el español, que contienen caracteres "especiales" (acentos y tildes), es la codificación/conjunto de caracteres, llamadas de **encoding** o **charset**.

Podemos configurar cual codificación queremos utilizar en nuestro documento por medio de la configuración de `charset` en la tag `<meta>`. Uno de los valores más comunes usados hoy en dia es el **UTF-8**, también llamado de **Unicode**. Hay otras posibilidades, como el **latin1**, muy usado antiguamente.

El **UTF-8** es la recomendación actual para encoding en la Web por ser ampliamente soportada en navegadores y editores de código, aparte de ser compatible con prácticamente la mayoria de los idiomas del mundo. Es el que usaremos en el curso.

```
<html>
    <head>
      <meta charset="utf-8">
      <title>MusicDot</title>
    </head>
    <body>

    </body>
</html>
```

## 4.3 LA ETIQUETA <BODY>

La tag `<body>` contiene el cuerpo de un documento HTML, que es exhibido por el navegador en su ventana, o sea, todo el contenido visible del sitio. Es necesario que el `<body>` tenga al menos un elemento "hijo", o sea, una o más etiquetas HTML dentro de él.

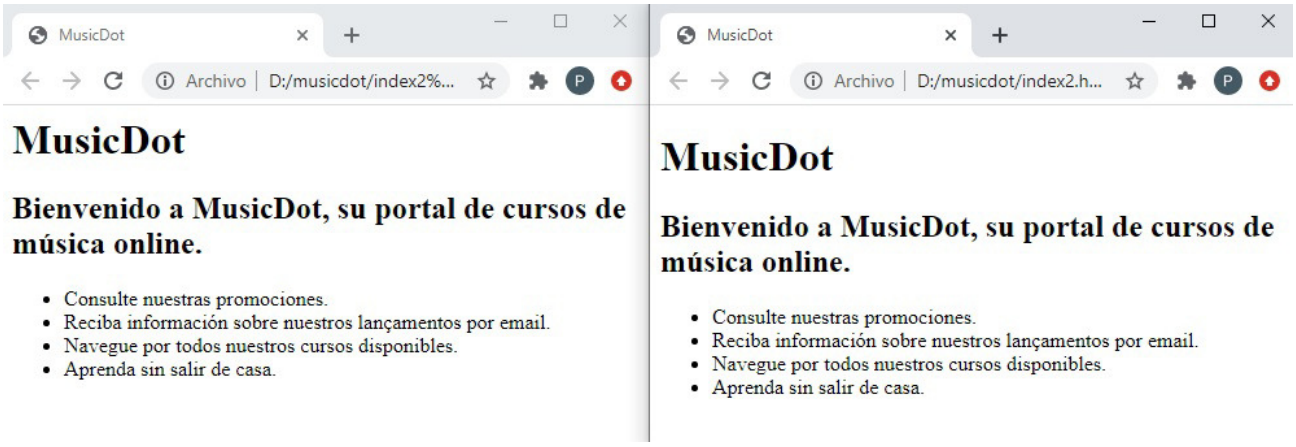
```
<html>
  <head>
    <meta charset="utf-8">
    <title>MusicDot</title>
  </head>
  <body>
    <h1>MusicDot</h1>
  </body>
</html>
```

En este ejemplo, usamos la etiqueta `<h1>`, que indica el título principal de la página.

## 4.4 LA INSTRUCCIÓN DOCTYPE

El `DOCTYPE` no es un etiqueta HTML, sino una instrucción especial. Ella indica para el navegador cual **versión de HTML** debe ser utilizada para exhibir la página. Cuando no colocamos esa instrucción la página es exhibida en una especie de "*modo de compatibilidad*" en la cual algunas etiquetas y estilizaciones no funcionan 100% correctamente. Principalmente las etiquetas y estilizaciones más actuales (lanzadas en la versión 5 del HTML). Inclusive es posible ver la diferencia en la hoja de estilos

patrón que el navegador usa cuando no colocamos esa instrucción.




La imagen de la izquierda es la página **sin** `Doctype` y la imagen de la derecha es la página **con** `Doctype` . Dá para ver que existe una pequeña diferencia entre las dos páginas, principalmente con relación a los espaciados.

Utilizaremos `<!DOCTYPE html>` , que indica para el navegador la utilización de la versión más reciente del HTML - la versión 5, actualmente*.

Hay muchas posibilidades más complicadas en esa parte de `DOCTYPE` que eran usadas en versiones anteriores del HTML y del XHTML. Hoy en día, nada de eso es relevante. Lo recomendado es **siempre usar la última versión de HTML**, usando la declaración de `DOCTYPE` simple:

```
<!DOCTYPE html>
```

La declaración del DOCTYPE, puede ser escrita toda en mayúscula o toda en minúscula o con la primera letra mayúscula: `<!DOCTYPE HTML>` , `<!DOCTYPE html>` , `<!Doctype HTML>` , `<!Doctype html>` , `<!doctype html>` , `<!doctype HTML>` . El resultado será el mismo para todos los casos.

*Obs: desde mayo de 2019 el desarrollo del HTML es mantenido por el W3C (World Wide Web Consortium) https://www.w3.org/, WHATWG y comunidad de desarrolladores, y su especificación es abierta en el Github https://github.com/whatwg/html, y desde este movimento el HTML es considerado un "patrón vivo" (living standard) donde su versión a partir del 5 es actualizada continuamente.*

Capítulo 5

# ESTILIZANDO CON CSS

Cuando escribimos código HTML, marcamos el contenido de la página con etiquetas (**tags**) que mejor representan el significado de aquel contenido. Cuando abrimos la página en el navegador es posible percibir que este muestra las informaciones con estilos diferentes.

Un h1, por ejemplo, por defecto es presentado en negrito con una fuente mayor. Párrafos de texto son espaciados entre sí, y así sucesivamente. Eso quiere decir que el navegador tiene un *estilo patrón* para las etiquetas que usamos.

Sin embargo, para hacer sitios bonitos, o con el visual próximo de una dada identidad visual (design), vamos a necesitar *personalizar la presentación patrón de los elementos* de la página.

Antiguamente, eso era hecho en el propio HTML. Caso hubiese la necesidad de que un título sea rojo, había que hacer:

```
<h1><font color="red">MusicDot años 90</font></h1>
```

Aparte de la etiqueta `<font>` , varias otras etiquetas de estilo existían. Pero eso es pasado. Hoy en día **usar etiquetas HTML para estilo es una mala práctica** y jamás deben ser usadas, son interpretadas apenas para el modo de compatibilidad.

En su lugar, surgió el **CSS** (*Cascading Style Sheet* o hoja de estilos en cascada), que es otro lenguaje, separada del HTML, con objetivo único de cuidar de la estilización de la página. La ventaja es que el CSS es más robusto que HTML para estilización, como veremos. Pero, principalmente, escribir la formatación visual mezclando con contenido de texto en HTML se mostró algo impracticable. El código CSS resuelve eso separando las cosas; las reglas de estilo no aparecen más en el HTML, apenas en el CSS.

## 5.1 SINTAXIS Y INCLUSIÓN DE CSS

La sintaxis del CSS tiene estructura simple: es una declaración de propiedades y valores separados por un signo de dos puntos ":", y cada propiedad es separada por un signo de punto y coma ";" de la siguiente forma:

```
color: blue;
background-color: yellow;
```

El elemento que recibe esas propiedades será exhibido con el texto en color azul y con el fondo amarillo. Esas propiedades pueden ser declaradas de tres formas diferentes.

## Atributo style

La primera de ellas es con el atributo `style` en el proprio elemento HTML:

```
<p style="color: blue; background-color: yellow;">
El contenido de esta etiqueta será exhibido en azul con fondo amarillo en el navegador!
</p>
```

Pero acabamos de discutir que una de las grandes ventajas del CSS era mantener las reglas de estilo fuera del HTML. Usando el atributo `style` no pareciera que hicimos eso. Justamente por eso no se recomienda ese tipo de uso en la práctica, pero sí, los que veremos a continuación.

## La etiqueta style

La otra forma de utilizar el CSS es declarando sus propiedades dentro de la etiqueta `<style>`.

Como estamos declarando las propiedades visuales de un elemento en otro lugar de nuestro documento, necesitamos indicar de alguna forma a cual elemento nos referimos. Hacemos eso utilizando un **selector CSS**. Es básicamente una forma de buscar ciertos elementos dentro de la página que recibieron las reglas visuales que queremos.

En el ejemplo a seguir, usaremos el selector que selecciona todas las etiquetas `p` y altera su color y el fondo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sobre la MusicDot</title>
    <style>
      p {
        color: blue;
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <p>
      El contenido de esta etiqueta será exhibido en azul con fondo amarillo!
    </p>
    <p>
      <strong>También</strong> será exhibido en azul con fondo amarillo!
    </p>
  </body>
</html>
```

El código dentro de la etiqueta `<style>` indica que estamos alterando el color y el fondo de todos los elementos con la etiqueta `p`. Decimos que seleccionamos esos elementos por el nombre de su etiqueta, y aplicamos ciertas propiedades CSS apenas en ellos.

Revisando entonces la estructura de uso del CSS:

```
selector {
    propiedad: valor;
}
```

Algunas propiedades contienen "sub propiedades" que modifican una parte específica de aquella propiedad que vamos a trabajar, siendo su sintaxis:

```
selector {
    propiedad-subpropiedad: valor;
}
```

En el ejemplo de abajo, en ambos casos, trabajamos con la propiedad `text`, que estiliza la apariencia del texto del selector informado. Podemos especificar cuáles propiedades específicas del texto queremos modificar, en el caso `text-align` la alineación del texto, y con `text-decoration` colocamos el efecto de subrayado.

```
p {
    text-align: center;
    text-decoration: underline;
}
```

## Archivo externo

La tercera forma de declarar los estilos de nuestro documento es con un archivo externo con la extensión `.css`. Para que sea posible declarar nuestro CSS en un archivo aparte, necesitamos indicar en nuestro documento HTML, un enlace entre él y la hoja de estilo (archivo con la extensión `.css`).

Aparte de mejorar la organización del proyecto, la hoja de estilo externa trae también las ventajas de mantener nuestro HTML más limpio y de reaprovechar una misma hoja de estilos para diversos documentos.

La indicación de uso de una hoja de estilos externa debe ser hecha dentro de la etiqueta `<head>` de un documento HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>MusicDot | Sobre la empresa</title>
    <!-- Inclusión del archivo CSS -->
    <link rel="stylesheet" href="estilos.css">
  </head>
  <body>
    <p>
      El contenido de esta etiqueta será exhibido en azul con fondo amarillo!
    </p>
    <p>
      <strong>También</strong> será exhibido en azul con fondo amarillo!
    </p>
  </body>
</html>
```

Y dentro del archivo `estilos.css` colocamos apenas el contenido del CSS:

```
p {
  color: blue;
  background-color: yellow;
}
```

## 5.2 PROPIEDADES TIPOGRÁFICAS Y FUENTES

De la misma forma que alteramos colores, podemos alterar el texto. Podemos definir fuentes con el uso de la propiedad `font-family` .

La propiedad `font-family` puede recibir su valor con o sin comillas dependiendo de su composición, por ejemplo, cuando una fuente tiene el nombre separado por *espacio*.

Por defecto, los navegadores más conocidos exhiben texto en un tipo que conocemos como "serif". Las fuentes más conocidas (y comúnmente utilizadas como patrón) son "Times" y "Times New Roman", dependiendo del sistema operacional. Ellas son llamadas de **fuentes serifadas** pelos pequeños ornamentos en sus terminaciones.

Podemos alterar la familia de fuentes que queremos utilizar en nuestro documento para la familia "sans-serif" (sin serifas), que contienen, por ejemplo, las fuentes "Arial" y "Helvetica". Podemos también declarar que queremos utilizar una familia de fuentes "monospace" como, por ejemplo, la fuente "Courier".

*Obs: Fuentes monospace pueden ser tanto con serifa o sin serifa. Monospace quiere decir apenas que todas las letras poseen el mismo tamaño*

```
h1 {
  font-family: serif;
}

h2 {
  font-family: sans-serif;
}

p {
  font-family: monospace;
}
```

Es posible, y es muy común, declarar el nombre de algunas fuentes que queremos verificar si existen en el computador, permitiendo que tengamos un mejor control de la forma como nuestro texto será exhibido.

En nuestro proyecto, las fuentes no tienen ornamentos, vamos a declarar esa propiedad para todo el documento por medio del su elemento `body` :

```
body {
  font-family: "Helvetica", "Lucida Grande", sans-serif;
}
```

En este caso, el navegador verificará si la fuentes "Helvetica" está disponible y la utilizará para exhibir los textos de todos los elementos del nuestro documento que, por cascada, heredarán esa propiedad del elemento `body`.

Caso la fuente "Helvetica" no esté disponible, el navegador verificará la disponibilidad de la próxima fuente declarada, en nuestro ejemplo sería la "Lucida Grande". Caso el navegador no encuentre también esa fuente, él mismo solicitará cualquier fuente que pertenezca a la la familia "sans-serif", declarada al final, y la utiliza para exhibir el texto, sin importar cuál sea esta.

Tenemos otras propiedades para manipular las fuentes, como la propiedad `font-style`, que define el estilo de la fuentes que puede ser: `normal` (normal en la vertical), `italic` (inclinada) y `oblique` (oblicua).

## 5.3 ALINEACIÓN Y DECORACIÓN DE TEXTO

Ya vimos una serie de propiedades y sub propiedades que determinan el tipo y estilo de la fuente. Vamos a conocer algunas formas de cambiar las disposiciones de los textos.

En el ejemplo a seguir vamos cambiar la alineación del texto con la propiedad `text-align`.

```
p {
  text-align: right;
}
```

El ejemplo determina que todos los párrafos de nuestra página, tengan el texto alineado para la derecha. También es posible determinar que un elemento tenga su contenido alineado al centro cuando definimos el valor `center` para la propiedad `text-align`, o entonces definir que el texto debe ocupar toda la anchura del elemento aumentando el espaciado entre las palabras con el valor `justify`. Por defecto, el texto es alineado a la izquierda, con el valor `left`, pero es importante recordar que esa propiedad se propaga en cascada.

Es posible configurar también una serie de espaciados de texto con el CSS:

```
p {
  line-height: 3px; /* tamaño de la altura de cada línea */
  letter-spacing: 3px; /* tamaño del espacio entre cada letra */
  word-spacing: 5px; /* tamaño del espacio entre cada palabra */
  text-indent: 30px; /* tamaño del margen de la primera línea del texto */
}
```

## 5.4 IMAGEN DE FONDO

La propiedad `background-image` permite indicar un archivo de imagen para ser exhibido al fondo del elemento. Por ejemplo:

```
h1 {
  background-image: url(sobre-background.jpg);
```

```
}
```

Con esa declaración, el navegador va solicitar un archivo `sobre-background.jpg` , que debe estar en la misma carpeta del archivo CSS donde consta esa declaración. Pero podemos también pasar una dirección de la web para usar imágenes remotamente:

```
body {
  background-image: url(https://i.imgur.com/uAhjMNd.jpg);
}
```

## 5.5 BORDES

Las propiedades del CSS para definir los **bordes** de un elemento nos presentan una serie de opciones. Podemos, para cada borde de un elemento, determinar su color, su estilo de exhibición y su anchura. Por ejemplo:

```
body {
  border-color: red;
  border-style: solid;
  border-width: 1px;
}
```

La propiedad `border` tiene una forma resumida para escribir los mismos estilos que añadimos arriba, pero de una forma más simple:

```
body {
  border: 1px solid red;
}
```

Para que la definición del color sobre el borde haga efecto, es necesario que la propiedad `border-style` tenga cualquier valor diferente del patrón `none` .

Podemos también definir en cuál de los lados del nuestro elemento queremos el borde usando la sub propiedad que indica la dirección:

```
h1 {
  border-top: 1px solid red; /* borde roja encima */
  border-right: 1px solid red; /* borde roja a la  derecha */
  border-bottom: 1px solid red; /* borde roja abajo*/
  border-left: 1px solid red; /* borde roja a la  izquierda */
}
```

Conseguimos hacer también comentarios en el CSS usando la siguiente sintaxis:

```
/* dejando el fondo de amarillo oro */
body {
    background-color: gold;
}
```

## 5.6 COLORES EN LA WEB

Propiedades como `background-color`, `color`, `border-color`, entre otras aceptan un color como valor. Existen varias formas de definir colores cuando utilizamos el CSS.

La primera, más simple, es usando el nombre del color:

```
h1 {
    color: red;
}

h2 {
    background-color: yellow;
}
```

Lo difícil es acertar la variación exacta de color que queremos en el *design* y también cada navegador tiene su patrón de color para los nombres de colores. La W3C obliga que todos los navegadores tengan por lo menos 140 nombres de colores estandarizados, pero existen más de 16 millones de colores en la web y sería extremadamente complicado nombrar cada una de ellos. Por eso, es raro usar colores con sus nombres. Lo más común es definir el color con base en su composición RGB.

RGB (Red, Green y Blue) es el sistema de color usado en los monitores, ya que cada pixel en los monitores poseen 3 leds (un rojo, un verde y un azul) y la combinación de esos 3 colores forman los 16 millones de colores que vemos en los monitores. Podemos escoger la intensidad de cada uno de esos tres leds básicos, en una escala de 0 a 255.

Un amarillo fuerte, por ejemplo, tiene 255 de Red, 255 de Green y 0 de Blue (255, 255, 0). Si quieres un color naranja, basta disminuir un poco el verde (255, 200, 0). Y así sucesivamente.

