Posicionamiento y Layout

Introducción

A pesar de conocer a grandes rasgos el proceso que conlleva crear una maqueta, aún no hemos profundizado en algunos conceptos importantes a la hora de construir maquetas que representen fielmente el diseño presentado en una representación visual.

En esta unidad conoceremos algunas propiedades que nos ayudarán a controlar las dimensiones, bordes, márgenes y relleno de una caja o contenedor y los tipos de cajas que podremos usar. También aprenderemos a posicionar elementos dentro de un diseño usando reglas *CSS* de posicionamiento y cambiaremos el flujo de los elementos usando flotaciones.

Para poder aprender de mejor manera el uso y aplicación de estás propiedades usaremos como ejemplo el siguiente proyecto:

Conociendo el proyecto

Monttaña es una empresa de transporte dedicada al traslado de turistas de diversos puntos de interés al rededor de Chile. Esta empresa desea cambiar el diseño de las entradas de blog, las cuáles se encuentran totalmente desactualizadas del diseño principal del sitio.

Nuestro equipo compuesto por diseñadores UX y UI crearon la representación visual de la entrada usando como base la investigación realizada cuando crearon el sitio principal de Monttaña.

Nosotros con otras tareas que nos entrego el jefe de proyecto no pudimos construir la estructura *HTML* ni la base de estilos del sitio, de modo que otro maquetador de nuestro equipo hizo el trabajo por nosotros.

Al terminar su trabajo el maquetador nos entrega el proyecto en un archivo comprimido que incluye la maqueta, la representación visual y la guía de estilos con la cual realizó el trabajo.

Con todos estos elementos que tendremos a disposición le daremos forma al layout de nuestra maqueta.

Comencemos conociendo el contenido del archivo `index.html` que contiene la estructura de la entrada.

Contenido de index.html

Si nos fijamos está totalmente constituido por clases *BEM*, además este *HTML* tiene una gran cantidad de etiqueta que sería bueno analizar.

Primero nos encontraremos con una etiqueta `<nav>` que representa a la barra de navegación del sitio.

```
<header class="header">
    <nav class="navbar">
        <div class="navbar__logo">
            <a href="#">
                <img src="assets/images/logo.png" alt="Logo principal de
Monttaña tour operador" class="">
            </a>
        </div>
        <div class="navbar__container">
            <ul class="navbar__list">
                <li><a href="#">Tours</a></li>
                <li><a href="#">Nosotros</a></li>
                <li><a href="#">Blog</a></li>
                <li><a href="#">Contacto</a></li>
            </ul>
        </div>
    </nav>
</header>
```

Luego, veremos que fuera del `<header>` se encuentra la etiqueta `<main>`, la cual contiene a su vez otra etiqueta interesante llamada `<article>` que representa a cualquier contenido relacionado a contenidos como entradas de blog, comentarios, foros, entre otros.

Así mismo, al interior de `<article>` veremos otras etiquetas interesantes que vale la pena mencionar, como por ejemplo la etiqueta `<hgroup>` que agrupa a uno o varios títulos en su interior. Esto es útil para enfatizar el título principal y el subtítulo principal de nuestra página.

Además de esta etiqueta nos encontraremos con otro elemento muy útil llamado `<figure>` el cual es usado para contener diversos tipos de contenidos que hagan referencia al flujo principal de un artículo. En el se incluyen generalmente imágenes, videos, gráficos, diagramas, ejemplos de código, entre otros elementos.

En este caso, dentro de la etiqueta `<figure>`, podremos encontrar varias imágenes que hacen referencia al contenido principal de la entrada, y debajo de ellas está la etiqueta `<figcaption>` que provee una descripción de las imágenes incluidas dentro de `<figure>`.

```
<main>
    <article class="single-post">
        <hgroup class="single-post__heading">
            <h1 class="single-post__main-title">Los Imperdonables
Santiaguinos</h1>
            <h5 class="single-post__meta">20 enero de 2018 Wikiexplora</h5>
        </hgroup>
        <figure class="single-post__featured-image">
            <img src="assets/images/featured-image.jpg" alt="Vista de la
cordillera de los Andes vista desde Santiago" class="image_medium
image_relative">
            <figcaption class="image-caption">Vista de la cordillera de los
Andes vista desde Santiago.</figcaption>
        </figure>
    </article>
</main>
```

Sigamos conociendo las etiquetas que contiene la página. Al final del contenido, antes de la paginación podremos encontrar una cita de la fuente del contenido descrita en la entrada de prueba. Como vemos está se encuentra contenida por la etiqueta `<blockquote>`, que tiene un atributo `cite=""` que indica de dónde se obtuvo la cita.

```
<blockquote class="single-post__quote"
cite="http://www.wikiexplora.com/Los_Imperdonables_Santiaguinos">
    <p>Fuente: Wikiexplora</p>
</blockquote>
```

Las últimas dos etiquetas que veremos antes de conocer los elementos restantes de la maqueta son la etiqueta `<small>` que define a un texto de relevancia secundaria como los copyright y notas legales y la etiqueta llamada `<address>` que contiene información relevante del autor del sitio web.

```
<footer>
    <div class="footer__container">
        <div class="footer__logo">
            <a href="#"><img src="assets/images/logo-footer.png" alt="Logo
Monttaña tour operador"></a>
        </div>
        <small>Monttaña Tour Operador Limitada, &copy; Todos los derechos
reservados 2018.</small>
    </div>
    <div class="footer__container">
        <address class="footer__contact-info">
            <p><span class="footer_text-emphasis">Dirección:</span> Manuel
Montt 007, Providencia, Región Metropolitana, Chile.</p>
            <p><span class="footer_text-emphasis">Teléfono:</span> <a
href="tel:+56 98007123123">+56 9 8 007 123 123</a></p>
            <p><span class="footer_text-emphasis">Correo:</span> <a
href="mailto:hola@tourmonttana.cl"> hola@tourmonttana.cl</a></p>
        </address>
    </div>
</footer>
```

Estructura de proyecto:

Ahora veamos cómo se constituye la estructura del proyecto.

Anteriormente vimos que en la raíz del proyecto se encuentra `index.html` el cual contiene la estructura *HTML* de nuestro sitio, además de encontrarse la carpeta `assets`, lugar donde se encuentran los estilos e imágenes usados en la página.

```
.
├── assets
│    ├── css
│    ├── images
│    └── sass
└── index.html
```

Si ingresamos al interior de `css` encontraremos el output con los estilos transformados de *Sass*. Debajo encontraremos las imágenes que componen al artículo, incluyendo también los logos del sitio. Finalmente encontraremos la carpeta `sass` que incluye el manifiesto de *Sass*, junto con las carpetas *patrón 7-1* que contienen los archivos parciales de *Sass*.

```
.
├── assets
│    ├── css
│    │    ├── main.css
│    │    └── main.css.map
│    ├── images
│    │    ├── cerro-la-campana.jpg
│    │    ├── cerro-manquehue.jpg
│    │    ├── cerro-pintor.jpg
│    │    ├── cerro-pochoco.jpg
│    │    ├── cerro-provincia.jpg
│    │    ├── cerro-san-ramon.jpg
│    │    ├── featured-image.jpg
│    │    ├── glaciar-el-morado.jpg
│    │    ├── glaciar-juncal.jpg
│    │    ├── glaciar-la-paloma.jpg
│    │    ├── glaciar-san-francisco.jpg
│    │    ├── logo-footer.png
│    │    ├── logo.png
│    │    └── refugio-plantat.jpg
│    └── sass
│         ├── abstracts
│         ├── base
│         ├── components
│         ├── layout
│         ├── main.scss
│         ├── pages
│         ├── themes
│         └── vendors
└── index.html
```

Si miramos más en detalle los directorios podremos encontrar:

Abstracts: En abstracts veremos los parciales de *mixin* y variables. Por ahora *mixins* no contiene nada y variables contiene todos los elementos visuales que reutilizaremos dentro de la maqueta. Es importante acotar que estos valores *CSS* se encuentran en el interior de **la guía de estilos**.

Base: El directorio base contiene tres parciales que agregó el maquetador. El primero contendrá las reglas bases de la maqueta, el segundo contendrá las reglas reset para homogeneizar el sitio y el último contiene las reglas base de tipografías del sitio.

Components: En este directorio se encuentran los componentes del sitio. Según la guía de estilos estos están separados por botones, la leyenda de las imágenes, los tamaños usados para las imágenes, los estados de los links y estilos de la barra de navegación.

Layout: En layout se encuentra los parciales que componen el diseño como *header*, el contenido principal que en este caso es el artículo y el *footer*.

Pages y Themes: Los directorios `pages` y `themes` se encuentran sin contenido debido a que nuestra maqueta no es necesario el uso de estilos extras para páginas, ni tampoco incluye temas.

Vendors: Finalmente el directorio `vendors` contiene el archivo normalize, el cual nos ayudará a dar consistencia y a normalizar nuestros estilos en todo tipo de navegadores.

```
└── sass
    ├── abstracts
    │    ├── _mixins.scss
```

```
│   └── _variables.scss
├── base
│   ├── _base.scss
│   ├── _reset.scss
│   └── _typography.scss
├── components
│   ├── _buttons.scss
│   ├── _image-caption.scss
│   ├── _images.scss
│   ├── _links.scss
│   └── _navigations.scss
├── layout
│   ├── _footer.scss
│   ├── _header.scss
│   └── _single-post.scss
├── main.scss
├── pages
├── themes
└── vendors
    └── normalize.scss
```

Como vemos a esta maqueta le faltan estilos para ser realmente fiel a la representación visual, por lo tanto desde ahora nosotros construiremos los estilos restantes para este proyecto.

Conociendo el modelo de cajas

Antes comenzar a crear los estilos que faltan de la maqueta es importante conocer algunos conceptos importantes relacionados al tamaño y posicionamiento de los elementos.

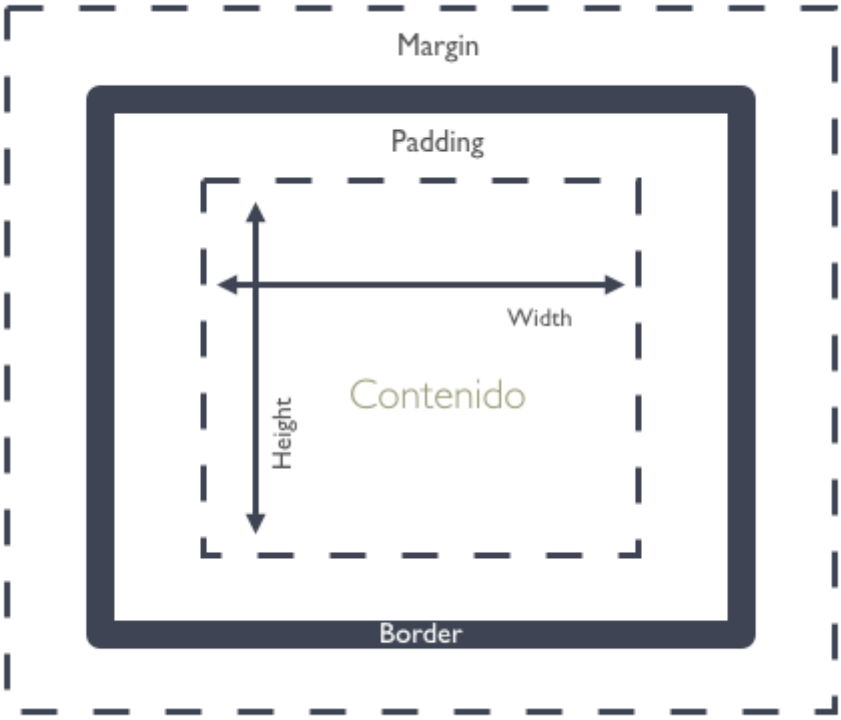
El navegador asume que nuestro documento *HTML* está modelado como un árbol de elementos y para que estos elementos sean dibujados en el documento deberán ser transformados en una caja, la cual tiene un tamaño y una posición.

El tamaño es cuánto espacio ocupa el elemento en nuestro sitio y la posición tiene relación hacia donde se sitúa el elemento.

Comencemos hablando sobre el tamaño.

Para poder hacerlo deberemos conocer el modelo de cajas.

¿Qué es el modelo de cajas?



El modelo de cajas nos ayuda a manejar las propiedades asociadas al tamaño, borde y relleno de los elementos incluidos en una página web.

De esta forma nosotros podremos modificar cuanto espacio estos ocupan, agregar bordes y modificar el tamaño del borde, así como también modificar margen o espacio entre un elemento y otro y agregar relleno dentro del contenido.