En CSS, podemos escribir los colores teniendo como base su composición RGB. En el CSS3 hay hasta una sintaxis bien simple para aquello, la vemos a continuación:

```
h3 {
    color: rgb(255, 200, 0);
}
```

Esta sintaxis funciona en los navegadores más modernos y hasta algunos navegadores antiguos pero no es la más común en la práctica, justamente por cuestiones de compatibilidad. Lo más común es la **notación hexadecimal**. Esa sintaxis tiene soporte universal en los navegadores y es más corta de escribir, a pesar de ser más difícil de ser leída y comprendida.

```
h3 {
    background-color: #f2eded;
}
```

En el fondo, sin embargo, las dos formas están basadas en el sistema RGB. En la notación hexadecimal (que comienza con #), se definen 6 caracteres, los primeros 2 indican el canal Red, los dos siguientes, el Green, y los dos últimos, Blue; o sea, RGB. Usamos la matemática para escribir menos

caracteres, cambiando la base numérica de decimal para hexadecimal.

En la base hexadecimal, los dígitos van de 0 a 15 (en vez de 0 a 9 de la base decimal común). Para representar las cifras de 10 a 15, usamos letras de A a F. En esta sintaxis, por tanto, podemos utilizar números de 0-9 y letras de A-F.

Hay una cuenta detrás de esas conversiones, pero su editor de imágenes debe ser capaz de entregar ambos valores sin problemas. Un valor 255 se convierte en FF en la notación hexadecimal. El color **#f2eded**, por ejemplo, es equivalente a **rgb(242, 237, 237)**, un gris claro.

Vale aquí un consejo en cuanto al uso de colores hexadecimales, cada vez que los caracteres presentes en la composición de la base se repitan, estos pueden ser simplificados. Entonces un número en hexadecimal **3366ff**, puede ser simplificado para **36f**.

*Obs: los 3 pares de números deben ser iguales entre sí, o sea, si tenemos un hexadecimal #33aabc no podemos simplificar nada del código.*

Capítulo 6

# ESPACIADOS Y DIMENSIONES

Tenemos algunas formas de trabajar con espaciados y dimensiones. Para espaciado interno y externo usamos respectivamente `padding` y `margin`, y para redimensionar elementos podemos usar las propiedades de anchura y altura o `width` y `height`. Vamos a ver más a fondo estas propiedades.

## 6.1 DIMENSIONES

Es posible determinar las dimensiones de un elemento, por ejemplo:

```
p {
  background-color: red;
  height: 300px;
  width: 300px;
}
```

Todos los párrafos de nuestro HTML ocuparán 300 pixeles de altura y de anchura, con el color de fondo rojo.

> Si usamos el inspector de elementos del navegador veremos que el restante del espacio ocupado por el elemento se convierte en `margin`

## 6.2 ESPACIADOS

### Padding

La propiedad **padding** es utilizada para definir un espaciado interno en elementos (por espaciado interno queremos decir, la distancia entre el límite del elemento, su borde, y su respectivo contenido) y tiene las sub propiedades listadas a seguir:

- padding-top
- padding-right
- padding-bottom
- padding-left

Esas propiedades aplican una distancia entre el límite del elemento y su contenido de arriba, a la derecha, abajo y a la izquierda respectivamente. Este orden es importante para que entendamos cómo funciona la *shorthand property* (acortamiento) del padding.

Podemos definir todos los valores para las subpropiedades del padding en una única propiedad, llamadas exactamente de `padding`, y su comportamiento es descrito en los ejemplos a seguir:

Si pasamos solamente un valor para la propiedad `padding`, ese mismo valor es aplicado en todas las direcciones.

```
p {
  padding: 10px;
}
```

Si pasamos dos valores, el primero será aplicado arriba y abajo (equivalente a pasar el mismo valor para `padding-top` y `padding-bottom`) y el segundo será aplicado a la derecha y a la izquierda (equivalente a pasar el mismo valor para `padding-right` y `padding-left`).

```
p {
  padding: 10px 15px;
}
```

Si pasamos tres valores, el primero será aplicado de arriba (equivalente a `padding-top`), el segundo será aplicado a la derecha y a la izquierda (equivalente a pasar el mismo valor para `padding-right` y `padding-left`) y el tercero valor será aplicado abajo del elemento (equivalente a `padding-bottom`).

```
p {
  padding: 10px 20px 15px;
}
```

Si pasamos cuatro valores, serán aplicados respectivamente a `padding-top`, `padding-right`, `padding-bottom` y `padding-left`. Para facilitar la memorización de ese orden, basta recordar que los valores son aplicados en **sentido horario**.

```
p {
  padding: 10px 20px 15px 5px;
}
```

Un tip para omitir valores del padding:

Cuando necesitas omitir valores, siempre omite en el sentido anti-horario comenzando a partir de la sub propiedad `-left` .

Como los valores tienen posicionamiento fijo a la hora de declarar los espaciados, el navegador no sabe cuándo y cuál valor debe ser omitido. Por ejemplo:

Se tenemos un padding:

```
h1 {
 padding: 10px 25px 10px 15px;
}
```

El código no puede sufrir el acortamiento porque por más que los valores de `top` y `bottom` sean iguales, los valores `right` y `left` no son y ellos son los primeros a ser omitidos. Vea lo que pasa cuando vamos a omitir el valor de `10px` del bottom:

```
h1 {
    padding: 10px 25px 15px;
}
```

El navegador va interpretar de la siguiente forma:

```
h1 {
    padding: top right bottom;
}
```

Que al final va quedar igual a:

```
h1 {
    padding-top: 10px;
    padding-right: 25px;
    padding-bottom: 15px;
    padding-left: 25px;
}
```

Y eses valores no son los que nosotros colocamos al comienzo con `padding: 10px 25px 10px 15px;`

## Margin

La propiedad `margin` es bien parecida con la propiedad `padding` , excepto que ella añade espacio después del límite del elemento, o sea, es un espaciado a parte del elemento en sí (espaciado externo). A parte de las sub propiedades listadas a seguir, la *shorthand property* `margin` se comporta de la misma forma que la *shorthand property* del `padding` vista en el tópico anterior.

- margin-top

- margin-right
- margin-bottom
- margin-left

Existe una forma extra de permitir que el navegador defina cuál será la dimensión de la propiedad `padding` o `margin` conforme el espacio disponible en la pantalla: definimos el valor `auto` para los espaciados que queremos.

En el ejemplo a seguir, definimos que un elemento no tiene ningún margen arriba o abajo de su contenido y dejamos que el navegador defina un margen igual para ambos lados laterales (izquierda y derecha) de acuerdo con el espacio disponible:

```
p {
  margin: 0 auto;
}
```

Capítulo 7

# LISTAS HTML

No son raros los casos en que queremos exhibir una lista en nuestras páginas. HTML tiene algunas etiquetas definidas para que podamos hacer eso de forma correcta. La lista más común es la lista no ordenada definida por la etiqueta `<ul>` (*unordered list*).

```
<ul>
  <li>Primer ítem de la lista</li>
  <li>
    Segundo ítem de la lista:
    <ul>
      <li>Primer ítem de la lista anidada</li>
      <li>Segundo item de la lista anidada</li>
    </ul>
  </li>
  <li>Tercer ítem de la lista</li>
</ul>
```

Note que, para cada ítem de la lista no ordenada, utilizamos una marcación de ítem de lista `<li>` (*list item*). En el ejemplo de arriba, utilizamos una estructura compuesta en la cual el segundo ítem de la lista contiene una nueva lista. La misma etiqueta de ítem de lista `<li>` es utilizada cuando demarcamos una lista ordenada.

```
<ol>
  <li>Primer ítem de la lista</li>
  <li>Segundo item de la lista</li>
  <li>Tercer ítem de la lista</li>
  <li>Cuarto ítem de la lista</li>
  <li>Quinto ítem de la lista</li>
</ol>
```

Las listas ordenadas ( `<ol>` - *ordered list*) también pueden tener su estructura compuesta por otras listas ordenadas como en el ejemplo que tenemos para las listas no ordenadas. También es posible tener listas ordenadas anidadas en un ítem de una lista no ordenada y viceversa.

## 7.1 LISTAS DE DEFINICIÓN

Existe un tercer tipo de lista que debemos utilizar para demarcar un glosario, cuando listamos términos y sus significados. Esa lista es la **lista de definición** (*definition list*).

```
<dl>
  <dt>HTML</dt>
  <dd>
    HTML es el lenguaje de marcación de textos utilizada
```

```
    para exhibir textos como páginas en Internet.
  </dd>
  <dt>Navegador</dt>
  <dd>
    Navegador es el software que requisita un documento HTML
    a través del protocolo HTTP y exhibe su contenido en una
    ventana.
  </dd>
</dl>
```

Para estilizar el formato patrón de las listas ordenadas y no ordenadas, podemos utilizar la propiedad `list-style-type` en el CSS:

```
ul {
  /* alterar para círculo antes de cada <li> de la lista no-ordenada */
  list-style-type: circle;
}

ol {
  /* alterar para una secuencia alfabética antes de cada <li> de la lista ordenada */
  list-style-type: upper-alpha;
}
```

> También podemos usar la *shorthand property* `list-style: circle`

## 7.2 LINKS EN HTML

Cuando necesitamos indicar que un trecho de texto se refiere a un otro contenido, sea en el mismo documento o en otra dirección (por ejemplo una página en la web), utilizamos la etiqueta de ancla `<a>`.

Existen tres diferentes usos para las anclas. Uno de ellos es la definición del **link** o enlace:

```
<p>
  Visite el sitio de <a href="https://www.aluracursos.com">Alura</a>.
</p>
```

Note que el ancla está marcando apenas la palabra **Alura** de todo el contenido del párrafo ejemplificado. Eso significa que, al hacer clic con el cursor del mouse en la palabra **Alura**, el navegador redireccionará el usuario para el sitio de **Alura**, indicado en el atributo `href`.

> Podemos agregar el atributo `target=""` para especificar dónde esa página irá cargar. Por defecto la página irá abrir en la misma pestaña de la página que tiene el link, pero si queremos que la página abra en otra pestaña podemos colocar el valor `_blank` dentro de ese atributo:
>
> ```
> <a href="https://www.aluracursos.com" target="_blank">
> ```

Otro uso para la etiqueta de ancla es la demarcación de destinos para links dentro del propio documento, lo que llamamos de ***bookmark***.

```
<p>Más informaciones <a href="#info">aqui</a>.</p>
<p>Contenido de la página...</p>

<h2 id="info">Más informaciones sobre el asunto:</h2>
<p>Informaciones...</p>
```

De acuerdo con el ejemplo de arriba, al hacer clic sobre la palabra **aquí**, demarcada con un link, el usuario será llevado a la porción de la página donde el *bookmark* **info** es visible. *Bookmark* es el elemento que tiene el atributo `id`.

Es posible, con el uso de un link, llevar el usuario a un *bookmark* presente en otra página.

```
<a href="https://www.aluracursos.com/contacto">
  Entre en contacto sobre el curso
</a>
```

El ejemplo de arriba hará con que el usuario que hace clic en el link sea llevado a una parte de la página indicada en la dirección, específicamente en el punto donde el *bookmark* **contacto** sea visible.

El otro uso para la etiqueta de ancla es la demarcación de destinos para los links dentro del propio sitio, pero no en la misma página donde estamos. Por ejemplo, estamos en la página **sobre.html** y queremos un link para la página **index.html**.

```
<p>Accede a <a href="index.html">nuestra tienda</a>.</p>
```

Capítulo 8

# SELECTORES MÁS ESPECÍFICOS Y HERENCIA

Durante el curso veremos otros tipos de selectores. Por ahora veremos un selector que deja nuestra estilización un poco más especifica de lo que hicimos hasta ahora.

Veamos el siguiente ejemplo:

**HTML:**

```
<img src="img/logo-musicdot" atl="Logo de MusicDot">

<figure>
    <img src="img/matriz-musicdot" alt="Foto de la matriz de MusicDot">
    <figcaption>Matriz de MusicDot</figcaption>
</figure>

<figure>
    <img src="img/familia-tupfeln" alt="Foto de la família Tupfeln">
    <figcaption>Família Tüpfeln</figcaption>
</figure>
```

**CSS:**

```
img {
    width: 300px;
}
```

En el código de arriba estamos aplicando una anchura de `300px` para todas las etiquetas `<img>`. Pero, ¿y si quisiéramos aplicar esa anchura apenas para las imágenes que están en las figuras? Es allí que entra el selector más específico:

```
figure img {
    width: 300px;
}
```

Ahora estamos aplicando la anchura de `300px` apenas a las imágenes que son hijas de la etiqueta `<figure>`.

Otra forma de seleccionar elementos más específicos es usando el atributo `id=""` en los elementos que queremos estilizar y después hacer la llamadas de selector de `id`:

**HTML:**

```
<img src="img/logo-musicdot" atl="Logo de MusicDot">

<figure>
    <img src="img/matriz-musicdot" alt="Foto de la matriz MusicDot" id="matriz-musicdot">
    <figcaption>Matriz de MusicDot</figcaption>
</figure>

<figure>
    <img src="img/familia-tupfeln" alt="Foto de la família Tupfeln" id="familia-tupfeln">
    <figcaption>Família Tüpfeln</figcaption>
</figure>
```

**CSS:**

```
#matriz-musicdot {
    width: 300px;
}

#familia-tupfeln {
    width: 300px;
}
```

Solo que no es recomendado el uso de `id` para la estilización de elementos ya que la idea del atributo es para hacer una referencia única en la página como hicimos en la parte de los links. Cuando queremos estilizar elementos específicos es mejor utilizar el atributo `class=""` . El comportamiento en el CSS será idéntico al del atributo `id=""` , pero `class` fue hecho para ser usado en el CSS y en el JavaScript.

Arreglando el ejemplo anterior, usando clases:

**HTML:**

```
<img src="img/logo-musicdot" atl="Logo de MusicDot">

<figure>
    <img src="img/matriz-musicdot" alt="Foto de la matriz de MusicDot" class="matriz-musicdot">
    <figcaption>Matriz de MusicDot</figcaption>
</figure>

<figure>
    <img src="img/familia-tupfeln" alt="Foto de la família Tupfeln" class="familia-tupfeln">
    <figcaption>Família Tüpfeln</figcaption>
</figure>
```

**CSS:**

```
.matriz-musicdot {
    width: 300px;
}

.familia-tupfeln {
    width: 300px;
}
```

## 8.1 GRADO DE ESPECIFICIDAD DE UN SELECTOR

Existe una cosa muy importante en el CSS que necesitamos tomar cuidado es el **grado de especificidad de un selector**. Esto es, la prioridad de interpretación de un selector por el navegador. Para entender estas reglas de especificidad de un selector vamos a usar una forma de puntuación. Al crear un **selector de etiqueta** su puntuación es **1**. Cuando usamos un **selector de clase** su puntuación es **10**. Cuando usamos un **selector de id** su puntuación vale **100**. Al final, el navegador suma la puntuación de los selectores aplicados a un elemento, y las propiedades con el selector de mayor puntuación son las que valen.

```
<body>
    <p class="parrafo" id="parrafo-rosado">Texto</p>
</body>
```

```
p { /* Puntuación 1 */
    color: blue;
}

.parrafo { /* Puntuación 10 */
    color: red;
}

#parrafo-rosado { /* Puntuación 100 */
    color: pink;
}
```

En el ejemplo de arriba el párrafo va quedar con el color rosado porque el selector que tiene el color rosado es el selector de mayor puntuación.

Cuando los elementos poseen la misma puntuación, que prevalece es la propiedad del último selector:

```
p { /* Puntuación 1 */
    color: blue;
}

p { /* Puntuación 1 */
    color: red;
}
```

En el ejemplo de arriba el color del párrafo será rojo.

Podemos también sumar los puntos para dejar nuestro selector más fuerte:

```
body p { /* Selector de tag + otro selector de tag = 2 pontos */
    color: brown;
}

p { /* Puntuación 1 */
    color: blue;
}
```

En el ejemplo de arriba dejamos nuestro selector más específico para los `<p>` que están dentro de una etiqueta `<body>` , por tanto el color del párrafo será marrón.

En resumen:

Cuanto más específico es nuestro selector, mayor su puntuación en el nivel de especificidad del CSS. Por lo tanto debemos siempre trabajar con una baja especificidad, para que no sea imposible sobrescribir valores cuando sea necesario en una situación específica.

## 8.2 HERENCIA

La cascada del CSS, significa justamente la posibilidad de que elementos hijos hereden características de estilización de elementos superiores. Estas características están definidas en las propiedades de los elementos padres, que pueden o no pasar a sus descendientes sus valores.

Vamos a ver el ejemplo del código a seguir:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Un ejemplo</title>
</head>
<body>
    <p>Una breve explicación de algo con un <a href="https://google.com">link</a> para una referencia
 de otra página</p>
    <figure>
        <img src="img/foto" alt="Una foto">
        <figcaption>Una foto</figcaption>
    </figure>
</body>
</html>
```

Vamos a cambiar la familia de las fuentes de toda la página. Una forma que podemos hacer es seleccionar todas las etiquetas que contienen texto ( `<p>` , `<a>` y `<figcaption>` ) y colocar la familia de fuentes que queremos:

```
p {
    font-family: 'Helvetica', sans-serif;
}

a {
    font-family: 'Helvetica', sans-serif;
}

figcaption {
    font-family: 'Helvetica', sans-serif;
}
```

Pero eso da mucho trabajo y estar repitiendo código. En vez de colocar esa propiedad en cada uno de los elementos textuales de la nuestra página, podemos colocar en el elemento superior a estas etiquetas, en este caso es el elemento `<body>` .