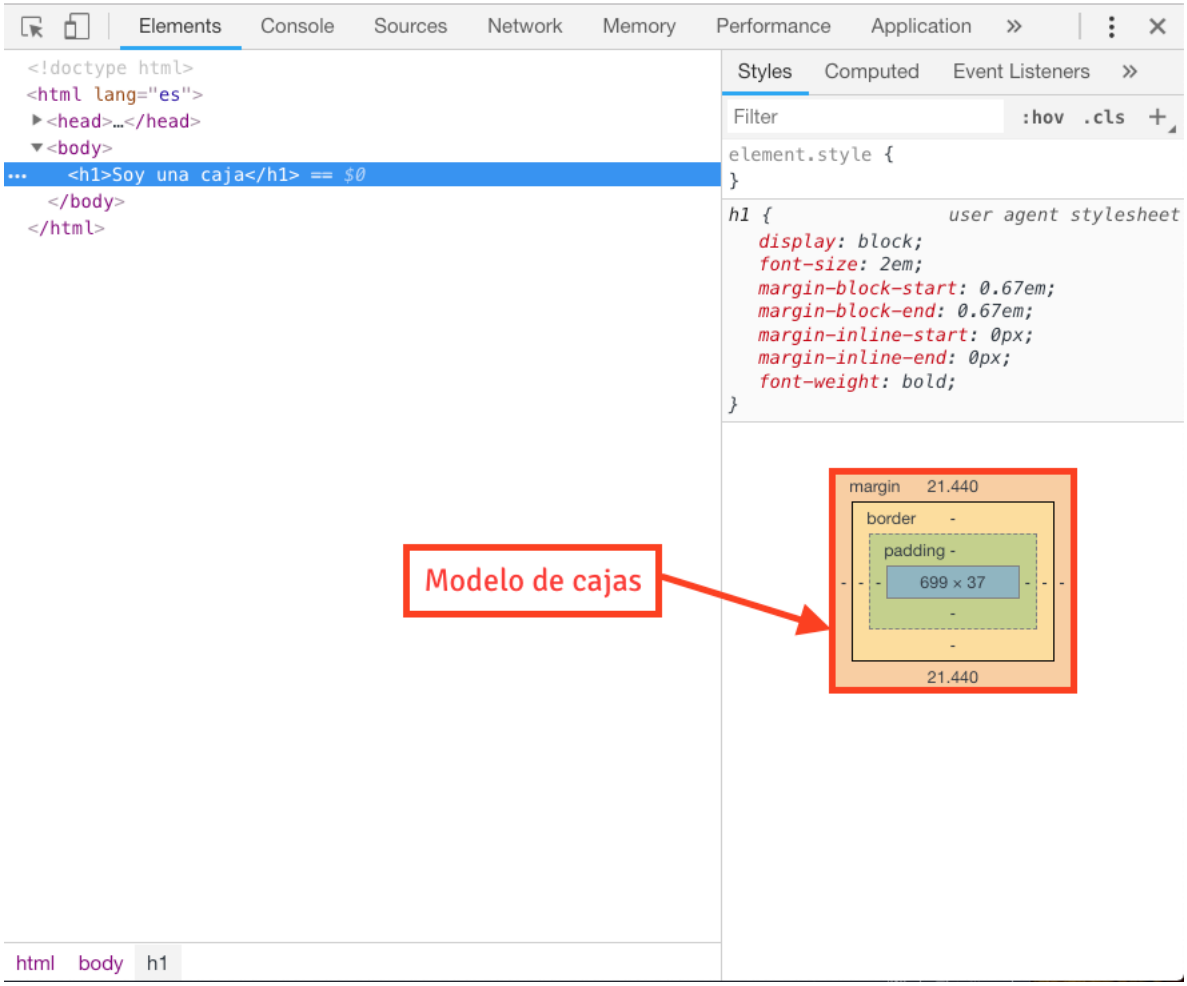
veamos un ejemplo par conocer más sobre el modelo de cajas:

Para eso crearemos una nueva página dentro de nuestro editor de texto. Cuando ya este creada agregaremos la base del HTML y un texto descriptivo como por ejemplo "Soy una caja". Al finalizar guardaremos el archivo en algún lugar fácil de encontrar.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Modelo de cajas</title>
</head>
<body>
    <h1>Soy una caja</h1>
</body>
</html>
```

Luego, busquemos el archivo y abramos la página con el navegador. Cuando veamos que el documento HTML se cargó presionemos el botón derecho e ingresemos a la opción inspeccionar elementos. Desde aquí deberemos asegurarnos que estamos en la pestaña `elementos`.

Ahora que sabemos que estamos ahí seleccionemos el elemento a inspeccionar, que en este caso será `<h1>` y en la pestaña `styles` aparecerá el modelo de cajas. Aquí podremos ver el tamaño que ocupa, su *padding*, borde y márgenes.

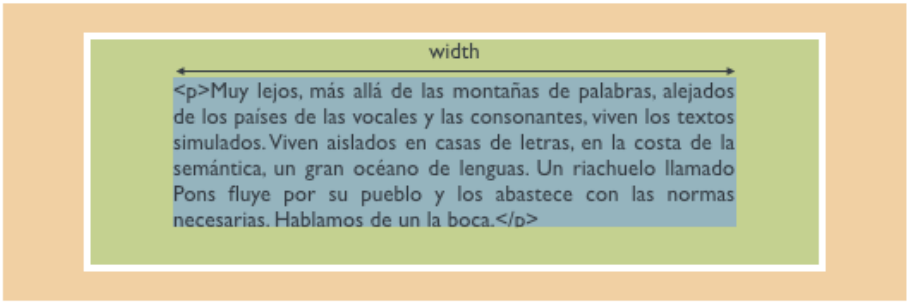


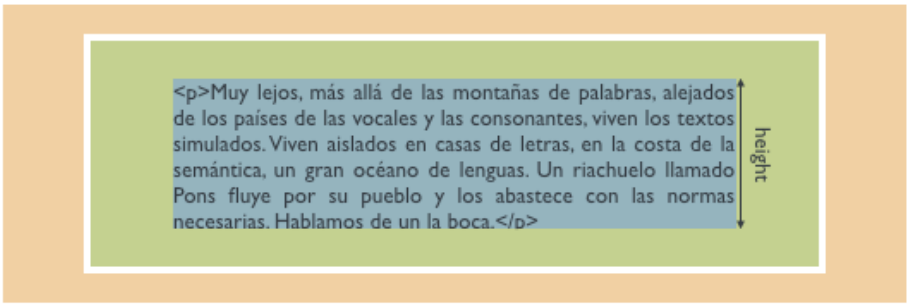
Dimensiones de una caja

Ahora que ya sabemos cómo ver visualmente el modelo de cajas es momento de conocer algunas propiedades CSS básicas que contiene el modelo de cajas.

Width y height

Comencemos con las propiedades fundamentales como es el ancho y el largo.





`Width` indica cuánto ancho tenemos para agregar contenido dentro de nuestra caja y cuanto alto hay disponible para el contenido.

Por defecto las cajas se ajustan al tamaño del contenido que incluyen, de modo que es importante dimensionar su tamaño usando las propiedades anteriormente mencionadas.

Para conocer de mejor manera esto agreguemos una caja dentro del documento *HTML* que usamos como ejemplo anteriormente. En él crearemos un `<div>` con clase `class="container_blue"` y un un texto simple, como por ejemplo "Soy una caja azul". En `<head>` agreguemos una etiqueta `<style>` y dentro de ella escribamos la clase `class="container_blue"` y finalmente agreguemos un `background-color: blue;` y un color de texto blanco `color: white;`.