```
body {
    font-family: 'Helvetica', sans-serif;
}
```

En el ejemplo de arriba todos los elementos hijos de la etiqueta `<body>` van a recibir la propiedad

`font-family:` y eso es el que llamamos de **herencia**. La herencia surge cuando elementos heredan propiedades de los elementos de arriba de ellos (elementos padre).

*Obs: Para saber si una propiedad deja herencia o no, es posible consultar en su especificación o en el sitio MDN https://developer.mozilla.org/.*

## 8.3 PARA SABER MÁS: EL VALOR INHERIT

Imagina que tenemos la siguiente división con una imagen:

```
<div>
    <img src="box-model.png" alt="box model">
</div>
```

```
div {
    border: 2px solid;
    border-color: red;
    width: 30px;
    height: 30px;
}
```

Queremos que la imagen llene todo el espacio de la `<div>`, pero las propiedades `width` y `height` no son aplicadas en cascada, siendo así, somos obligados a definir el tamaño de la imagen manualmente:

```
img {
    width: 30px;
    height: 30px;
}
```

Esta no es una solución sustentable, porque, caso alteramos el tamaño de la `<div>`, tendremos que recordar de alterar también el tamaño de la imagen. Una forma de resolver este problema es utilizando el valor **inherit** para las propiedades `width` y `height` de la imagen:

```
img {
    width: inherit;
    height: inherit;
}
```

El valor `inherit` indica para el elemento hijo que debe utilizar el mismo valor presente en el elemento padre, siendo así, toda vez que el tamaño del elemento padre fuera alterado, automáticamente el elemento hijo heredará el nuevo valor, facilitando así, el mantenimiento del código.

Acuérdate de que el `inherit` también afecta propiedades que no son aplicadas en cascada.

Capítulo 9

# DESACOPLAMIENTO CON CLASES

Muchas veces cuando estamos declarando los estilos de una página HTML, encontramos más fácil usar el selector de nombre de la etiqueta en vez de usar clases. Pero esto puede causar problemas imprevistos si no es usado con cautela.

Vamos a analizar el siguiente código para entender la situación:

**HTML:**

```
<h1>MusicDot</h1>

<h2>Historia</h2>
<p>Texto</p>

<h2>Diferenciales</h2>
<ul>
    <li>Diferencial 1</li>
    <li>Diferencial 2</li>
    <li>Diferencial 3</li>
</ul>
```

**CSS:**

```
h2 {
    font-size: 24px;
    font-weight: bold;
    border-bottom: 1px solid #000000;
}
```

En el ejemplo de arriba agregamos estilo en las etiquetas `<h2>` para tener un tamaño de fuente mayor, una fuente más gruesa y un borde inferior para hacer el efecto de línea divisoria. Hasta aquí está todo cierto. Pero ahora vamos a colocar otro título en la página llamado "Sobre MusicDot" y ese título tiene relevancia mayor que los otros dos títulos que ya tenemos (Historia y Diferenciales), por lo tanto, vamos a tener que modificar sus etiquetas para una de menor importancia:

```
<h1>MusicDot</h1>

<h2>Sobre MusicDot</h2>
<p>Introducción</p>

<!-- Cambiamos para h3 pues queremos que tengan menos relevancia que el título "Sobre MusicDot" -->
<h3>Historia</h3>
<p>Texto</p>

<!-- Cambiamos para h3 pues queremos que tengan menos relevancia que el título "Sobre MusicDot" -->
<h3>Diferenciales</h3>
```

```
<ul>
    <li>Diferencial 1</li>
    <li>Diferencial 2</li>
    <li>Diferencial 3</li>
</ul>
```

Ahora con esos cambios de la estructura del HTML, nuestro CSS está cambiando un elemento que no es el que inicialmente queríamos cambiar. Vamos a tener que hacer el cambio en el CSS para usar nuestras alteraciones en el elemento cierto.

En este ejemplo la solución es relativamente simple, pero imagine que tenemos selectores más complejos, con herencias, etc. La alteración ya no sería tan simple. Por eso lo ideal es declarar estilos con clases en vez de nombres de etiquetas. Un ejemplo para dar nombre a las clases es que ellas representen el papel que estas etiquetas están ejerciendo en conjunto con los estilos declarados, en nuestro caso, estamos declarando un conjunto de estilo para subtítulos.

Vea cómo queda el resultado del desacoplamiento del conjunto de estilos del nombre de la etiqueta, para que que ahora sea con clases:

**HTML:**

```
<h1>MusicDot</h1>

<h2>Sobre la MusicDot</h2>
<p>Introducción</p>

<!-- Adicionamos la classe subtítulo-->
<h3 class="subtitulo">Historia</h3>
<p>Texto</p>

<!-- Adicionamos la classe subtítulo-->
<h3 class="subtitulo">Diferenciales</h3>
<ul>
    <li>Diferencial 1</li>
    <li>Diferencial 2</li>
    <li>Diferencial 3</li>
</ul>
```

**CSS:**

```
.subtitulo {
    font-size: 24px;
    font-weight: bold;
    border-bottom: 1px solid #000000;
}
```

Usando clases, podemos modificar toda la estructura HTML sin preocuparnos si estas alteraciones afectarán la estilización que hicimos al comienzo.

Capítulo 10

# ELEMENTOS ESTRUCTURALES Y SEMÁNTICA DE LOS ELEMENTOS

Vimos muchas etiquetas para representar diversos elementos en nuestra página HTML como, por ejemplo, `<h1>` para títulos, `<p>` para párrafos, `<figure>` para figuras, etc. Nuestra mayor preocupación con el desarrollo de páginas debe ser conseguir representar todo con etiquetas que condicen con su contenido. Veremos etiquetas como, por ejemplo, `<section>`, `<article>`, `<address>`, entre otras, con la intención de representar con mayor precisión el contenido que queremos mostrar. Estas etiquetas son llamadas también de elementos semánticos, o sea, que consiguen pasar una información con un significado específico para el contenido interpretado por el navegador, de esta forma no depende apenas del texto dentro de la etiqueta para entender lo que hay en aquella parte del sitio.

Uno de los grandes motivos para preocuparnos con la semántica que usamos en el sitio viene de las herramientas de indexación de buscadores, que colocan los sitios más semánticos y estructurados como prioridad en las respuestas de las búsquedas. Otra preocupación es con las herramientas de accesibilidad, que permiten que personas con deficiencia, por ejemplo personas ciegas, consigan usar un sitio a través de programas lectores de pantalla de forma estandarizada y sin problemas.

Una cosa que necesitamos recordar es que debemos escoger las etiquetas por lo que ellas representan y no como son mostradas en la pantalla del navegador. Estilización debe quedar en el CSS y estructura en el HTML.

Las únicas etiquetas que son de propósito genéricas y que son usadas apenas para facilitar la estilización en el CSS son los etiquetas `<div>` y `<span>`. Estas dos etiquetas no representan ningún contenido necesariamente, `<div>` representa una división de bloques y `<span>` una marcación para texto (sin quebrar la línea del texto).

Capítulo 11

# CONOCIENDO METODOLOGÍAS DE CSS

Vimos el concepto de desacoplar estilos usando clases y los beneficios que eso nos trae, pero para cada elemento que vamos a estilizar necesitamos pensar en un nombre diferente y eso puede quedar muy complicado sin una metodología a seguir.

Existen varias metodologías de CSS pero durante el curso vamos a usar una llamado ***BEM*** (Block Element Modifier). La ventaja de usar ***BEM*** para quien está comenzando con desarrollo HTML y CSS es que, es que es una metodología que se enfoca en la estructura y facilita bastante a la hora de planificar los nombres de las clases.

**BEM** usa un concepto de `bloque__elemento--modificador` para nombrar sus clases, siendo que `bloque` es el elemento html que representa una división de contenido cuya existencia ya tiene un sentido por sí misma, `elemento` representa una parte semántica del `bloque` y `modificador` es una señalización de *comportamiento* o *estilización*.

> Los divisores entre `bloque__elemento--modificador` son llamados de: double `snake__case` y double `kebab--case`. Cuando queremos una división como el *espacio* usamos o `kebab-case` o el `camelCase`. `Kebab-case` es el más común para HTML y CSS y `camelCase` es más común en JavaScript.

Vamos a ver cómo funciona **BEM** en el siguiente ejemplo:

```
<!-- section representa una pantalla (por ejemplo) de productos. Pero no de cualquier producto, sino
de productos más vendidos -->
<section class="productos productos--más-vendidos">
    <!-- El h2 representa el título de ese panel -->
    <h2 class="productos__titulo">Productos más vendidos</h2>
    <ul class="productos__lista">
        <!-- li representa el producto en sí -->
        <li class="productos__producto">
            <figure>
                <img src="img/producto1.png" alt="Foto del Producto 1">
                <figcaption>Foto del producto 1</figcaption>
            </figure>
        </li>
        <!-- li representa el producto en sí, pero en este caso también tenemos un producto en destaq
ue -->
        <li class="productos__producto productos__producto--destaque">
```

```
            <figure>
                <img src="img/producto-destaque.png" alt="Foto del Producto en Destaque">
                <figcaption>Foto del producto en destaque</figcaption>
            </figure>
        </li>
        <li class="productos__producto">
            <figure>
                <img src="img/producto3.png" alt="Foto del Producto 3">
                <figcaption>Foto del producto 3</figcaption>
            </figure>
        </li>
        <li class="productos__producto">
            <figure>
                <img src="img/producto4.png" alt="Foto del Producto 4">
                <figcaption>Foto del producto 4</figcaption>
            </figure>
        </li>
    </ul>
</section>
```

De la forma que montamos la estructura de arriba queda más fácil saber lo que estamos estilizando en el CSS. Vea la diferencia:

```
section h2 { /* ¿Es el h2 de la sección que tiene los productos? ¿Y si necesito cambiar mi  estructur
a para un h3? */
    font-size: 40px;
    font-weight: 800;
}

.productos__titulo { /* Ahora aquí, yo se que voy a estilizar el título de la pantalla de productos.
Incluso cambiando para un h3 continuará funcionando*/
    font-size: 40px;
    font-weight: 800;
}
```

## 11.1 TIPOS DE DISPLAY

Existen 2 tipos de display que caracterizan la exhibición patrón de la mayor parte de los elementos HTML: `display: block` y `display: inline` . La forma más fácil de ver la diferencia entre ellos es usando las etiquetas que poseen esas propiedades por defecto, `<p>` y `<a>` respectivamente, y colocar una color de fondo.

**HTML:**

```
<p>Un párrafo que es block</p>
<a>Un link que es inline</a>
```

**CSS:**

```
p {
    background-color: blue;
}

a {
    background-color: red;
}
```

Si analizamos el espacio que esos elementos ocupan. La etiqueta `<p>` ocupa toda la anchura de la página en cuanto la etiqueta `<a>` ocupa apenas el espacio necesario para mostrar el texto que colocamos. Vamos a colocar más elementos en nuestro ejemplo de arriba.

**HTML:**

```
<p>Un párrafo que es block</p>
<p>Otro párrafo que es block</p>
<a>Un link que es inline</a>
<a>Otro link que es inline</a>
```

**CSS:**

```
p {
    background-color: blue;
}

a {
    background-color: red;
}
```

Con esta modificación un párrafo quedó uno debajo del otro y los links quedaron uno del lado del otro. Esos comportamientos son los esperados de los elementos `block` y `inline`. Como un elemento `block` ocupa toda la anchura de la pantalla, no podemos colocar otro elemento del lado porque no hay espacio. Ahora como en `inline` el elemento ocupa sólo el espacio necesario para mostrar nuestro texto entonces podemos colocar otros elementos que quepan en aquel espacio. Bueno, vamos a resolver el problema de espacio de la etiqueta `<p>`:

**HTML:**

```
<p>Un párrafo que es block</p>
<p>Otro párrafo que es block</p>
<a>Un link que es inline</a>
<a>Otro link que es inline</a>
```

**CSS:**

```
p {
    background-color: blue;
    width: 30%;
}

a {
    background-color: red;
    width: 60%;
}
```

Bueno, ahora tenemos dos problemas. Incluso con el espacio extra, los párrafos no quedaron uno al lado del otro y nuestros links no tuvieron alteraciones en sus anchuras. Usando el inspector de elementos de nuestro navegador podemos ver lo que está pasando con esos elementos.

Seleccionando la etiqueta `<p>` en nuestro inspector conseguimos ver que esta realmente está

ocupando `30%` del espacio de la pantalla del navegador, pero tiene un **Margen** extra, que no fuimos nosotros que lo colocamos en el CSS. Pues existe una `margin` ocupando el restante del espacio que era ocupado por la etiqueta `<p>`. Utilizando la propiedad `margin-right: 0px;` no parece hacer efecto. Sin embargo, ese es el comportamiento esperado de un elemento `block`.

Vamos a ver ahora lo que pasó con nuestros links. Nuestros links parecen haber ignorado completamente la propiedad de anchura que colocamos. Más una vez, está correctamente, ese es el comportamiento patrón de un elemento `inline`. Diferente de un elemento `block`, un elemento `inline` no recibe propiedades de tamaño (`width` y `height`) y eso puede generar algunos problemas con estilización. Fue creado entonces el `display: inline-block` que permite usar lo mejor de los dos mundos. Vamos usar el mismo ejemplo nuevamente solo que cambiando el tipo de display del link:

**HTML:**

```
<p>Un párrafo que es block</p>
<p>Otro párrafo que es block</p>
<a>Un link que es inline</a>
<a>Otro link que es inline</a>
```

**CSS:**

```
p {
    background-color: blue;
    width: 30%;
}

a {
    background-color: red;
    width: 60%;
    display: inline-block;
}
```

¡Perfecto! Ahora nuestro link recibe el tamaño que colocamos y es exhibido un `<a>` debajo del otro. Si cambiamos el tamaño de esa etiqueta `<a>` para un tamaño de `40%`, por ejemplo, vemos que nuestras etiquetas `<a>` quedan una al lado del otro.

Un resumen de los displays:

- `display: block`:
  - El elemento ocupa toda la anchura disponible
  - Si disminuimos el tamaño de ese elemento el espacio restante será ocupado por una `margin` que no puede ser retirada
- `display: inline`:
  - Permite que otro elemento quede a su lado caso haya espacio
  - El elemento ocupa apenas el espacio necesario para mostrar su contenido

- No recibe propiedades de tamaño

- `display: inline-block`:
  - Permite que otro elemento quede a su lado caso haya espacio
  - El elemento inicialmente ocupa apenas el espacio para mostrar su contenido
  - Puede recibir propiedades de tamaño

## ¿Cuándo usar `inline` o `inline-block`?

Lo ideal es nunca limitar nuestras opciones cuando vamos escoger una propiedad. Si el único propósito de cambiar el `display` de un elemento es para dejarlo uno al lado del otro, vamos a usar `inline-block`. Si por algún motivo hubiera necesidad de cambiar el tamaño de ese elemento, ya estamos con el `display` correcto y no necesitaremos cambiarlo de nuevo.

Capítulo 12

# UNIDADES RELATIVAS CON EM Y REM

Para definir el espacio a ser usado por un elemento podemos hacer uso de medidas relativas con `%`. Usamos esa medida relativa cuando queremos que un elemento use, por ejemplo, `100%` del espacio disponible.

**HTML:**

```
<div>
    <img src="img/flor.png" alt="Foto de una flor">
</div>
```

**CSS:**

```
div {
    width: 400px;
}
```

En el ejemplo pasado, dependiendo del tamaño de la imagen, la imagen puede ultrapasar el espacio que definimos para la `<div>` o puede ocupar un espacio menor. Si queremos que la imagen ocupe todo el espacio de la `<div>` podemos usar la unidad relativa `%`:

```
div {
    width: 400px;
}

img {
    width: 100%; /* Ocupe 100% del espacio disponible */
}
```

La gran ventaja de usar `%` es que no importa el tamaño que colocamos en la `<div>`, la `<img>` siempre va acompañar el tamaño de la etiqueta madre (la `<div>`).

**EM** y **REM** tienen el mismo concepto de `%` pero en vez de ser basadas en el tamaño de un elemento, esas medidas son basadas en tamaño de fuente. **EM** usa el tamaño de la fuente del elemento padre y **REM** usa el tamaño de las fuentes del `<html>`.

**HTML:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Un ejemplo</title>
</head>
```

```
<body>
    <div>
        <img src="img/foto" alt="Una foto">
    </div>
</body>
</html>
```

**CSS:**

```
html {
    font-size: 10px;
}

div {
    font-size: 20px;
}

img {
    width: 10em; /* La anchura será de 200px */
    height: 10rem; /* La altura será de 100px */
}
```

La ventaja de usar esas medidas es que si tenemos otros elementos usando esas medidas y necesitamos cambiar el tamaño de todos los elementos proporcionalmente, basta cambiar en un solo lugar. Estas unidades de medida son ideales para cuando el sitio necesita ser exhibido en diferentes tamaños de pantalla, donde en cada tamaño de pantalla la fuente debe ser exhibida en escalas de tamaños diferentes y proporcionales entre sí.

Capítulo 13

# SITIO WEB MÓVIL VS SITIO WEB DESKTOP

El volumen de usuarios que acceden a Internet por medio de dispositivos móviles creció exponencialmente en los últimos años. Usuarios de iPhones, iPads y otros *smartphones* y tablets tienen demandas diferentes de los usuarios desktop. Pues estos dispositivos muchas veces están conectados en una red móvil con datos transmitidos vía 3G o 4G, este tipo de conexión puede presentar inestabilidad en la velocidad de carga de datos y archivos. A parte de eso es preciso preocuparse con la accesibilidad para personas con deficiencia, que también usan este tipo de dispositivo en el día a día, como sus recursos de comando de voz, y pantalla *touchscreen* que soporta diferentes tipos de gestos con los dedos, activando funcionalidades del *smartphones*.

**¿Cómo atender a esos usuarios?**

Para que nuestro sitio soporte usuarios móviles, antes que nada, necesitamos tomar una decisión: ¿hacer un sitio exclusivo y diferente, enfocado en dispositivos móviles o adaptar nuestro sitio para que funcione en cualquier dispositivo?

Varios sitios en el internet adoptan la estrategia de tener un sitio diferente orientado para dispositivos móviles usando un subdominio diferente como "m." o "mobile.", como https://m.youtube.com.

Ese abordaje es el que trae mayor facilidad a la hora de pensar en las capacidades de cada plataforma, desktop y móvil, permitiendo que entreguemos una experiencia personalizada y optimizada para cada situación. Pero, hay diversos problemas envueltos en esta opción:

- ¿Cómo atender adecuadamente diversos dispositivos tan diferentes como es el caso de un *smartphone* con pantalla pequeña o una tablet con pantalla mediana? ¿Y si consideramos también los SmartTVs, ChromeCast, AppleTV? ¿Tendríamos que montar un sitio específico para cada tipo de plataforma?
- Muchas veces esos sitios móviles son versiones limitadas de los sitios web originales y no unicamente ajustes de usabilidad para dispositivos diferentes. Eso frustra a las personas que usan, cada vez más, dispositivos móviles para completar las mismas tareas que antes hacían en el desktop.
- Dar mantenimiento en un sitio ya es bastante trabajoso, imagine dar mantenimiento en dos.

- Tendremos contenidos duplicados en sitios "diferentes", pudiendo perjudicar su SEO (optimización para motores de búsqueda) caso no sea realizado con el cuidado que piden las recomendaciones para este escenario.
- Tendremos que tratar con redireccionamiento entre URLs móviles y normales, dependiendo del dispositivo. Como por ejemplo, si una persona recibe un link de una página con la dirección del sitio desktop, y abre en el celular, tendrá que ser redireccionada automáticamente para la versión móvil. Y la misma cosa en el sentido contrario, al abrir un link de la dirección móvil en un computador o pantalla grande, deberá ser redireccionado para la URL normal.

Un abordaje que acostumbra ser muy utilizada es la de tener un único sitio, accesible en todos los dispositivos móviles. Adeptos de la idea de la Web única (**One Web**) consideran que lo mejor para el usuario es tener el mismo sitio del desktop normal también accesible en el mundo móvil. Es lo mejor también para quien desarrolla, que no necesitará mantener varios sitios diferentes. Y es lo que garantiza la compatibilidad con la mayor variedad de dispositivos.

Ciertamente, la idea no es hacer el acceso al sitio móvil exactamente de la misma forma que el desktop. Usando las tecnologías del CSS3, hoy muy bien soportadas por los navegadores, podemos usar la misma base de layout y marcación pero ajustando el design para cada tipo de dispositivo.

Hoy en día no existe tanto esa creencia de que el sitio necesita tener exactamente la misma experiencia que en el desktop. Podemos crear experiencias exclusivas para cada tipo de dispositivo, pero es importante que el usuario aun consiga hacer las funciones (por ejemplo realizar una compra).

**¿Cómo desarrollar un sitio exclusivo para Móvil?**

Desde el punto de vista de código, es el abordaje más simple: basta hacer su página con design más ajustado y llevando en cuenta que la pantalla será pequeña (En general, se usa width de 100% para que se adapte a las pequeñas variaciones de tamaños de pantallas entre *smartphones* diferentes).

Una dificultad estará en el servidor para detectar si el usuario está accediendo de un dispositivo móvil o no, y redireccionarlo para el lugar cierto. Eso acostumbra envolver código en el servidor que detecta el navegador del usuario a través del *User-Agent* del navegador.

Es una buena práctica también incluir un link para la versión normal del sitio caso el usuario no quiera la versión móvil.

## 13.1 CSS MEDIA TYPES

Desde la época del CSS2, hay una preocupación con el soporte de reglas de *layout* diferentes para cada situación posible. Eso es realizado usando los llamados *media types*, que pueden ser declarados en la etiqueta link del HTML a través del atributo `media`:

**HTML:**

```
<link rel="stylesheet" href="site.css" media="screen">
<link rel="stylesheet" href="print.css" media="print">
<link rel="stylesheet" href="handheld.css" media="handheld">
```

Otra forma de declarar los *media types* es separar las reglas dentro del propio archivo CSS usando la notación `@media`:

```
@media screen {
  body {
    background-color: blue;
    color: white;
  }
}

@media print {
  body {
    background-color: white;
    color: black;
  }
}
```

El *media type screen* determina la visualización patrón, que es una pantalla digital (monitores de computador o pantallas de *smarthphones*). Es muy común también tener un CSS con *media type print* con reglas de impresión (por ejemplo, retirar navegación, formatear colores para dejarlos más adecuados para lectura en papel, etc). Había también el *media type handheld*, dirigido para dispositivos móviles. Con él, conseguíamos adaptar el sitio para los pequeños dispositivos como celulares tipo WAP y *palmtops*.

El problema es que ese tipo *handheld* nació en una época en que los celulares eran mucho más simples que los que tenemos hoy, principalmente sus pantallas, por lo tanto, se acostumbraban formatos simples para visualización de las páginas.

Cuando los nuevos *smartphones touchscreen* comenzaron a surgir - en especial, el iPhone, estos tenían la capacidad para abrir páginas completas y tan complejas como las del desktop, debido a su pantalla digital avanzada. Por eso, el iPhone y otros celulares modernos ignoran las reglas de *handheld* y acaban por encajarse en la categoría *media type screen*.

Aparte de eso, incluso si *handheld* funcionara en los *smartphones*, ¿como trataríamos los diferentes dispositivos de hoy en día como tablets, televisores, etc?

La solución vino con el CSS3 y sus ***media queries***.

## 13.2 CSS3 MEDIA QUERIES

Todos los *smartphones* y navegadores modernos soportan una nueva forma de adaptar el CSS basado en las propiedades de los dispositivos, las **media queries** del CSS3.

En vez de simplemente hablar que determinado CSS es para *handheld* en general, nosotros podemos ahora indicar que determinadas reglas del CSS deben ser vinculadas a propiedades del dispositivo como tamaño de la pantalla, orientación (*landscape* o *portrait*) y hasta resolución en **dpi** (*dots per inch*).

```
<link rel="stylesheet" href="base.css" media="screen">
<!-- usando media queries -->
<link rel="stylesheet" href="mobile.css" media="(max-width: 480px)">
```

Otra forma de declarar los *media queries* es separar las reglas dentro del mismo archivo CSS:

```
@media screen {
  body {
    font-size: 16px;
  }
}

@media (max-width: 480px) {
  body {
    font-size: 1em;
  }
}
```

Vea como `@media` ahora puede recibir expresiones complejas. En este caso, estamos indicando que queremos que las pantallas con anchura máxima de 480px tengan una fuente de 1em.

Puedes probar eso apenas redimensionando el propio navegador desktop para un tamaño menor que 480px.

## 13.3 VIEWPORT

Pero, si intentamos rodar nuestro ejemplo anterior en un iPhone o Android de verdad, veremos que aun estamos viendo la versión desktop de la página. La regla del `max-width` parece ser ignorada.

En verdad, la cuestión es que los *smartphones* modernos tienen pantallas grandes y resoluciones altas, justamente para permitirnos ver las fotos y vídeos en alta resolución. Las dimensiones de la pantalla de un iPhone SE por ejemplo es 1280px por 720px. Y existen *smartphones* con Android que llegan a tener pantallas con resolución 4K.

Aun así, la experiencia en estos celulares es bien diferente que en los desktops. 4K en una pantalla de 4 pulgadas es bien diferente de 4K en un notebook de 16 pulgadas. La resolución cambia. Los celulares acostumbran tener una resolución en dpi bien mayor que los desktops.

**¿Cómo organizar nuestra página?**

Los *smartphones* saben que considerar la pantalla como 4K no ayudará al usuario a visualizar una página Web optimizada para pantallas menores. Para ello el concepto de *device-width* que,

resumidamente, representa un número en *pixels* que el fabricante del aparato considera como más próximo de la sensación que el usuario tiene al visualizar la pantalla.

En los iPhones, por ejemplo, el *device-width* es considerado como 370px, incluso hasta en dispositivos con capacidad de exhibir una resolución mucho más alta.

Por patrón, iPhones, Androids y similares acostumbran considerar el tamaño de la pantalla visible, llamadas de *viewport* como lo suficientemente grande para comportar los sitios desktop normales. Por eso nuestra página es mostrada sin *zoom* como si estuviésemos en el desktop.

Apple creó entonces una solución utilizando metadatos, que después fue copiada por otras marcas de *smartphones*, que es configurar el valor que juzgamos más adecuado para el *viewport*:

```
<meta name="viewport" content="width=370">
```

Eso hace con que la pantalla sea considerada con anchura de 370px, haciendo con que nuestro *layout* móvil finalmente funcione y nuestras *media queries* también.

Aún mejor, podemos colocar el *viewport* con el valor `device-width` definido por el fabricante, dando más flexibilidad con dispositivos diferentes con tamaños diferentes:

```
<meta name="viewport" content="width=device-width">
```

## 13.4 RESPONSIVE WEB DESIGN

Los pequeños cambios que hacemos usando `@media` intentando hacer que la experiencia del usuario en diversos dispositivos sea más atrayente es lo que el mercado llama de ***Web Design Responsive***. Este termino surgió en un famoso articulo de *Ethan Marcotte* y dice lo siguiente:

**Son 3 los elementos de un *design* responsivo:**

- *layout* fluido usando medidas flexibles, como porcentajes
- *media queries* para ajustes de design
- uso de imágenes flexibles

La idea del ***Web Design Responsive*** es que la página se **adapte a diferentes condiciones**, en especial a diferentes resoluciones. A pesar que el uso de porcentajes existen hace décadas en la Web, fue la popularización de las *media queries* que permitieron *layouts* verdaderamente adaptativos.

## 13.5 MOBILE-FIRST

En la construcción de Sitios Web hasta hace unos años se seguía como base el proceso que llamamos de ***"desktop-first"***. Eso significa que proyectamos nuestra página para el *layout* de desktop y, en un segundo momento, se realizar la adaptación para ambiente móvil.

En la práctica, actualmente esto ya no es muy interesante porque necesitamos **deshacer** algunas cosas ya construidas del sitio como ser posicionamientos y ajustes de altura y/o achura para poder adaptar para *mobile*.

Es mucho más común y recomendado el uso de la práctica inversa: el ***mobile-first***. O sea comenzar el desarrollo para el móvil y, después, agregar el soporte a *layouts* de desktop. En el código, no existe ningún secreto, la clave es usar más *media queries* `min-width` en vez de `max-width`, más común en códigos con estrategias ***desktop-first***.

El grande cambio del ***mobile-first*** es que permite un abordaje mucho más simple e incremental. Se comienza el desarrollo por el área más simple y limitada, con más restricciones. El uso de la pantalla pequeña nos va a forzar a crear páginas más simples, enfocadas y objetivas. Después, la adaptación para desktop con *media queries*, es apenas una cuestión de adaptar el layout.

En contrapartida el abordaje ***desktop-first*** comienza por el ambiente más libre y va intentando cortar cosas cuando llega en el móvil. Ese tipo de adaptación es, en la práctica, mucho más trabajosa.

Capítulo 14

# EL PROCESO DE DESARROLLO DE UNA PANTALLA

Existe hoy en el mercado una gran cantidad de empresas especializadas en el desarrollo de sitios y aplicaciones web, así como algunas empresas de desarrollo de software o agencias de comunicación que tienen personas capacitadas para ejecutar ese tipo de proyecto.

Cuando es detectada la necesidad del desarrollo de un sitio o aplicación web, la empresa que tiene esa necesidad debe pasar todas las informaciones relevantes del proyecto para la empresa que va a ejecutarlo. La empresa responsable por su desarrollo debe analizar muy bien esas informaciones y utilizar investigaciones que complementen al proyecto y claro certificarse de la validez de esas informaciones.

Un proyecto de sitio o aplicación web envuelve muchas disciplinas en su ejecución, pues son diversas características a ser analizadas y diversas las posibilidades presentadas por una plataforma. Por ejemplo, debemos conocer muy bien las características del público objetivo, pues él define cuál es el mejor abordaje para definir la navegación, tono lingüístico y visual a ser adoptado, entre otras. La afinidad del público con Internet y el dispositivo puede inclusive definir el tipo y la intensidad de las innovaciones que pueden ser utilizadas.

Por eso, la primera etapa del desarrollo del proyecto queda a cargo de la persona que cuida de la experiencia de usuario (*UX Designer*) junto con una persona de *Design* y alguien de contenido. Ese grupo de personas analiza y dirige una serie de informaciones de las características que interactuarán con el sitio, definiendo la cantidad, contenido, localización y estilización de cada información.

El resultado del trabajo de ese equipo es una serie de definiciones sobre la navegación (mapa del sitio) y un esbozo de cada una de las visiones, que son los *layouts* de las páginas, y visiones parciales como, por ejemplo, los diálogos de alerta y confirmación de la aplicación. Como esas visiones son aún esbozos, la parte de estilo del sitio queda más genérica: son utilizadas fuentes, colores e imágenes neutras, aunque las informaciones escritas deben ser definidas en esa fase del proyecto.

Esos esbozos de las visiones son lo que llamamos de ***wireframes*** y guían durante el proceso de *design*.

Con los ***wireframes*** en manos, es hora de agregar las imágenes, colores, fuentes, fondos, bordes y otras características visuales. Este trabajo es realizado por el mismo equipo ya mencionado que utilizan

herramientas gráficas como Adobe Photoshop, Adobe Illustrator, Figma, entre otras. El resultado del trabajo de ese equipo es que llamamos de ***layout***. Los *layouts* son imágenes estáticas con el visual completo a ser implementado. A pesar de que los navegadores sean capaces de exhibir imágenes estáticas, exhibir una única imagen para el usuario final en el navegador no es la forma ideal de publicar una página.

Para que las informaciones sean exhibidas de forma correcta y para posibilitar otras formas de uso e interacción con el contenido, es necesario que el equipo de **programación front-end** transforme esas imágenes en páginas interactivas utilizables por los navegadores.

De todas las tecnologías disponibles, la más recomendada es ciertamente el HTML, pues es el lenguaje que el navegador entiende. Todas las otras tecnologías citadas dependen del HTML para ser exhibidas correctamente en el navegador y, últimamente, el uso del HTML, en conjunto con el CSS y JavaScript, ha evolucionado al punto de que podemos sustituir algunas de esas otras tecnologías donde teníamos más poder y control en relación a la exhibición de gráficos, efectos e interactividad.

## 14.1 ANALIZANDO EL LAYOUT

Antes de escribir cualquier código, es necesario un análisis del *layout*. Con ese análisis, definiremos las principales áreas y bloques de nuestras páginas. Note que hay un encabezado (un área que potencialmente se repetirá en más de una página), un rodapié y un contenido principal. Siguiendo el pensamiento de escribir nuestro código pensando en semántica en primer lugar, ya podemos imaginar como será la estructura en el documento HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
    <title>MusicDot</title>
</head>
<body>
    <header>
        <!-- Contenido del header -->
    </header>
    <main>
        <!-- Contenido principal -->
    </main>
    <footer>
        <!-- Contenido del footer -->
    </footer>
</body>
</html>
```

Una recomendación es la de comenzar a planear el código siempre analizando de fuera para dentro. Por lo tanto, después de ver las 3 principales camadas ( `<header>` , `<main>` y `<footer>` ) vamos a profundizarnos en una de ellas. Vamos a partir del orden de la declaración y profundizarnos más en la etiqueta `<header>` . Dentro de `header` tenemos un **logo** y 3 **links**. Ya sabemos que el logo es una

**imagen**:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
    <title>MusicDot</title>
</head>
<body>
    <header>
        <!-- Contenido del header -->
        <img src="img/logo-musicdot.png" alt="Logo de MusicDot">
    </header>
    <main>
        <!-- Contenido principal -->
    </main>
    <footer>
        <!-- Contenido del footer -->
    </footer>
</body>
</html>
```