```
<!DOCTYPE html>
<html lang="es">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Modelo de cajas</title>
        <style media="screen">
                .container_blue {
                        background-color: blue;
                }
        </style>
</head>
<body>
        <h1>Soy una caja</h1>
        <div class="container_blue">Soy una caja azul</div>
</body>
</html>
```

Si guardamos y recargamos la página podremos ver que la caja, aún cuando no le hemos dado dimensiones, tendrá un ancho y un alto relacionados a su contenido.

```
<style media="screen">
        .container_blue {
                width: 500px;
                height: 500px;
                background-color: blue;
                color: white;
        }
</style>
```

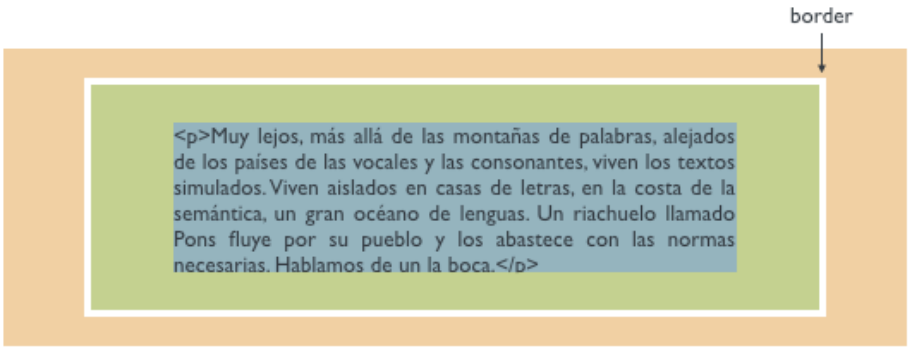
Si le damos un ancho y un alto `500px`, podremos ver que la caja tiene las dimensiones definidas por nosotros. Esto a grandes rasgos es importante, ya que podremos tener el control total del tamaño de una caja.

Controlando el modelo de cajas

Sigamos conociendo las propiedades que podremos usar para controlar el modelo de cajas, en esta ocasión revisaremos las propiedades `border`, `margin` y `padding`. El comprender cómo funcionan estas propiedades hará que trabajar en la distribución de los elementos de un proyecto sea rápido y consistente.

Border

Comencemos con la propiedad *border*.



Border nos permite especificar las características del borde de un elemento Estas características pueden ser: el estilo del borde, su ancho y color.

```
/* Reglas CSS básicas de border */
p {
        border-style: solid;
        border-width: 1px;
        border-color: black;
}

/* Versión corta de propiedad border */
div {
        border: solid 1px black;
}
```

A su vez, existe una versión corta de todas las propiedades vistas anteriormente llamada `border`, en la cual podremos agregar el tipo el tamaño del borde, el tipo de borde que se usará y el color que tendrá, este orden es excluyente para esta propiedad.

Algo importante a tener en cuenta sobre *border* es que todas la cajas por defecto incluyen borde, aún cuando no hayamos definido sus características.

Ahora que sabemos esto pongamos en práctica estas propiedades usando el *HTML* que usamos anteriormente. Dentro de la clase `.container_blue` agreguemos un `border-style: solid`, luego demos un tamaño de ancho al borde de `10px` y un color `coral`.

```
<style media="screen">
        .container_blue {
                width: 500px;
                height: 500px;
                border-style: solid;
                border-width: 10px;
                border-color: coral;
                background-color: blue;
                color: white;
        }
</style>
```

Sí guardamos y recargamos la página podremos ver que se creó un borde de `10px`, con bordes color coral.

Lo mismo que hicimos anteriormente lo podremos hacer de manera más limpia usando la versión corta de estas propiedades.

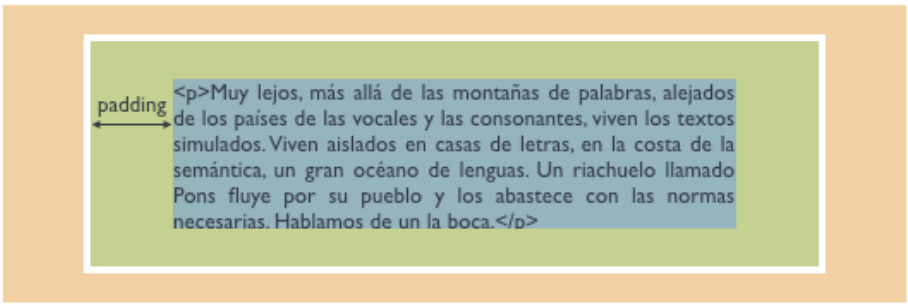
Borremos las tres reglas que agregamos hace un rato y agreguemos la propiedad `border` con tamaño de `10px`, solido y un color `coral`.

```
<style media="screen">
        .container_blue {
                width: 500px;
                height: 500px;
                border: 10px solid coral;
                background-color: blue;
                color: white;
        }
</style>
```

Veamos qué sucede al recargar. Pues bien, es el mismo resultado.

Padding

La siguiente propiedad que veremos será *padding*.



El *padding* es el espacio que hay entre el contenido y el borde dentro de un elemento.

Existen diferentes propiedades que podemos usar para controlar el relleno o aire del elemento. Cada una representa el lugar en donde se modificará, pudiendo modificar el *padding* hacia arriba, derecha, abajo e izquierda.

Además de las propiedades vistas, podremos usar una versión corta que incluye todas las reglas de de *padding*. Estas deberán seguir un orden que siguen al sentido del reloj, o sea, arriba, derecha, abajo, izquierda.

Revisemos como funciona *padding* usando el ejemplo anterior.

Cambiemos el contenido asociado al `<div>` con clase `.containter_blue` por un texto pre-formateado *lorem ipsum*.