Ahora con los links necesitamos notar que son links que van para otras páginas dentro del nuestro propio sitio, por tanto eses 3 **links** hacen parte de una **navegación** y son 3 **links** en secuencia. Cuando tenemos elementos iguales en secuencia tenemos una **lista**. En nuestro caso aquí el orden de los links no importa:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
    <title>MusicDot</title>
</head>
<body>
    <header>
        <!-- Contenido del header -->
        <img src="img/logo-musicdot.png" alt="Logo de la MusicDot">
        <nav>
            <ul>
                <li><a href="#">Contacto</a></li>
                <li><a href="#">Entrar</a></li>
                <li><a href="#">Matriculate</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <!-- Contenido principal -->
    </main>
    <footer>
        <!-- Contenido del footer -->
    </footer>
</body>
</html>
```

El próximo paso sería hacer la profundización de la otra etiqueta y así sucesivamente. Recuerde que eso es apenas una recomendación.

Capítulo 15

# ESTILIZANDO EL HEADER DE LA PAGINA HOME

## 15.1 CSS RESET

Cuando no especificamos ningún estilo para nuestros elementos del HTML, el navegador utiliza una serie de estilos por defecto, que son diferentes en cada uno de los navegadores. En algunos momentos de la construcción de nuestros proyectos webs, podremos enfrentar problemas con cosas que no habíamos previsto; por ejemplo, el espaciado entre los caracteres. En un determinado navegador puede que un texto que, por nuestra definición debería aparecer en 4 líneas, está apareciendo en 5, desconfigurando así todo nuestro *layout*.

Para evitar ese tipo de interferencia, algunos desarrolladores y empresas crearon algunos estilos que llamamos de **CSS Reset**. La intención es definir un valor básico para todas las características del CSS, sobrescribiendo totalmente los estilos por defecto del navegador.

De esta forma, podemos comenzar a estilizar nuestras páginas a partir del mismo punto de partida para todos los casos, lo que nos permite tener un resultado mucho más sólido en varios navegadores.

Existen algunas opciones para resetear los valores del CSS. Algunas que merecen destaque son las siguientes:

**HTML5 Boilerplate**

El HTML5 Boilerplate es un proyecto que pretende ofrecer un excelente punto de partida para quien pretende desarrollar un nuevo proyecto con HTML5. Una serie de técnicas para aumentar la compatibilidad de la nueva tecnología con navegadores un poco más antiguos están presentes y el código es totalmente gratuito. En su archivo "style.css", están reunidas diversas técnicas de CSS Reset. A pesar de ser consistentes, algunas de esas técnicas son un poco complejas, pero es un punto de partida que podemos considerar.

https://html5boilerplate.com/

**YUI3 CSS Reset**

Creado por los desarrolladores web de Yahoo, una de las referencias en el área, ese CSS Reset está compuesto de 3 archivos distintos. El primero de ellos, llamado de **Reset**, simplemente cambia todos los

valores posibles para un valor patrón, donde hasta las etiquetas `<h1>` y `<small>` pasan a ser exhibidos con el mismo tamaño. El segundo archivo es llamado de **Base**, donde algunos márgenes y dimensiones de los elementos son estandarizados. El tercero es llamado de **Font**, donde el tamaño de los tipos es definido para que tengamos un visual consistente inclusive en diversos dispositivos móviles.

**Eric Meyer CSS Reset**

Hay también el famoso CSS Reset de Eric Meyer, que puede ser obtenido en http://meyerweb.com/eric/tools/css/reset/. Es apenas un archivo con tamaño bien reducido.

Vale recordar que el uso de cada **reset** varía conforme la necesidad. Algunos CSS Resets son más *agresivos* que los otros, y también es importante saber que ellos pueden ser modificados para tus propias necesidades. Existen personas que desarrollan sus propios CSS Resets e inclusive acostumbran compartir sus códigos en ciertos foros sobre `HTML` y `CSS`.

## 15.2 FUENTES PROPIAS

Es común que páginas en la web tengan tipografías que combinen con su estética y lenguaje visual, también para la facilidad de lectura. Solo que no siempre los usuarios poseen las fuentes que queremos usar en nuestras páginas. Para eso necesitamos decidir cómo hacer para que nuestros usuarios tengan acceso a esas fuentes. Una forma muy común y fácil es usar ***Google Fonts***. Basta entrar en el sitio https://fonts.google.com/, escoger una tipografía y después escoger cómo importar las fuentes. La primera forma de importar las fuentes de ***Google Fonts*** es usando la etiqueta `<link>` y pasar la referencia para el ***Google Fonts***. No se preocupe que a la hora de escoger una fuente, el propio Google proporciona el código listo para utilizar:

```
<link href="https://fonts.googleapis.com/css?family=Roboto&display=swap" rel="stylesheet">
```

> Lo ideal es hacer esa importación antes de cualquier archivo CSS para garantizar que todos los archivos siguientes van a conseguir utilizar dichas fuentes.

La otra forma de importar es haciendo un ***@import*** en el proprio archivo CSS que va a usar las fuentes:

```
@import url('https://fonts.googleapis.com/css?family=Roboto&display=swap');

body {
    font-family: 'Roboto', sans-serif;
}
```

Por último una tercera forma de importar fuentes sin depender de servicios externos es importar el propio archivo de fuentes en el archivo CSS, usando el ***@font-face***:

```
@font-face {
  font-family: miFuente;
  src: url(fonts/mi-fuente-personalizada.woff);
}

body {
    font-family: 'miFuente', sans-serif;
}
```

Una observación muy importante para cuando vayamos a utilizar fuentes de la web:

Antes de usar cualquier fuentes verifique los derechos de autor de la misma y vea si es necesario solicitar algún permiso o comprar los derchos para usar la fuente. La ventaja de se usar ***Google Fonts*** es que todas las fuentes son abiertas para uso libre, pero en el caso de otras fuentes es bueno verificar antes. Lo que no queremos es el uso indebido de fuentes.

## 15.3 MODULARIZANDO COMPONENTES CON CSS ISOLADOS

Durante el desarrollo del proyecto, principalmente en la parte de planificación, definimos diversas secciones que van a englobar los diversos contenidos de nuestra página que pueden o no repetirse en otras páginas de nuestro sitio. Podemos lidiar con la situación de diversas formas:

### 1) CSS General con CSS Específico de la Página

El abordaje de crear un CSS general con un CSS específico de la página es bien conocida y muy utilizada en el mercado. La idea es crear un CSS que va a contener estilos que pueden repetirse en diversas páginas, como por ejemplo, tipografía, colores, tamaños y hasta algunos componentes, y después crear un CSS que va a contener estilos específicos en aquella página. Como todo en la vida, existen ventajas y desventajas de ese abordaje.

- Ventajas:
  - Sólo es necesario la importación de un archivo CSS para que la página ya tenga un estilo patrón.
  - Como todas las clases de estilos están en un solo lugar, podemos escribir el `html` ya colocando los nombres de clases que necesitamos. Casi como un *framework*.
- Desventajas:
  - Todas las páginas tendrán que cargar un archivo de estilos gigantesco independiente si van a usar todas las clases o no, lo que puede impactar en el desempeño.
  - Dependiendo del tamaño del archivo general puede ser muy complejo encontrar selectores que queremos usar.
  - Si por algún motivo queremos usar algo que era para ser exclusivo de una página en otra página,

tendremos que colocarlo en el archivo general lo que puede desordenar el archivo general.
    - El mantenimiento puede ser complejo dependiendo del tamaño del archivo.

## Un CSS Para Cada Componente De La Página

Este abordaje también es bastante utilizado en el mercado, solo que es más utilizado en proyectos con el uso de frameworks (React, Angular) y preprocesadores de CSS (SASS). En esta abordaje cada sección o componente de la página tiene un CSS exclusivo.

- Ventajas:
    - Como cada componente tiene su propio CSS solo es necesario importar los componentes que necesitamos usar en cada página, evitando importar estilos innecesarios.
    - Organización y mantenimiento queda menos complicada porque es más claro saber exactamente cuál archivo trabajar.
- Desventajas:
    - Necesitamos importar un archivo CSS diferente para cada componente que queremos usar, lo que puede generar líneas de imports gigantescas.

Capítulo 16

# PROGRESSIVE ENHANCEMENT

El concepto de *progressive enhancement* (perfeccionamiento continuo) define que la construcción de una página parte de una base común y con la garantía de ejecutarse en los más diversos navegadores para después acrecentar pequeñas mejoras incluso cuando estas sólo funcionan en navegadores más modernos.

Si alguna de esas mejoras no es soportada por el navegador, de todas formas el usuario conseguirá acceder el sito web, con el impacto de tener su experiencia reducida.

Este concepto no se aplica uniformemente para todas las páginas y proyectos web. Por lo tanto este punto debe ser pensando aisladamente para la estructura, estilo y comportamiento. Cada punto se comporta diferentemente cuando no es soportado por el navegador.

Personas diferentes van usar nuestro sitio en dispositivos diferentes de los que nosotros usamos para desarrollar, en lugares diferentes y en condiciones muy diferentes de las que desarrollamos el sitio.

## 16.1 CONDICIONES, OPCIONES, LIMITACIONES Y RESTRICCIONES

- Tamaño de pantalla.
    - Personas pueden acceder nuestro sitio de diversos dispositivos con tamaño de pantallas diferentes, por ejemplo, celulares, TVs, tablets, notebooks, entre otros.
- Ausencia de mouse y teclado.
    - No todos los dispositivos del mundo soportan un mouse y un teclado.
- Pointer menos preciso: .
    - Generalmente en la ausencia de un ratón, el ***touchscreen*** se convierte en el apuntador del dispositivo.
- Navegadores diferentes y navegadores en versiones diferentes
    - Navegadores desactualizados soportan menos tecnologías que navegadores más nuevos, y diferentes navegadores muestran páginas de formas diferentes.
- Tipos de conexiones.

    - El 3G y la fibra óptica a 1 GB/s tienen velocidades bien diferentes que impactan como la página puede ser mostrada.
- Personas con Deficiencias.
    - Algunas personas pueden tener dificultades para acceder nuestro sitio como problemas de visión: ceguera, miopía, visión borrosa; motora: incapacidad de usar un mouse, control motor con limitaciones; audición: sordera.
- Personas usando el sitio en situaciones diferentes.
    - En el metro o en autobús lleno y personas sentadas en la oficina o en casa.

Que significa todo esto para quien o quienes están desarrollando el sitio: definir y garantizar que el mismo sea accesible en las condiciones **definidas**. ¿Cómo garantizar eso? Probar en todas las situaciones y modificar el código siempre que la prueba en una dada situación falle. Si no pensamos y probamos en dispositivos y casos de uso en condiciones diferentes o con limitaciones y restricciones, hay grandes posibilidades de que nuestro sitio solo quede utilizable para quien se encuadre en el perfil probado. ¿Y si no da para probar en todas las situaciones? ¿Como intentar "reducir" o estandarizar el esfuerzo? Intentar seguir un flujo de desarrollo (o pensamiento) que "automáticamente" incluya la mayor parte de las situaciones:

- En relación al espacio de pantalla: analogía de la caja de fósforo versus caja de zapato. ¿Lo que cabe en una caja de zapato cabe en una caja de fósforo? ¿y viceversa?

Debemos pensar y probar **primero** en la base que es igual/mínima para todas las personas y después mejorar/añadir el código para situaciones donde apliquen esas mejoras. A continuación te sugerimos un orden de desarrollo, el por qué de cada paso y como probar:

1. **Contenido**: contenido es lo que todas las personas quieren ver en un sitio y se debe comenzar siempre por aquí.
    - ¿Cual tipo de contenido la página va a tener? ¿Como el contenido va a ser agrupado? ¿Dónde va cada contenido? ¿Cuál es la cantidad de contenido en cada lugar?
    - Pruebas: revisión de contenido, ortografía, etc.
2. **Semántica con HTML**: semántica es una mejora al contenido que está sin marcación.
    - ¿Cómo ubicarse entre 1000 líneas de contenido para marcar y dar mantenimiento a ese contenido?
    - Prueba: ¿la etiqueta escogida (ejemplo: `<footer>` ) mejora la localización del código y la legibilidad para quien está desarrollando?
    - ¿Cómo usar el sitio sin acceso al visual? Muchas personas dependen sólo del contenido para navegar en el sitio. Esas personas usan **lectores de pantalla** que interpretan la página y la dejan accesible y navegable de una forma similar a la forma como se ubica en el código (por las etiquetas), para quienes desarrollan código. Lectores de pantalla permiten que una persona lea directamente el contenido del `<footer>` en vez de leer todo el contenido de la página del inicio

hasta el fin para llegar en el contenido del `<footer>` .

- Prueba: definir casos de uso, abrir y usar el sitio de acuerdo en un lector de pantalla
- Otra prueba: programas que exhiben el árbol de accesibilidad (ejemplo: Dev Tools del Firefox)

3. **CSS**: estilos harán que el contenido sea exhibido de una forma **mejor**. El foco aún es el contenido, entonces estilos son el tercer paso de **mejora**.

   - Un estilo debe mejorar y mantener el contenido siempre accesible en cualquier situación.
   - Recomendamos comenzar limitando la anchura y/o la altura del *viewport*. Eso hace que el CSS no impida el acceso al contenido en pantallas menores y el contenido que está en una pantalla menor acaba siendo accesible también en pantallas mayores.
   - Probar en diversas versiones de navegadores.

A la hora de escoger cuál código HTML y CSS escribir. El W3C, el Progressive Enhancement y los navegadores desactualizados recomiendan:

- ¿Las etiquetas escogidas son identificadas por los *User Agents* ? Si la etiqueta es nueva y no existe aún en aquella versión de navegador, ¿que pasa?. Etiquetas nuevas son mejoras y en navegadores desactualizados se convierten en `<div>` . El contenido no va dejar de ser exhibido.

- Propiedades y valores nuevos del CSS (#rrggbbaa) en navegadores desactualizados son desconsiderados y la etiqueta sigue en frente siendo exhibida en la página.

- El sitio no va a parar de funcionar. Cosas antiguas no dejan de existir o de funcionar. Este concepto es también utilizado para actualizar el HTML, el CSS y otros patrones a cada nueva versión. Nuevas versiones generalmente no cambian lo que había antes, mejoran.

- WCAG: tamaño de fuentes, contraste, legibilidad, etc.

# DISPLAY FLEX

Vimos algunas formas de manipular posicionamiento de elementos como `display: inline/block/inline-block`, `margin` y `text-align`, pero todas esas formas son muy "**rígidas**" a la hora de distribuir elementos en la página. Nosotros conseguimos posicionar con una cierta precisión los elementos en la pantalla, pero ese posicionamiento acaba siendo poco flexible, en el sentido de que si el *container* o la pantalla cambian de tamaño, los elementos no van a tener su posicionamiento adaptado. Todo necesita ser siempre calculado con precisión para que el restante de los elementos dispuestos en la pantalla no sean afectados por las cambios de dimensiones.

Pensando en esa flexibilidad de posicionamiento, los desarrolladores crearon un nuevo tipo de `display`, el `display: flex;`.

## 17.1 FLEX CONTAINER

El `display: flex` funciona de una forma diferente de los otros displays. Cuando colocamos esa propiedad en un elemento, ese elemento se convierte en un ***flex container***, a partir de allí podemos manipular todos los elementos hijos (***flex items***) de ese ***flex container*** con propiedades nuevas. Esas propiedades deben ser usadas en el elemento que es un ***flex container***.

> Por defecto, cuando aplicamos `display: flex` para un elemento, automáticamente todos elementos hijos quedan un al lado del otro como si estuviesen sobre el efecto de `display: inline`.

### Propiedades de un Flex Container

- `justify-content`:

  Esa propiedad ajusta horizontalmente los elementos hijos del ***flex container***

  - **flex-start**: Es el valor estándar. Los elementos quedan pegados uno al lado del otro a la izquierda del flex container.
  - **flex-end**: Los elementos quedan pegados uno al lado del otro a la derecha del *flex container*.

- **center**: Los elementos quedan pegados uno al lado del otro en el medio del *flex container*.
- **space-between**: El primer elemento queda totalmente a la izquierda del *flex container* y el último queda totalmente a la derecha. El resto de los elementos quedan distribuidos con un espaciado igual entre ellos.
- **space-around**: Cada elemento queda con un espaciado igual **en vuelta** del mismo. Eso quiere decir que el primer elemento va a tener un espaciado mayor a la derecha del que a la izquierda porque va a sumar con el espaciado a la izquierda del segundo elemento.
- **space-evenly**: Corrige el "problema" del valor de arriba. Los elementos tendrán un espaciamiento igual en ambos lados.

- `align-items`:

  Esa propiedad ajusta verticalmente los elementos hijos del ***flex container***

  - **stretch**: Es el valor estándar. Los elementos se "estiran" para que todos queden con la misma altura.
  - **flex-start**: Los elementos quedan todos alineados con la borde superior del *flex container*.
  - **flex-end**: Los elementos quedan todos alineados con la base del *flex container*.
  - **center**: Los elementos quedan todos alineados con el medio del *flex container*.
  - **baseline**: Los elementos quedan alineados con base del contenido textual de cada uno de ellos.

- `flex-wrap`:

  Esa propiedad trabaja con la "quiebra de línea" de los elementos en línea.

  - **nowrap**: Es el valor estándar. Los elementos van quedar uno al lado del otro, inclusive cuando no exista más espacio horizontal.
  - **wrap**: Los elementos que no caben más en el espacio lateral reciben una quiebra de línea, o sea, van para la línea de abajo.
  - **wrap-reverse**: Los elementos que no caben más en el espacio lateral reciben una quiebra de línea de arriba, o sea, van para la línea de arriba.

Con el *flexbox*, alinear un elemento verticalmente y horizontalmente y de forma proporcional y responsiva se convirtió en una tarea muy fácil.

También es posible aplicar propiedades para los ***flex items***, en el blog CSS Tricks existe un guía completo y visual con los efectos de cada propiedad de un *flexbox*: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Capítulo 18

# RESPONSIVIDAD Y FALLBACK

En el día a día de desarrollo de páginas Web y aplicaciones, siempre acabamos encontrando casos de incompatibilidad de propiedades CSS con los navegadores que usamos. No todas las personas actualizan sus navegadores, sea por no saber actualizar, por una característica específica de aquella versión, compatibilidad con el sistema operacional, etc. Por lo tanto, algunas propiedades CSS que usamos no van a funcionar en todos los navegadores.

El sitio caniuse.com muestra un gráfico sobre compatibilidad de propiedades CSS con diversas versiones de navegadores, y la idea de usar este sitio es por cuenta de las métricas que nos presenta. Una de esas métricas es la cantidad de usuarios utilizando versiones diferentes de navegadores.

Vamos a montar nuestro sitio pensando en la mayoría de los usuarios (que acostumbran usar versiones más actualizadas de navegadores) pero no olvidando los pequeños porcentajes que usan navegadores más antiguos (IE 10 por ejemplo). Solo que no podemos dejar de colocar nuevas propiedades y etiquetas por cuenta del pequeño porcentaje de personas que usan navegadores muy antiguos. ¿Qué hacemos entonces?

Veamos el siguiente ejemplo:

```
.mi-elemento {
    background-color: #f00;
}
```

En el ejemplo colocamos un color de fondo rojo en un elemento que posee la clase `mi-elemento`. Ahora vamos observar este otro ejemplo:

```
.mi-elemento {
    background-color: #f00;
    background-color: #0f0;
}
```

Ahora el navegador va a leer primero el color rojo y después va sustituir por el color verde porque estamos usando la misma propiedad en el mismo elemento, o sea, quien fue declarado por último, gana la preferencia. Ahora vamos a ver un ejemplo más:

```
.mi-elemento {
    background-color: #f00;
    background-color: (10);
}
```

La función `colordestacado()` no es un CSS válido por lo tanto el navegador va a leer el color rojo

primero y después va intentar leer la función, pero como esa función no existe, el navegador ignora la sobrescrita y va a mantener el color de fondo rojo.

Esto puede parecer un error de código, pero en la verdad es una técnica llamadas de *fallback*, donde caso una propiedad no pueda ser interpretada, otra puede asumir su lugar. Esa es la forma más ideal de mantener la compatibilidad con navegadores más antiguos. Siguiendo el concepto de *Progressive Enhancement*, comenzamos a colocar nuestras propiedades basadas en los navegadores más antiguos y después vamos creciendo para otras propiedades más actualizada de navegadores más nuevos, así naturalmente vamos dejando nuestro sitio responsivo y compatible con diversos navegadores.

A la hora de decidir cuál propiedad antigua será usada como "base" del perfeccionamiento continuo, es importante verificar si dicha propiedad todavía vale la pena para la época en que el sitio está siendo creado.

No vamos usar propiedades del *Internet Explorer 6* porque ya es un navegador extremadamente antiguo y que no presenta más paquetes de seguridad para el uso en el internet. Pero podemos considerar el *Internet Explorer 11* que aún posee una cantidad de usuarios considerable.

Capítulo 19

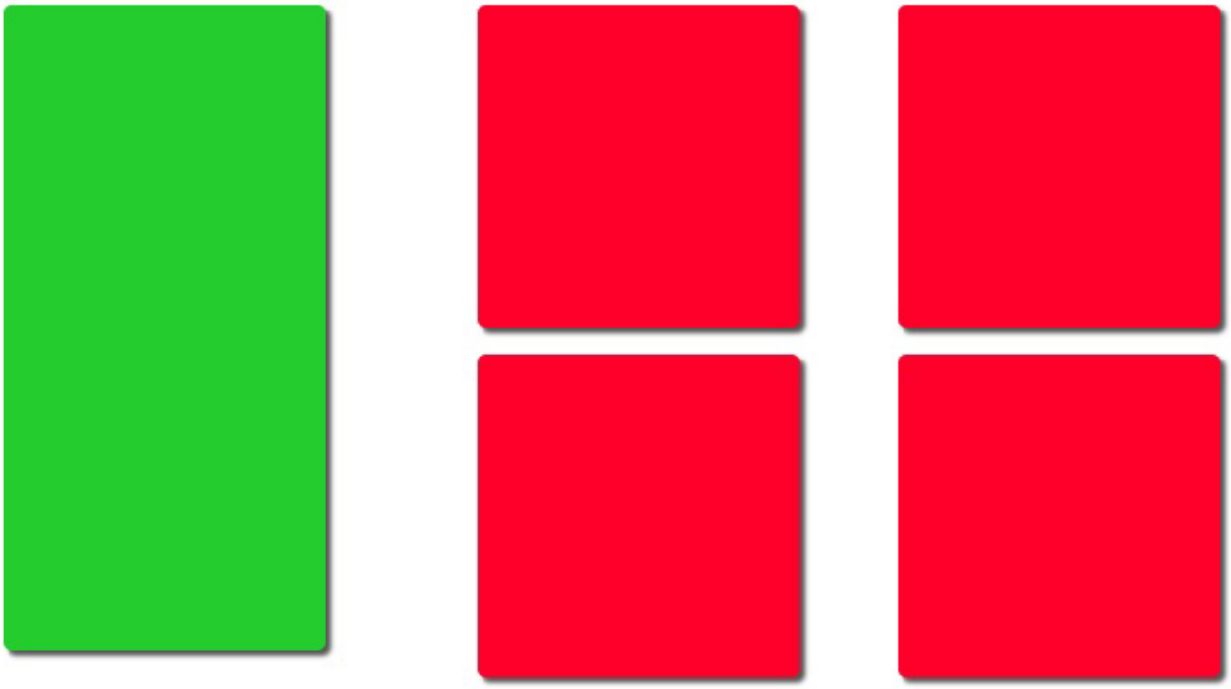
# DISPLAY: GRID

La propiedad `display: flex` ya nos proporcionó grandes ventajas en relación a las formas que acostumbrábamos manipular posicionamiento de elementos, solo que encontramos una cierta complicación cuando necesitamos posicionar elementos de forma *bidimensional*. `Flex` es una óptima herramienta para cuando tenemos elementos que necesiten ser distribuidos de forma igual y con una dirección bien definida.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
    <title>Ejemplo</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <main class="flex-container">
        <div class="foto"></div>
        <div class="foto"></div>
        <div class="foto"></div>
        <div class="foto"></div>
        <div class="foto"></div>
        <div class="foto"></div>
        <div class="foto"></div>
        <div class="foto"></div>
    </main>
</body>
</html>
```

```
.flex-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-evenly;
}

.foto {
    width: 200px;
    height: 200px;

    border-radius: 5px;

    box-shadow: 3px 3px 3px #000000aa;

    background-color: #ff002b;

    margin-bottom: 1rem;
}
```

En el ejemplo de arriba el `display: flex` funciona perfectamente porque la distribución es unidireccional, aunque exista una segunda línea para el `flex-wrap`. Pero si ahora queremos hacer algo como:

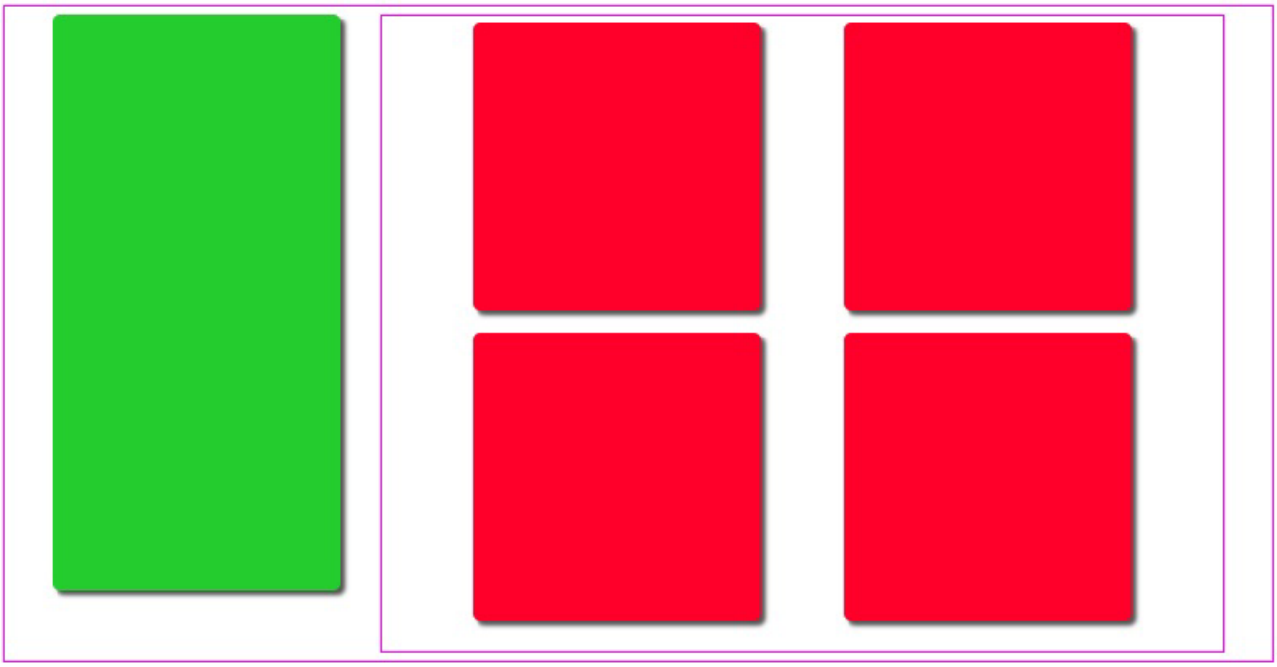


`display: flex` no consigue tratar el posicionamiento *bidimensional*. Necesitamos cambiar la estructura para que los elementos *ocupen el espacio de una línea*:

```
<main>
    <div class="flex-container"> <!-- Primera línea que contiene "apenas" el bloque verde y el contai
ner que guarda los elementos rojos-->
        <div class="foto foto-destaque-verde"></div> <!-- Bloque verde y el primer elemento de la lín
ea -->
        <div class="flex-container foto-container"> <!-- Container que va a guardar el restante de lo
s elementos rojos y el segundo elemento de la línea -->
            <div class="foto"></div>
            <div class="foto"></div>
            <div class="foto"></div>
            <div class="foto"></div>
        </div>
    </div>
</main>
```

```
.flex-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-evenly;
}

.foto-container {
    width: 66%;
}
```

```css
.foto {
    width: 200px;
    height: 200px;

    border-radius: 5px;

    box-shadow: 3px 3px 3px #000000aa;

    background-color: #ff002b;

    margin-bottom: 1rem;
}

.foto-destaque-verde {
    background-color: #24cc2d;

    height: 400px;
}
```

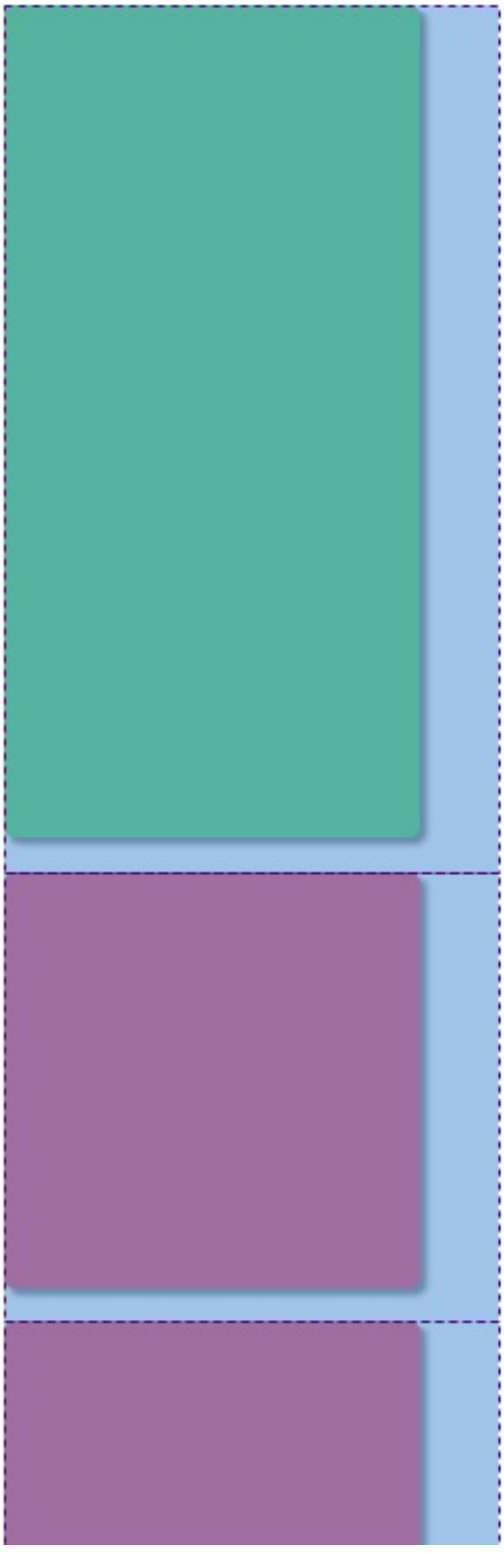


Vamos a entender qué pasó. Como el `flex` solo consigue trabajar con una dirección, entonces tuvimos que crear la primera línea conteniendo apenas dos elementos: el **bloque verde** y una **caja vacía**. En esa caja vacía, creamos nuestra segunda línea (dejamos la caja como un `flex container` ) con los 4 bloques rojos dentro y limitamos el tamaño de esa caja para que el espacio sea mejor dividido. Con este ejemplo ya conseguimos percibir que un simple código comenzó a quedar muy complicado. Imagine hacer *layouts* más complejos usando esa técnica del `diplay: flex` . Por cuenta de esa dificultad de hacer *layouts* con una complejidad *bidimensional* es que surgió el `display: grid` .

La idea del `grid` es exactamente de no necesitar cambiar la estructura del **HTML** y que toda la

parte de posicionamiento de *layout* quede apenas en el **CSS**.

Cuando colocamos la propiedad `display: grid` en un elemento, el se transforma en un *grid container* de la misma forma que un elemento se convierte en *flex container* cuando recibe la propiedad `display: flex` . Después que definimos un *grid container* necesitamos definir su estructura. Por defecto un *grid container* coloca cada ítem en una línea y todos en la misma columna:



Para ver mejor las marcaciones de grid, use la herramienta de desarrollo de su navegador.

Sabemos que necesitamos de 3 columnas y 2 líneas para conseguir el efecto que queremos. Entonces vamos a usar las propiedades de *grid container* que van a definir el número de columnas y el número de líneas respectivamente: `grid-template-columns:` y `grid-template-rows` . La propiedad de `grid` permite que usemos otra unidad de medida: `fr` (que significa fracción).

## 19.1 GRID-TEMPLATE-COLUMNS

Podemos colocar infinitos valores dentro de esa propiedad, pero cada valor dentro de esa propiedad va representar **una columna**. Queremos 3 columnas que ocupen el espacio de la página equitativamente. Vamos a usar la nueva unidad de medida `fr` .

```
.grid-container {
    display: grid;

    grid-template-columns: 1fr 1fr 1fr;
}
```

En el código de arriba, cuando escribimos `1fr 1fr 1fr` estamos diciendo que queremos **3** columnas que ocupen 1 fracción del espacio disponible, o sea, cada columna va a tener 1/3 del espacio disponible lateralmente.

## 19.2 GRID-TEMPLATE-ROWS

La propiedad que crea las líneas funciona de la misma forma que la propiedad que crea columnas. Por lo tanto, si queremos dos líneas que ocupen el mismo espacio:

```
.grid-container {
    display: grid;

    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr;
}
```

Solo que después de haber colocado las dos líneas aún así los elementos no quedan distribuidos correctamente. El elemento verde aún ocupa apenas una línea y no dos, entonces necesitamos indicarle la cantidad de filas con `grid-row` , en este caso 2:

```
.foto-destaque-verde {
    background-color: #24cc2d;

    grid-row: span 2;

    height: 400px;
}
```

La propiedad `grid-row` es usada únicamente en los elementos hijos de un *grid container* y esta recibe dos valores: fila de inicio y fila de fin. En nuestro ejemplo usamos el valor `span` , que dice que queremos mezclar líneas, y el `2` , la cantidad de líneas que vamos a mezclar. Cuando usamos el `span`

el próximo valor va a ser la cantidad de líneas o columnas que vamos mezclar, por lo tanto, no indicamos cuál línea es la línea de fin. Cuando no colocamos el valor de fin, el navegador coloca automáticamente `span 1` que quiere decir "ocupe apenas 1 línea/columna".

Ahora sí, nuestro bloque verde está en la posición correcta en relación a los elementos rojos. Usando `grid` no necesitamos cambiar la estructura HTML para conseguir el efecto que queríamos en el comienzo y el código quedó mucho más simple de entender.

Vamos a incrementar la dificultad a nuestro ejemplo. Vamos a cambiar la posición del bloque verde para la derecha. Para ello necesitamos usar la propiedad `grid-columns` para decir donde queremos que nuestro bloque verde comience y termine, pero apenas colocar esa propiedad en el bloque verde no es suficiente, necesitamos colocar esa propiedad también en los bloques rojos y eso una vez más ya comienza a aumentar significativamente la complejidad y dificultad de mantenimiento.

Para mejorar esa capacidad de movimiento de elementos dentro del ***grid***, fue creada una propiedad llamadas ***grid-template-areas***, que nombra cada espacio del *grid* creado. Vamos a seleccionar nuestro *layout* normal y tratar de nombrarlo basándonos en lo que ya tenemos:

```
.grid-container {
    display: grid;

    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr;
    grid-template-areas:
        "verde rojo rojo"
        "verde rojo rojo";
}
```

Como vemos en el ejemplo de arriba en la primera línea (el primer conjunto de comillas): `primer espacio: bloque verde, segundo espacio: bloque rojo, tercer espacio: bloque rojo` ; segunda línea (el segundo conjunto de comillas): `primer espacio: bloque verde, segundo espacio: bloque rojo, tercer espacio: bloque rojo` . Ahora solo necesitamos decir quien es el bloque rojo y quien es el bloque verde:

```
.grid-container {
    display: grid;

    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr;
    grid-template-areas:
        "verde rojo rojo"
        "verde rojo rojo";
}

.foto-destaque-verde {
    background-color: #24cc2d;

    /*
    grid-column: 4;
    grid-row: span 2;
    Este código no es más necesario por cuenta del grid-template-area
```

```
    */
    grid-area: verde;

    height: 400px;
}
```

Antes de colocar nombre en los otros bloques, vamos a nombrar apenas el bloque verde y después probar. Podemos observar que los elementos continúan en la misma posición. Ahora vamos a cambiar la posición del bloque verde:

```
.grid-container {
    display: grid;

    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr;
    grid-template-areas:
        "rojo rojo verde"
        "rojo rojo verde";
}

.foto-destaque-verde {
    background-color: #24cc2d;

    /*
    grid-column: 4;
    grid-row: span 2;
    Este código no es más necesario por cuenta del grid-template-area
    */
    grid-area: verde;

    height: 400px;
}
```

Incluso sin nombrar los bloques rojos con la propiedad `grid-area` llegamos a nuestro objetivo. En este caso solo queremos cambiar la posición del bloque verde y queremos que el restante se adapte conforme necesario, entonces podemos cambiar los valores de **rojo** para apenas un punto "**.**" .

```
.grid-container {
    display: grid;

    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr;
    grid-template-areas:
        ". . verde"
        ". . verde";
}

.foto-destaque-verde {
    background-color: #24cc2d;

    /*
    grid-column: 4;
    grid-row: span 2;
    Este código no es más necesario por cuenta del grid-template-area
    */
    grid-area: verde;

    height: 400px;
}
```

No podemos dejar espacios vacíos que el navegador no va entender, necesitamos dejar alguna cosa para indicar el espacio del ***GRID***.