```
<div class="container_blue">
        Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
</div>
```

Ahora agreguemos un *padding* dentro del selector `.containter_blue` con un valor de `30px`.

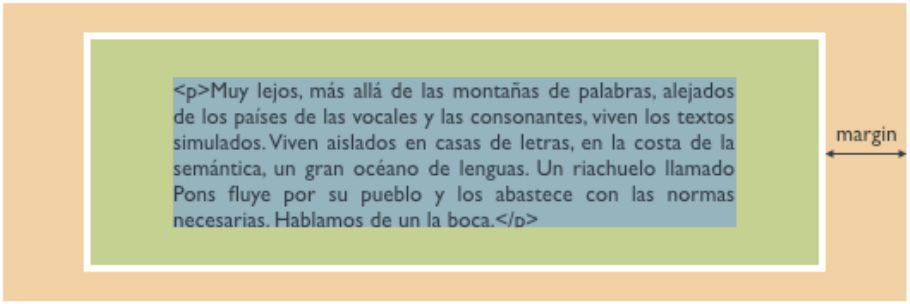
```
<style media="screen">
        .container_blue {
                width: 500px;
            height: 500px;
            border: 10px solid coral;
            padding: 30px;
            background-color: blue;
            color: white;
        }
</style>
```

Si recargamos la página podremos ver que se agrego un espacio de `30px` hacia todas direcciones. En el caso de querer definir diferentes valores hacia el alto y el ancho, podremos hacerlo usando agregando más de un valor como por ejemplo `20px` alto y `30px` de ancho como relleno.

Como vemos la disposición del contenido cambió dentro de la caja haciendo que el contenido se vea mas ordenado y limpio dentro del layout.

Margin

La última propiedad del modelo de cajas que veremos será *margin*.



Los márgenes de una caja son el espacio que hay entre el borde y los elementos que lo rodea, eso quiere decir que con los márgenes podremos crear espacios entre el borde de dos cajas contiguas.

Los márgenes, al igual que *padding*, tienen diferentes propiedades que podemos usar para controlar el margen del elemento. Cada una representa el lugar en donde se modificará el margen del elemento. Las opciones que tendremos serán margen hacia arriba, derecha, abajo e izquierda.

Además, podremos usar una versión corta que incluye todas las reglas de margen, las cuales son similares a *padding*. Estas deberán seguir un orden que siguen al sentido del reloj, o sea, arriba, derecha, abajo, izquierda.

```
/* Reglas usadas para margin */
p {
        margin-top: 10px;
        margin-right: 20px;
        margin-bottom: 30px;
        margin-left: 40px;
}

/* Versión corta de propiedad margin */
div {
        margin: 10px 20px 30px 40px;
}
```

Veamos cómo funciona esta propiedad copiando un el `<div>` que creamos hace poco y pegando este abajo. Luego dentro de la clase `.containter_blue` agreguemos un margen con valores de `30px`, `20px`, `30px`, y `20px`.

```
<style media="screen">
        .container_blue {
            width: 500px;
            height: 500px;
            border: 10px solid coral;
            margin: 30px 20px 30px 20px;
            background-color: blue;
            color: white;
        }
</style>
```

Si revisamos el navegador veremos que las cajas se separaron una de la otra desde el borde.

Asimismo, si la propiedad tiene valores similares hacia arriba y abajo, y de izquierda a derecha, podremos disminuir los valores incluidos en esta propiedad acortándolo a sólo dos, esto quiere decir que si los valores `top` y `bottom` son iguales y los valores `left` y `right` también lo podremos acortar.

Cambiemos los valores del margen que agregamos anteriormente por esta nueva regla, eliminando los dos primeros valores.

```
<style media="screen">
        .container_blue {
            width: 500px;
            height: 500px;
            border: 10px solid coral;
            margin: 30px 20px;
            background-color: blue;
            color: white;
        }
</style>
```

Si recargamos el navegador veremos que la distribución de nuestras cajas se mantuvo.

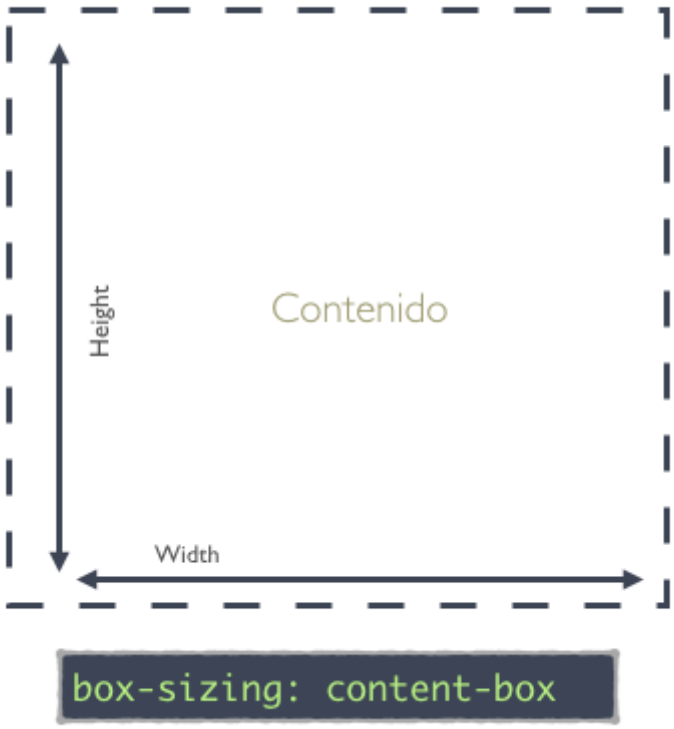
Calculando las dimensiones de una caja

Antes de comenzar a darles estilo a nuestro proyecto conoceremos una propiedad muy útil para calcular los valores de nuestras cajas de manera intuitiva llamada `box-sizing`.

Con la propiedad `box-sizing` podremos definir cómo se calcula el ancho y alto de un elemento.

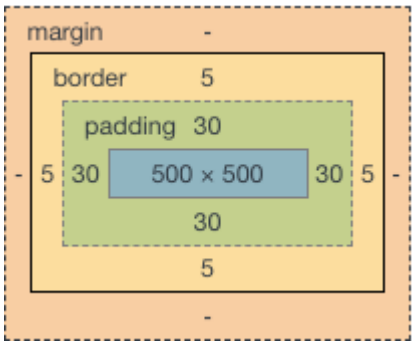
Podremos definir estas cajas como:

Content-box



La cual es la forma por defecto que el navegador calcula el modelo de cajas. Este valor suma solamente el alto y ancho definido en una caja, exceptuando el borde y *padding*.

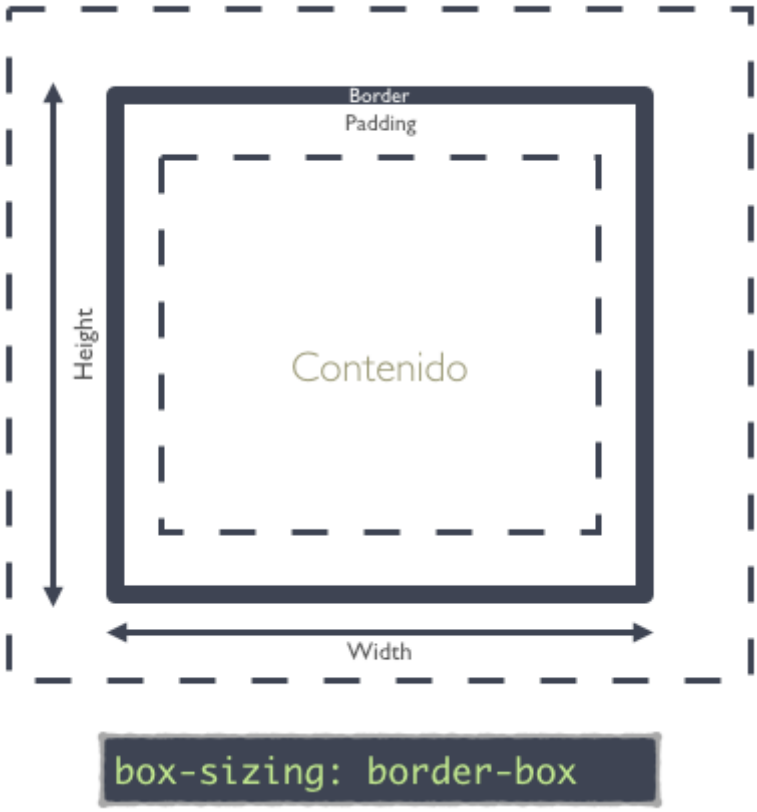
Esto en la práctica significa que si tenemos una caja de 500 x 500 píxeles de alto y ancho, un *padding* de `30px`, y un borde de `5px`, el tamaño será de `500px` de alto y ancho, sin embargo si sumamos este tamaño con el *padding* y el borde obtendremos un valor real de `535px`.



Calcular las cajas de esta forma puede resultar tediosa, especialmente cuando trabajamos con unidades de medida relativa o diseños responsivos.

Para evitar que pasemos mucho tiempo calculando nuestras cajas tendremos a disposición otro valor llamado *border-box*.

Border-box_

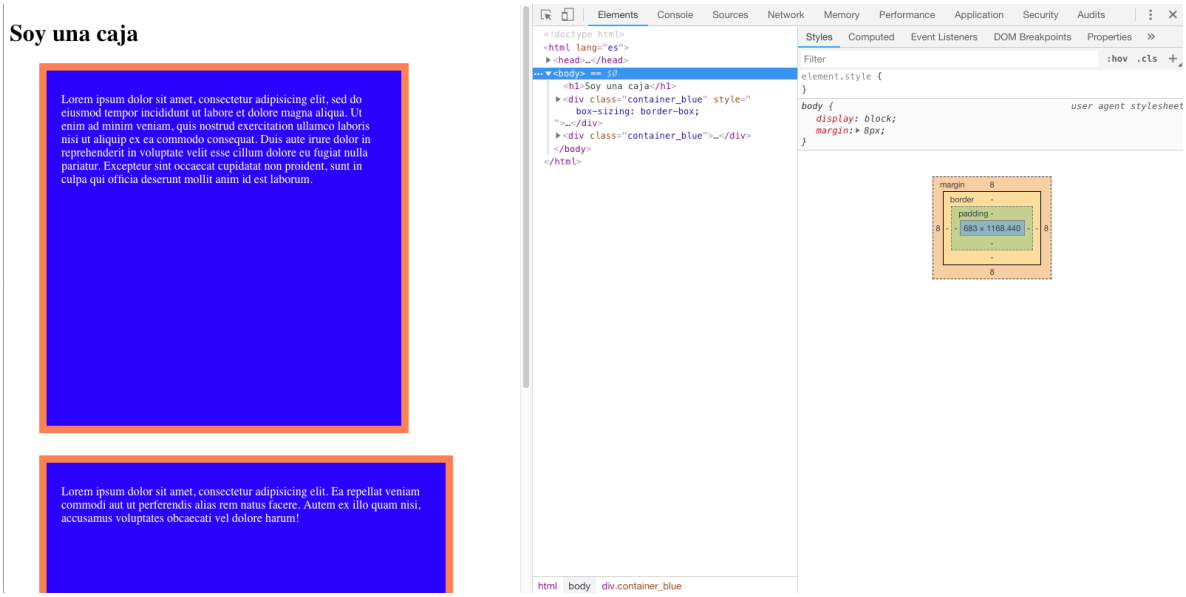


La propiedad *border-box* calcula el tamaño de la caja usando el alto y ancho, más el *padding* y el borde.

Veamos un ejemplo de este valor usando la página de prueba que hemos estado manipulando hasta ahora.

si revisamos las cajas que agregamos hace un rato estas vienen por defecto con un cálculo del tipo *content-box*, el cual tiene un valor total de la caja de `530px` de ancho y `520px` de alto, sin contar el valor del margen.

Ahora, si afectamos a la primera caja de la página agregando en el inspector de elementos la propiedad `box-sizing: border-box;` veremos que el cálculo asociado a la caja cambió a `500px` de ancho y `500px` de alto.



Como vemos el cálculo de nuestra caja al usar *border-box* es más fácil de hacer ya que los tamaños asociados a la caja como el *padding* y el borde se suman con el tamaño del alto y ancho, dando un mejor control de estas dimensiones por parte de nosotros.

Algo importante a tener en cuenta sobre estos valores es que en cualquiera de los dos casos es necesario agregar un alto o ancho para poder calcular el contenedor o caja que queramos afectar.

Ya teniendo claro como calcular nuestras cajas es momento de comenzar a dimensionar y controlar la distribución de las cajas del proyecto.

Aplicando el modelo de cajas en un proyecto

Ha llegado el momento de comenzar a dimensionar y controlar la distribución de las cajas en la maqueta a medio terminar.

Para hacerlo seguiremos el orden de carga que tienen los *@imports* del manifiesto de *Sass* a modo de crear los estilos de la maqueta de manera organizada.

Primero comencemos revisando los parciales del directorio base. Cómo sabemos acá se encuentran todos los estilos base del proyecto. En este caso el maquetador anterior configuró el tamaño del *body* a `1280px`, además de organizar los estilos base de las tipografías.

Por ahora nosotros no agregaremos estilos a estos parciales debido a que nuestro objetivo es organizar las distribución de la maqueta, cosa que por ahora no podemos decidir sin antes organizar el layout.

Agregando estilos al layout

En este directorio nos encontraremos con tres parciales los cuáles hacen referencia al diseño del mockup, el que está dividido en *header*, *main* o single post y *footer*.

Comencemos a dar estilos a nuestra maqueta dentro del parcial `_single-post` debido a que es el lugar más fácil por donde comenzar.

La primera parte que debemos distribuir su contenido son los títulos principales.

Estos estilos son parte del bloque `.single-post__heading` los cuáles debemos centrar para obtener un resultado similar al mockup. Para ello usaremos la una propiedad muy útil llamada `text-align`.

¿Qué es *text-align*?

Esta propiedad nos ayudará con el alineamiento del contenido dentro de una caja. Las posibilidades que tenemos para posicionar el contenido dentro de una caja con esta propiedad son `left`, `center` y `right`.

Para nuestro caso dentro de la clase `.single-post__heading` agregaremos esta propiedad y le daremos un valor `center`. Asimismo, para dejar al título comprimido, tal como se ve en el mockup, debemos darle un *padding* de a lo menos `200px` de ancho y 0 de alto, de esta manera el título se verá similar a la representación visual.