Capítulo 20

# BOOTSTRAP Y FRAMEWORKS DE CSS

Una tendencia en alta en el mundo front-end es el uso de frameworks CSS con estilos base para nuestra página. En vez de comenzar todo el proyecto desde cero, creando todo los estilos manualmente, existen frameworks que ya traen toda una base construida de donde podemos iniciar nuestra aplicación.

Existen muchas opciones, pero el **Bootstrap** tal vez sea el de mayor notoriedad. Fue creado por funcionarios de Twitter a partir de códigos que ellos ya usaban internamente. Fue liberado como opensource y ganó muchos adeptos. El proyecto creció bastante en madurez e importancia en el mercado a punto de desvincularse de Twitter y ser apenas **Bootstrap**.

https://getbootstrap.com/

Bootstrap trae una serie de recursos:

- Reset CSS
- Estilo visual base para mayoría de las etiquetas
- Íconos
- Grids listos para uso
- Componentes CSS
- Plugins JavaScript
- Todo responsivo y mobile-first

## 20.1 ESTILO Y COMPONENTES BASE

Para usar **Bootstrap**, solo es necesario incluir su CSS en la página:

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

Simplemente eso ya nos trae una serie de beneficios. Un reset es aplicado, y nuestras etiquetas ganan estilo y tipografía base. Eso quiere decir que podemos usar etiquetas como un `<h1>` o un `<p>` y estas tendrán un estilo característico de **Bootstrap**.

Aparte de eso, también son proporcionadas **muchas clases** con componentes adicionales que podemos aplicar en la página. Son varias opciones. Por ejemplo, para crear un título con una frase de apertura en destaque, usamos la clase **`jumbotron`** :

```
<div class="jumbotron jumbotron-fluid">
```

```
    <div class="container">
        <h1 class="display-4">Excelente, el primer paso fue dado</h1>
        <p class="lead">¡Gracias por matricularte en MusicDot!</p>
    </div>
</div>
```

**Bootstrap** utiliza la idea de reutilización de clases que vimos en capítulos anteriores para estilizar páginas. En el ejemplo de arriba, para crear un componente del tipo **jumbotron**, solamente necesitamos crear un elemento y llamar a la clase que representa ese componente. Eso facilita mucho a la hora de crear páginas desde cero, porque si la persona ya está acostumbrada con la nomenclatura y conoce bien los componentes de **Bootstrap**, es posible escribir el HTML con los nombres de clases correctos. Entonces se reduce el tiempo casi por la mitad a la hora de desarrollar, porque no necesitamos más, enfocarnos tanto en las propiedades del CSS y si en la estructura. En la documentación del **Bootstrap** existen varios ejemplos de estructuras que podemos literalmente copiar y pegar y alterar para el contenido que necesitamos.

*Curiosidad: jumbotron es una palabra en inglês que significa "pantalla gigante".*

> La recomendación para el uso del Bootstrap (principalmente para personas nuevas con el framework) es dejar una pestaña abierta del navegador con la documentación para conferir los componentes, estructuras y herramientas en cuanto se escribe la estructura.
>
> Podemos hacer nuestra propia adaptación del CSS de Bootstrap. Basta sobrescribir las clases que queremos usar en un archivo separado y hacer el import después del CSS de Bootstrap.

Algunos componentes de **Bootstrap** poseen interactividad con el usuario a través de JavaScript, entonces a parte de importar el archivo CSS necesitamos importar también el archivo de JavaScript. Solo que para que el archivo de JavaScript de **Bootstrap** funcione, se necesita de otro framework llamado *jQuery*.

https://jquery.com/

La importación del archivo de JavaScript es hecha antes del cierre del etiqueta `</body>` y el import del *jQuery* debe venir antes del archivo del **Bootstrap**:

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

Capítulo 21

# UN POCO DE LA HISTORIA DE JAVASCRIPT

Al inicio de la Web, las páginas eran poco o nada interactivas, eran documentos que presentaban su contenido exactamente como fueron creados para ser exhibidos en el navegador. Existían algunas tecnologías para la generación de páginas en el lado del servidor, pero habían limitaciones respecto a cómo el usuario consumía aquel contenido. Navegar a través de *links* y enviar informaciones a través de formularios era básicamente todo lo que se podía hacer.

## 21.1 HISTORIA

Viendo el potencial de la Web en el Internet para el público general y la necesidad de una interacción mayor del usuario con las páginas, Netscape, creó el navegador más popular de inicio de los años 90, llamado con el mismo nombre. Creó también *Livescript*, un lenguaje simple que permitía la ejecución de *scripts* contenidos en las páginas dentro del propio navegador.

Aprovechando el inminente suceso del Java, que venía conquistando cada vez más espacio en el mercado de desarrollo de aplicaciones corporativas, Netscape rebautizó *Livescript* como JavaScript en un acuerdo con SUN (hoy adquirida por la Oracle) para impulsar el uso de las dos. El entonces vice-líder de los navegadores, Microsoft, añadió a Internet Explorer el soporte a *scripts* escritos en VBScript y creó su propia versión de JavaScript, conocido como JScript.

JavaScript es el lenguaje de programación más popular en el desarrollo Web. Soportado por todos los navegadores, el lenguaje es responsable por prácticamente cualquier tipo de dinamismo que queramos hacer en nuestras páginas.

Si usamos todo el poder que este lenguaje tiene para ofrecer, podemos llegar a resultados impresionantes. Excelentes ejemplos de eso son aplicaciones Web complejas como Gmail, Google Maps y Google Docs.

## 21.2 CARACTERÍSTICAS DEL LENGUAJE

JavaScript, como el propio nombre sugiere, es un lenguaje de *scripting*. Un lenguaje de *scripting* es comúnmente definida como un lenguaje de programación que permite al programador controlar una o más aplicaciones de terceros. En el caso del JavaScript, podemos controlar algunos comportamientos de

los navegadores a través de trechos de código que son integrados en la página HTML.

Otra característica común en los lenguajes de *scripting* es que normalmente son lenguajes **interpretados**, o sea, no dependen de compilación para ser ejecutadas. Esa característica está presente en JavaScript: el código es interpretado y ejecutado conforme es leído por el navegador, línea a línea, así como HTML.

JavaScript también posee **gran tolerancia a errores**, una vez que las conversiones automáticas son realizadas durante las operaciones. Como será visto en el recorrer de las explicaciones, no siempre esas conversiones resultan en algo esperado, lo que puede ser fuente de muchos problemas/bugs, caso no conozcamos bien ese mecanismo.

El script programado es enviado en conjunto con el HTML para el navegador, pero ¿cómo el navegador sabrá diferenciar el script de un código html? Para que esa diferenciación sea posible, es necesario colocar el script dentro de la etiqueta `<script>` .

## 21.3 CONSOLA DEL NAVEGADOR

Existen varias formas de ejecutar códigos JavaScript en una página. Una de ellas es ejecutar códigos en el que llamamos de **Consola**. La mayoría de los navegadores desktop ya viene con esa herramienta de fábrica.

La Consola JavaScript exhibe diagnósticos del código y de la página abierta, interactúa con el código JavaScript de la página, y ejecuta cualquier JavaScript digitado.

En el Chrome, por ejemplo, es posible acceder a la Consola apretando **F12** y en seguida acceder la pestaña "Console" o directamente por el atajo de teclado **control + shift + j**, y en el Firefox por el atajo **control + shift + k**.

**DEVELOPER TOOLS**

La consola hace parte de una serie de herramientas embutidas en los navegadores específicamente para nosotros que estamos desarrollando un sitio. Esa serie de herramientas es lo que llamamos de Developer Tools, o apenas de Dev Tools. Más instrucciones em: https://developers.google.com/web/tools/chrome-devtools/

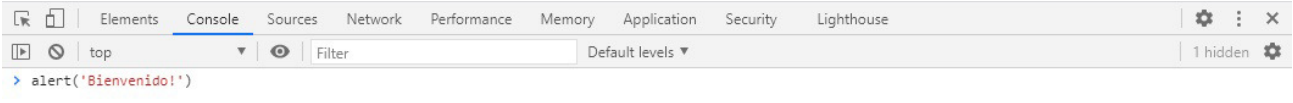



Figura 21.1: #

# 21.4 SINTAXIS BÁSICA DE JAVASCRIPT

## Operadores

Podemos sumar, substraer, multiplicar y dividir como en cualquier lenguaje:

Pruebe hacer algunos cálculos escribiendo directamente en la consola:

```
> 12 + 13
  25
> 14 * 3
  42
> 10 - 4
  6
> 25 / 5
  5
> 23 % 2
  1
```

## Variables

Para almacenar un valor para uso posterior, podemos crear una **variable**:

```
> var resultado = 102 / 17;
undefined
```

En el ejemplo de arriba, guardamos el resultado de `102 / 17` en la variable `resultado`. El comportamiento patrón al crear una variable en la consola es recibir el mensaje: **undefined**. Para obtener el valor que guardamos en ella o cambiar su valor, escribimos su nombre en la consola, por ejemplo:

```
> resultado
6

> resultado = resultado + 10
16

> resultado
16
```

También podemos alterar el valor de una variable, reasignando nuevos valores, por ejemplo, usando las operaciones básicas con una sintaxis bien compacta:

```
> var edad= 10; // undefined
> edad+= 10; // edad vale 20
> edad-= 5; // edad vale 15
> edad/= 3; // edad vale 5
> edad*= 10; // edad vale 50
```

## Number

Con este tipo de dato es posible ejecutar todas las operaciones que vimos anteriormente:

```
var pi = 3.14159;
var radio = 20;
```

```
var perimetro = 2 * pi * radio
```

*Obs: en JavaScript los números decimales son declarados con punto, pues usa el estándar americano.*

### String

No solo números podemos guardar en una variable. JavaScript tiene varios tipos de datos. Una string en JavaScript es utilizada para almacenar trechos de texto:

```
var empresa = "Alura";
```

Para exhibir el valor de la variable empresa fuera del consola, podemos ejecutar el siguiente código:

```
alert(empresa);
```

La función `alert` sirve para la creación de "ventanas" del navegador (**popup**) con algún **contenido de texto** que colocamos dentro de los paréntesis. ¿Qué pasa con el siguiente código?

```
var numero = 30;
alert(numero)
```

El número 30 es exhibido en un **popup**. Cualquier variable puede ser usada en el `alert` . JavaScript no irá a diferenciar el tipo de datos que está almacenado en una variable, y si necesario, intentará convertir el dato para el tipo deseado, en este caso un valor tipo Number fue convertido para String, y así puede ser exhibido por el alert.

### Automatic semicolon insertion (ASI)

Es posible omitir el punto y coma en el final de cada declaración. La omisión de punto y coma en JavaScript es posible debido al mecanismo llamado *automatic semicolon insertion* (ASI) (inserción de punto y coma automático).

## 21.5 LA ETIQUETA SCRIPT

La consola nos permite probar códigos directamente en el navegador. Pero, no podemos pedir a los usuarios del sitio que siempre abran la consola, copien y peguen un código para luego ser ejecutado.

Para que un código JavaScript sea ejecutado en la apertura de una página, es necesario utilizar la etiqueta `<script>` :

```
<script>
  alert("Hola, Mundo!");
</script>
```

La etiqueta `<script>` puede ser escrita dentro de la etiqueta `<head>` así como en la etiqueta `<body>` , pero debemos quedarnos atentos, porque el código es leído y ejecutado inmediatamente dentro del navegador. Vea las consecuencias de esa aplicación en los dos ejemplos a seguir:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Clase de JS</title>

    <script>
       alert("Hola, Mundo!");
    </script>

  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Lenguaje de programación</h2>
  </body>
</html>
```

Si ejecutamos el script vamos a identificar que el mismo interrumpe el procesamiento de la página. Imagine un script que demore un poco más para ser ejecutado o que exija alguna interacción del usuario como una confirmación. ¿No crees que sería más interesante cargar primero la página completamente antes de su ejecución, por una cuestión de desempeño y experiencia para el usuario?

Para realizar eso, basta remover el script del `<head>` y lo colocamos al final dentro del `<body>`, antes del cierre de la etiqueta `</body>`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Clase de JS</title>
  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Lenguaje de Programación</h2>

    <script>
      alert("Hola, Mundo");
    </script>

  </body>
</html>
```

## 21.6 JAVASCRIPT EN UN ARCHIVO EXTERNO

Si el mismo script fuera utilizado en otra página, ¿cómo hacemos? Imagine tener que reescribir el script toda vez que fuera necesario. Para que eso no ocurra, es posible importar scripts dentro de la página utilizando también la etiqueta `<script>`:

**HTML:**

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

**JS:**

```
alert("Hola, Mundo");
```

Con la separación del script en archivos externos es posible reaprovechar las funcionalidades de una página.

## 21.7 MENSAJES EN LA CONSOLA

Es común querer verificar el valor de alguna variable, el resultado de alguna operación durante la ejecución del código. En estos casos, podríamos usar un `alert()`. Pero, si ese contenido debería ser mostrado solamente para el desarrollador, la consola del navegador puede ser utilizada para imprimir este mensaje:

```
var mensaje = "Hola mundo";
console.log(mensaje);
```

> **IMPRESIÓN DE VARIABLES DIRECTAMENTE DE LA CONSOLA**
>
> Cuando esté con la consola abierta, no es necesario llamar `console.log(nombreDeVariable)`: puede llamar el nombre de la variable directamente que esta será impresa en la consola.

## 21.8 DOM: TU PÁGINA EN EL MUNDO JAVASCRIPT

Para que podamos hacer alteraciones dinámicas en un página con JavaScript, los navegadores proporcionan una estructura de datos que representa cada una de nuestros etiquetas en JavaScript. Esa estructura es llamadas de DOM, o sea, **D**ocument **O**bject **M**odel, traduciendo sería algo como "modelo de objetos del documento". Esa estructura de objetos también puede ser accedida a través de la variable global `document`.

*Obs: El termino "documento" es frecuentemente utilizado en referencia a nuestra página. En el mundo front-end, documento y página son sinónimos.*

## 21.9 QUERYSELECTOR

Antes de alterar nuestra página, necesitamos en primer lugar acceder a través de JavaScript, al elemento que queremos alterar. Como ejemplo, vamos alterar el contenido de un título de la página. Para acceder a este necesitamos hacer:

```
document.querySelector("h1")
```

Ese comando usa los **selectores CSS** para encontrar los elementos en la página. Usamos un selector de nombre de etiqueta pero podríamos haber usado selectores de clase o identificadores también:

```
document.querySelector(".class")
```

```
document.querySelector("#id")
```

## 21.10 ELEMENTO DE LA PÁGINA COMO VARIABLE

Si vas a utilizar varias veces un mismo elemento de la página, es posible salvar el resultado de cualquier `querySelector` en una variable:

```
var titulo = document.querySelector("h1")
```

Ejecutando en la consola, puedes verificar que el elemento correspondiente es seleccionado. Podemos de esta forma manipular su contenido. Podemos ver el contenido textual a través de:

```
titulo.textContent
```

Esta propiedad, también, puede recibir valores y ser modificada:

```
titulo.textContent = "Nuevo título"
```

## 21.11 QUERYSELECTORALL

A veces necesitamos seleccionar varios elementos en la página. Varias etiquetas con la clase `.tarjeta` por ejemplo. Si el retorno esperado debería ser más de un elemento, usamos `querySelectorAll` que devuelve una lista de elementos, esta lista es devuelta en la estructura de un *array*.

```
document.querySelectorAll(".tarjeta")
```

Podemos así acceder a los elementos de esa lista a través de sus respectivos índices (comenzando en cero) usando los corchetes:

```
// primera tarjeta
document.querySelectorAll(".tarjeta")[0]
```

## 21.12 ALTERACIONES EN EL DOM

Al alterar los elementos de la página, el navegador sincroniza los cambios y altera la aplicación en tiempo real.

## 21.13 FUNCIONES Y LOS EVENTOS DEL DOM

A pesar de ser interesante la posibilidad de alterar todo el documento por medio de JavaScript, es muy común que las alteraciones sean hechas cuando el usuario ejecuta alguna acción, como por ejemplo, cambiar el contenido de un botón al hacer clic en el mismo y no cuando la página carga. Sin embargo, por patrón, cualquier código colocado en el `<script>` , como ya vimos anteriormente, es ejecutado ni bien que el navegador lo lee.