```
.single-post__heading {
        text-align: center;
        padding: 0 200px;
}
```

Ahora le daremos algunos estilos los títulos y meta de la entrada. Para eso agregaremos escribiremos la `.single-post__main-title` y dentro de ella le daremos un tamaño de gigante con la variable `$xxx-large`.

```
.single-post__heading {
        text-align: center;
        padding: 0 200px;
}
```

Luego seguiremos con la clase `.single-post__meta` la cual contiene la información referente al creador de la entrada. Agreguemos un tamaño de fuente `$x-small` y definamos el color del texto a `$nickel`.

```
.single-post__meta {
        font-size: $x-small;
        color: $nickel;
}
```

Ahora demos un margen al ancho de nuestra imagen a modo de dar un tamaño similar a lo visto en el mockup. `0` de alto y `100px` de ancho está bien.

Luego centremos la imagen principal de la entrada usando la propiedad `text-align:center;` al interior de la clase `.single-post__featured-image`.

```
.single-post__featured-image {
        margin: 0 100px;
        text-align: center;
}
```

Con esto hemos terminado de definir los estilos de los títulos principales e imagen destacada, lo que es muy bueno, pero aún nos quedan algunas reglas por agregar para que representemos fielmente el contenido visual del mockup.

La siguiente clase que afectaremos será `.single-post__summary` la cual contiene la descripción del contenido del artículo.

Este tiene un espacio entre el contenido la imagen destacada y el contenido principal. Para que quede similar le daremos un *padding* de `10px` de alto y `0` de ancho.

```
.single-post__summary {
        padding: 10px 0;
}
```

La siguiente clase que afectaremos será el contenedor de contenido de la entrada al cual le daremos más espacio para que el contenido se comprima aún más. Agreguemos la clase `.single-post__content` y dentro de ella demos un *padding* de `0` de alto y `200px` de ancho.

Ahora usando *nestings* le daremos un tamaño de fuente a los `<h2>` con la variable `$xx-large`.

```
.single-post__content {
        padding: 0 200px;

        h2 {
                font-size: $xx-large;
        }
}
```

Las estilos que haremos ahora en este parcial será cambiar el margen izquierdo que viene por defecto en las imágenes.

Para eso iremos agregaremos la clase `.single-post__image` la cual tendrá un margen de `0` píxeles.

Luego de esto crearemos la clase `.single-post__quote` la que contendrá un *margin-left* de `0` píxeles, a modo de mover la etiqueta `<blockquote>` hacia la izquierda. Esta etiqueta es usada para contener citas o referencias de un contenido específico. En este caso todo el contenido de la entrada escrita en el mockup hace referencia a una entrada escrita en blog externo.

Sabido esto, usemos nesting para agregar cambiar los estilos de la etiqueta `<p>`. Dentro de esta cambiemos el tamaño de la fuente por más pequeña, o sea, usando la variable `$x-small`.

```
.single-post__quote {
        margin-left: 0;

        p {
                font-size: $x-small;
        }
}
```

Finalmente centremos los botones usando `text-align: center`.