Para guardar un código para que sea ejecutado en algún otro momento. Por ejemplo, cuando el usuario hace clic en un botón, es necesario utilizar algunos recursos de JavaScript en el navegador. Para entender mejor vamos a crear una **función**:

```
function muestraAlerta() {
  alert("Funciona");
}
```

Al crear una función, simplemente guardamos lo que estuviera dentro de la función, y ese código guardado solo será ejecutado cuando **llamamos** la función, como en el siguiente ejemplo:

```
function muestraAlerta() {
  alert("Funciona");
}

// haciendo una llamadas para la función muestraAlerta, que será ejecutada en este momento
muestraAlerta()
```

Para llamar a la función **muestraAlerta**, solo necesitamos utilizar el nombre de la función y luego después abrir y cerrar paréntesis.

Ahora, para que esa función sea llamada cuando el usuario hace clic en el botón de nuestra página, necesitamos realizar el siguiente código:

```
function muestraAlerta() {
  alert("Funciona");
}

// obteniendo un elemento a través de un selector de ID
var boton = document.querySelector("#botonEnviar");

boton.onclick = muestraAlerta;
```

Note que primeramente fue necesario seleccionar el botón y después definir el evento `onclick`. Lo que va a ser ejecutado cuando el usuario haga clic es la función `muestraAlerta`. Esa receta será siempre la misma para cualquier código que tenga que ser ejecutado después de alguna acción del usuario en algún elemento. Lo que varia es el elemento a ser seleccionando, el evento que va a ser accionado y cuál función será ejecutada.

## Cuales eventos existen

Existen diversos eventos que pueden ser utilizados en diversos elementos para que la interacción del usuario dispare alguna función:

- **oninput:** cuando un elemento input tiene su valor modificado
- **onclick:** cuando ocurre un clic con el mouse
- **ondblclick:** cuando ocurren dos clics en el mouse
- **onmousemove:** cuando se mueve el puntero del mouse
- **onmousedown:** cuando se mantiene presionado el botón del mouse
- **onmouseup:** cuando se suelta el botón del mouse (útil con los dos de arriba para controlar drag-

and-drop)

- **onkeypress:** cuando se presiona y se suelta una tecla
- **onkeydown:** cuando se presiona una tecla
- **onkeyup:** cuando se suelta una tecla
- **onblur:** cuando un elemento pierde foco
- **onfocus:** cuando un elemento gana foco
- **onchange:** cuando un `input`, `select` o `textarea` tienen su valor alterado
- **onload:** cuando la página es cargada
- **onunload:** cuando la página es cerrada
- **onsubmit:** disparado antes de enviar formulario (útil para realizar validaciones)

Existen también una serie de otros eventos más avanzados que permiten la creación de interacciones para drag-and-drop, y también la posibilidad de creación de eventos personalizados.

## 21.14 FUNCIONES ANÓNIMAS

En el ejercicio anterior indicamos que la función `muestraTamanho` debería ser ejecutada en el momento en que el usuario inserta el tamaño del producto en el `<input type="range">`. Note que no estamos ejecutando la función `muestraTamanho`, ya que no colocamos los paréntesis. Estamos apenas indicando el nombre de la función que debe ser ejecutada.

```
inputTamanho.oninput = muestraTamanho

function muestraTamanho(){
    outputTamanho.value = inputTamanho.value
}
```

¿Hay algún otro lugar del código en el cual necesitamos llamar esa función?, Pues hasta el momento no, ese ese el motivo por el cual asignamos un nombre a una función, para que sea posible usarla en más de un punto del código.

Es muy común que algunas funciones tengan una única referencia en el código. Es nuestro caso con la función `muestraTamanho`. En estos casos, JavaScript permite que creemos la función en el lugar donde antes estábamos indicando su nombre.

```
inputTamanho.oninput = function() {
    outputTamanho.value = inputTamanho.value
}
```

Transformamos la función `muestraTamanho` en una función sin nombre, una **función anónima**. La misma continúa siendo ejecutada normalmente cuando el usuario altera el valor del tamaño.

## 21.15 MANIPULANDO STRINGS

Cuando una variable almacena una string, es permitido consultar su tamaño y realizar

transformaciones en su valor.

```
var empresa = "Alura";

empresa.length; // tamaño de la string


//sobreescribe "ura" por "ina" de la string almacenada en la variable
empresa.replace("ura","ina"); // retorna Alina
```

A partir de la variable `empresa`, usamos el `punto` seguido de la acción `replace`. El punto permite acceder a los atributos (que guardan valores) o funciones de un elemento/objeto.

## 21.16 INMUTABILIDAD

**String es inmutable**. Para entender este concepto veamos el ejemplo de abajo, si la variable `empresa` fuera impresa después de las llamadas a la función `replace`, el valor continuará siendo "Alura". Para obtener una string modificada, es necesario asignar el retorno de cada función que manipula la string, pues una nueva string modificada es retornada:

```
var empresa = "Alura";

// substitui la parte "ura" por "ina"
empresa.replace("ura","ina");
console.log(empresa); // imprime Alura, no cambió!

empresa = empresa.replace("ura","ina");
console.log(empresa); // imprime Alina, si cambió!
```

## 21.17 CONVERSIONES

JavaScript posee funciones de conversión de string para number:

```
var textoEntero = "10";
var entero = parseInt(textoEntero);

var textoFloat = "10.22";
var float = parseFloat(textoFloat);
```

Perciba que en el primer ejemplo tenemos una string con los caracteres 1 y 0 dentro. Parece un *number*, pero no es; podemos ver que no es posible por ejemplo sumar `"10"+2`, eso va retornar `"102"`, o sea, JavaScript transforma todo en string y concatena los valores (junta los 2 y retorna un nuevo valor). Por eso muchas veces es necesario usar el `parseInt` para garantizar que los números sean realmente del tipo *number*, posibilitando realizar todas las operaciones aritméticas.

La misma situación ocurre con el `parseFloat`, que transforma strings en número con punto flotante (*float*).

## 21.18 MANIPULANDO NÚMEROS

*Number*, así como string, también es inmutable. El ejemplo de abajo altera el número de casas decimales con la función `toFixed` . Esta función retorna una string, pero, para que esta funcione correctamente, su retorno necesita ser capturado:

```
var milNumber = 1000;
var milString = milNumber.toFixed(2); // recibe el retorno de la función
console.log(milString); // imprime la string "1000.00"
```

## 21.19 CONCATENACIONES

Es posible concatenar (juntar) tipos diferentes y JavaScript se encargará de realizar la conversión entre los tipos, pudiendo resultar en algo no esperado.

### String con String

```
var s1 = "Alura";
var s2 = "Innovación";
console.log(s1 + s2); // imprime AluraInnovación
```

### String con otro tipo de datos

Como vimos, JavaScript tentará ayudar realizando conversiones cuando tipos diferentes fueren usados en una operación, pero es necesario estar atentos a la forma como él las realiza:

```
var num1 = 2;
var num2 = 3;
var nombre = "Alura"

// ¿Qué  imprimirá?

// Exemplo 1:
console.log(num1 + nombre + num2); // imprime 2Alura3

// Exemplo 2:
console.log(num1 + num2 + nombre); // imprime 5Alura

// Exemplo 3:
console.log(nombre + num1 + num2); // imprime Alura23

// Exemplo 4:
console.log(nombre + (num1 + num2)); // imprime Alura5

// Exemplo 5:
console.log(nombre + num1 * num2); // imprime Alura6
// La multiplicación tiene precedencia
```

### NaN - not a number

Vea el código abajo:

```
console.log(10-"curso")
```

El resultado es `NaN` (*not a number* - no es un número). Esto significa que estamos intentando hacer operaciones matemáticas con valores que no son números. El valor `NaN` aún posee una peculiaridad, definida en su especificación:

```
var resultado = 10-"curso"; // retorna NaN
resultado == NaN; // false
NaN == NaN; // false
```

No es posible comparar una variable con `NaN`, ni mismo `NaN` con `NaN`! Para saber si una variable es `NaN`, debe ser usada la función **isNaN()**:

```
var resultado = 10-"curso";
isNaN(resultado); // true
```

Capítulo 22

# PROPIEDADES CSS

## 22.1 PROPIEDAD FONT

```
.elemento {
  /* Controla el tamaño de la fuente */
  font-size: px, em, rem, pt, %;

  /* Controla el peso de la fuente */
  font-weight: 0 a 1000. Depende de la fuente;
  font-style: normal, italic, oblique;

  /* Controla la familia de la fuente */
  font-family: serif, sans-serif, monospace, custom;
}
```

## 22.2 PROPIEDAD TEXT

```
.elemento {
  /* Controla el alineamiento del texto */
  text-align: left, center, right, justify;

  /* Controla la capitalización del texto */
  text-transform: capitalize, uppercase, lowercase, none;

  /* Controla el tamaño de la indentación que es colocada antes de una línea de texto en un bloque */
  text-indent: px, em, rem, %;
}
```

## 22.3 PROPIEDAD LETTER-SPACING

```
.elemento {
  /* Controla el espaciado entre una letra y otra de un bloque de texto */
  letter-spacing: px, rem, em, %, pt;
}
```

## 22.4 PROPIEDAD LINE-HEIGHT

```
.elemento {
  /* Controla la altura de las líneas de un conjunto de texto */
  line-height: pt, px, rem, em, sin unidad de medida;
}
```

## 22.5 PROPIEDADES DE COLOR

```
.elemento {
  /* Hexadecimal #RRGGBB */
```

```
  color: #ff00ff;

  /* Hexadecimal shorthand #RGB */
  color: #f0f;

  /* Valor RGB de 0 la 255 rgb(R, G, B)*/
  color: rgb(255, 0, 255);

  /* Hexadecimal con opacidad (alpha) #RRGGBBAA */
  color: #ff00ff00;

  /* Valor RGB con opacidad */
  color: rgba(255, 0, 255, 0.0);
}
```

## 22.6 PROPIEDAD BACKGROUND

```
.elemento {
  /* Controla el color de fondo */
  background-color: hexadecimal, nombre, rgb, rgba;

  /* Coloca una imagen como plano de fondo */
  background-image: url();

  /* Controla el tamaño del plano de fondo. Dos valores pueden ser colocados, 'x' y 'y' o apenas el v
alor de 'x' que será agregado/eliminado proporcionalmente en y */
  background-size: x y, x/y, cover, contain, %, px, rem, em;

  /* Controla si la imagen de fondo va a tener repetición*/
  background-repeat: repeat, no-repeat;

  /* Controla la posición del plano de fondo */
  background-position: top right bottom left, top 10px, bottom 2rem right 2%;
}
```

## 22.7 PROPIEDAD BORDER

```
.elemento {
  /* Coloca un borde en todo elemento.
    Para espesura use un número con una unidad de medida.
    El estilo puede ser: none, dashed, dotted, double, groove, inset, outset, solid.
    Color: por las notaciones de colores ya vistas (nombre, hexadecimal, rgb ...)
 */
  border: espesura estilo color;

  /* Coloca un borde de arriba del elemento */
  border-top: espesura estilo color;

  /* Coloca un borde a la  derecha del elemento */
  border-right: espesura estilo color;

  /* Coloca un borde abajo del elemento */
  border-bottom: espesura estilo color;

  /* Coloca un borde a la  izquierda del elemento */
  border-left: espesura estilo color;
}
```

## 22.8 PROPIEDAD VERTICAL-ALIGN

```
.elemento {
  /* Alinea verticalmente elementos que son inline o inline-block */
  vertical-align: baseline, top, middle, bottom;
}
```

## 22.9 PROPIEDADES WIDTH Y HEIGHT

```
.elemento {
  /* Controla la anchura del elemento */
  width: px, rem, em, %;

  /* Controla la anchura mínima que un elemento puede tener */
  min-width: px, rem, em, %;

  /* Controla la anchura máxima que un elemento puede tener */
  max-width: px, rem, em, %;

  /* Controla la altura del elemento */
  height: px, rem, em, %;

  /* Controla la altura mínima que un elemento puede tener */
  min-height: px, rem, em, %;

  /* Controla la altura máxima que un elemento puede tener */
  max-height: px, rem, em, %;
}
```

## 22.10 PROPIEDAD BOX-SIZING

```
.elemento {
  /* Define cuál caja del box-model será usada como referencia para colocar propiedades de tamaño */
  box-sizing: border-box, content-box;
}
```

## 22.11 PROPIEDAD OVERFLOW

```
.elemento {
  /* Controla el comportamiento de los elementos internos que sobrepasan el espacio definido por la e
tiqueta madre*/
  overflow: visible, hidden, scroll, auto;

  /* Controla el comportamiento de los elementos internos que sobrepasan el espacio horizontal defini
do por la etiqueta madre */
  overflow-x: visible, hidden, scroll, auto;

  /* Controla el comportamiento de los elementos internos que sobrepasan el espacio vertical definido
 por la etiqueta madre */
  overflow-y: visible, hidden, scroll, auto;
}
```

Capítulo 23

# ATTRIBUTION-NONCOMMERCIAL-NODERIVATIVES 4.0 INTERNATIONAL

==========

Creative Commons Corporation ("Creative Commons") is not a law firm and does not provide legal services or legal advice. Distribution of Creative Commons public licenses does not create a lawyer-client or other relationship. Creative Commons makes its licenses and related information available on an "as-is" basis. Creative Commons gives no warranties regarding its licenses, any material licensed under their terms and conditions, or any related information. Creative Commons disclaims all liability for damages resulting from their use to the fullest extent possible. Using Creative Commons Public Licenses Creative Commons public licenses provide a standard set of terms and conditions that creators and other rights holders may use to share original works of authorship and other material subject to copyright and certain other rights specified in the public license below. The following considerations are for informational purposes only, are not exhaustive, and do not form part of our licenses. Considerations for licensors: Our public licenses are intended for use by those authorized to give the public permission to use material in ways otherwise restricted by copyright and certain other rights. Our licenses are irrevocable. Licensors should read and understand the terms and conditions of the license they choose before applying it. Licensors should also secure all rights necessary before applying our licenses so that the public can reuse the material as expected. Licensors should clearly mark any material not subject to the license. This includes other CC- licensed material, or material used under an exception or limitation to copyright. More considerations for licensors: wiki.creativecommons.org/Considerations_for_licensors

Considerations for the public: By using one of our public licenses, a licensor grants the public permission to use the licensed material under specified terms and conditions. If the licensor's permission is not necessary for any reason--for example, because of any applicable exception or limitation to copyright--then that use is not regulated by the license. Our licenses grant only permissions under copyright and certain other rights that a licensor has authority to grant. Use of the licensed material may still be restricted for other reasons, including because others have copyright or other rights in the material. A licensor may make special requests, such as asking that all changes be marked or described. Although not required by our licenses, you are encouraged to respect those requests where reasonable. More_considerations for the public:

## WIKI.CREATIVECOMMONS.ORG/CONSIDERATIONS_F

# OR_LICENSEES

Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions. Section 1 -- Definitions. a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image. b. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights. c. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements. d. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material. e. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License. f. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license. g. Licensor means the individual(s) or entity(ies) granting rights under this Public License. h. NonCommercial means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange. i. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them. j. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world. k. You means the individual or entity exercising the

Licensed Rights under this Public License. Your has a corresponding meaning. Section 2 -- Scope. a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

   a. reproduce and Share the Licensed Material, in whole or

   ```
   in part, for NonCommercial purposes only; and
   ```

   b. produce and reproduce, but not Share, Adapted Material

   ```
   for NonCommercial purposes only.
   ```

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a) (4) never produces Adapted Material.

5. Downstream recipients.

   a. Offer from the Licensor -- Licensed Material. Every

   ```
   recipient of the Licensed Material automatically
   receives an offer from the Licensor to exercise the
   Licensed Rights under the terms and conditions of this
   Public License.
   ```

   b. No downstream restrictions. You may not offer or impose

   ```
   any additional or different terms or conditions on, or
   apply any Effective Technological Measures to, the
   Licensed Material if doing so restricts exercise of the
   Licensed Rights by any recipient of the Licensed
   Material.
   ```

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i). b. Other rights.

7. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

8. Patent and trademark rights are not licensed under this Public License.

9. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties, including when the Licensed Material is used other than for NonCommercial purposes. Section 3 -- License Conditions. Your exercise of the Licensed Rights is expressly made subject to the following conditions. a. Attribution.

10. If You Share the Licensed Material, You must:

    a. retain the following if it is supplied by the Licensor

    ```
    with the Licensed Material:

      i. identification of the creator(s) of the Licensed
         Material and any others designated to receive
         attribution, in any reasonable manner requested by
         the Licensor (including by pseudonym if
         designated);

     ii. a copyright notice;

    iii. a notice that refers to this Public License;

     iv. a notice that refers to the disclaimer of
         warranties;

      v. a URI or hyperlink to the Licensed Material to the
         extent reasonably practicable;
    ```

    b. indicate if You modified the Licensed Material and

    ```
    retain an indication of any previous modifications; and
    ```

    c. indicate the Licensed Material is licensed under this

    ```
    Public License, and include the text of, or the URI or
    hyperlink to, this Public License.
    ```

    For the avoidance of doubt, You do not have permission under this Public License to Share Adapted Material.

11. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

2. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable. Section 4 -- Sui Generis Database Rights. Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material: a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database for NonCommercial purposes only and provided You do not Share Adapted Material; b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database. For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights. Section 5 -- Disclaimer of Warranties and Limitation of Liability. a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU. b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU. c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability. Section 6 -- Term and Termination. a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically. b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

3. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

4. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License. c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License. d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License. Section 7 -- Other Terms and Conditions. a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed. b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License. Section 8 -- Interpretation. a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License. b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions. c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor. d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal

# PROCESSES OF ANY JURISDICTION OR AUTHORITY.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at creativecommons.org/policies, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses. Creative Commons may be contacted at creativecommons.org.