```
.single-post__pagination {
        text-align: center;
}
```

Si nos fijamos los botones tienen los colores y estados como los plantea la guía de estilos, pero aún le falta el relleno. Para hacerlo iremos al parcial `_buttons` del directorio `containers`, y dentro de este archivo agregaremos un *padding* `8px` de alto y `20px` de ancho.

Con esto hemos finalizado de agregar los estilos específicos para el contenido principal de la maqueta.

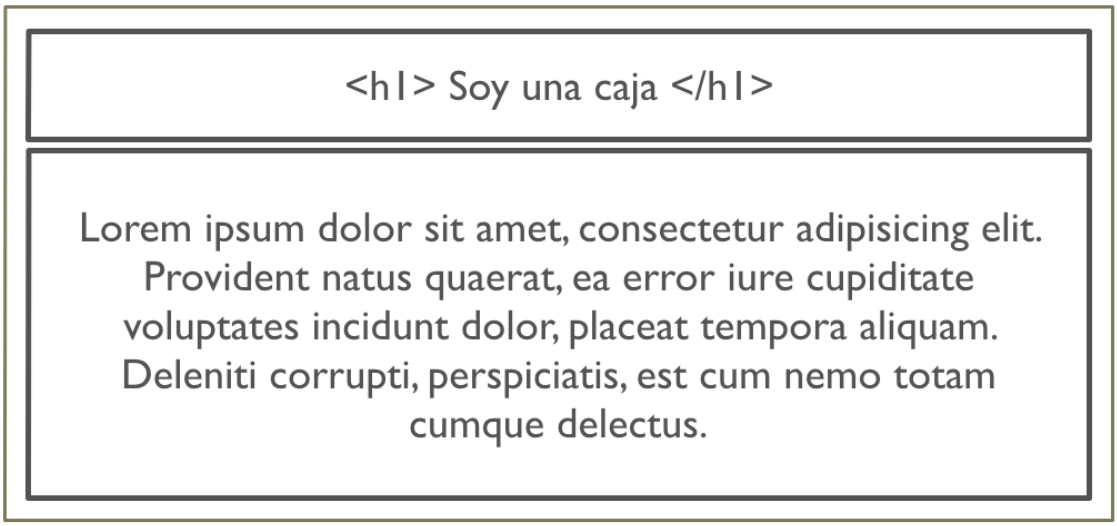
Tipos de Caja

Ahora que tradujimos el contenido principal del mockup a estilos *SCSS*, es momento de conocer un concepto importante a la hora de trabajar con el modelo de cajas.

Los elementos que componen un documento *HTML*, por lo general, tienen un comportamiento que los define dentro del flujo del *HTML*. Este comportamiento depende del tipo de elemento que representa al elemento.

Por ejemplo:

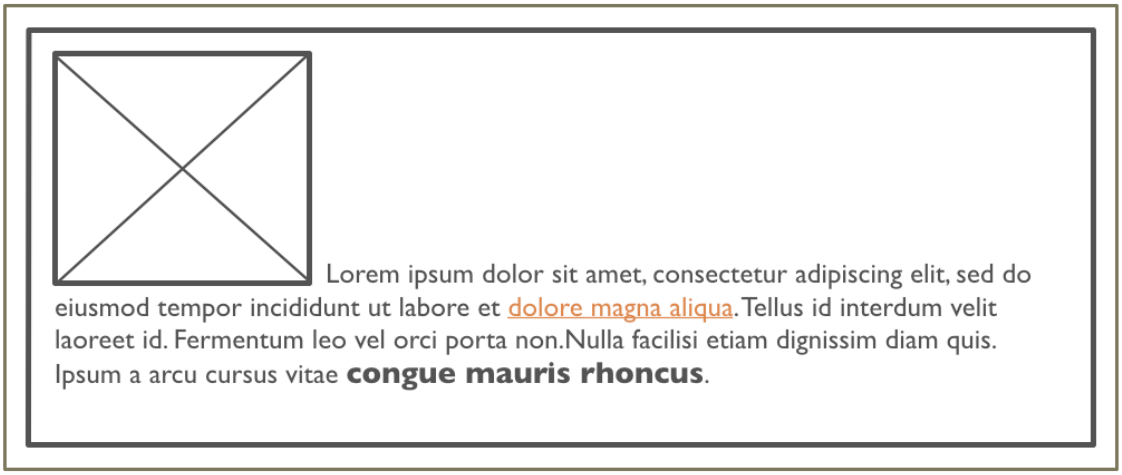
Elementos de bloque



Un título principal y un párrafo representan a un elemento de bloque. Esto significa que los elementos de bloque siempre comenzarán en una nueva línea y ocuparán todo el espacio disponible de la página, siempre y cuando no se defina un ancho para este elemento.

Algunos ejemplos de elementos de bloque son las etiquetas: `<li>`, `<h1>`, `<p>`, `<div>`, etc.

Elementos de línea



Por otra parte, existen otros elementos que se comportan diferente a los elementos de bloque los cuáles son llamados elementos inline. Estos se caracterizan por fluir al rededor de un texto usando sólo ocupan el mínimo espacio.

Alguno de los elementos *HTML* que tienen este comportamiento son las etiquetas: `<img>`, `<span>`, `<a>`, `<i>`, entre otras.

Alineamiento de cajas

Antes de continuar es importante acotar que el alineamiento de algunos elementos que realizamos en la maqueta tienen directa relación con los tipos de cajas vistos hasta ahora.

Cuándo realizamos un `text-align` en un elemento de bloque, los elementos se moverán hacia la izquierda, centro o derecha sin problemas ya que estos ocupan todo el espacio disponible, mientras que los elementos de línea, que ocupan la menor cantidad de espacio posible, no tienen la facultad de alinear su contenido debido a este límite de espacio.

Asimismo, el comportamiento de los elementos vistos anteriormente pueden ser cambiados usando la propiedad de *CSS* llamada `display`.

Cambiando el comportamiento de los elementos

Con la propiedad `display` con la cual podremos cambiar el comportamiento de los elementos a modo de especificar el tipo de comportamiento que deseamos que tenga este dentro de un layout.

En palabras simples, con esta regla podremos cambiar el estado de los elementos den línea a bloque y viceversa dándonos el poder absoluto para definir la disposición de los elementos dentro de nuestra maqueta.

Los valores que podremos usar para definirlos son:

block: Transforma el comportamiento de un elemento inline a bloque

inline: Hace que los elementos de bloque se comporten como si fueran inline

block: Transforma el comportamiento de un elemento inline a bloque

inline-block: Cambia el comportamiento los elementos de línea dándole atributos de elementos de bloque.

En esencia los valores `inline-block` permiten a los elementos en línea afectados agregar un alto y/o ancho dentro de otros elementos del mismo tipo.

Un ejemplo de este este comportamiento es un `<span>` dentro de un párrafo. Si agregamos la regla `display: inline-block`, junto con un alto y ancho de `200px` y un fondo de color `coral` dentro del `<span>` podremos ver que el elemento `<span>` se comporta como si fuera un elemento de bloque dentro de un elemento inline.

```
<!DOCTYPE html>
<html lang="es">
<head>
        <meta charset="UTF-8">
        <title>Modelo de cajas</title>
        <style media="screen">
                span {
                        display: inline-block;
                        width: 200px;
                        height: 200px;
                        background-color: coral;
                }
        </style>
</head>
<body>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Natus
repellendus, quo praesentium <span>officia provident tempore</span>, accusamus
vel, atque, repudiandae culpa doloribus minus maiores consequuntur. Magnam
sapiente harum veniam iste esse!</p>
</body>
</html>
```

none: Los valores none nos ayudarán a eliminar un elemento dentro de una página.

A grandes rasgos esta propiedad elimina visualmente un elemento que se encuentre dentro de nuestra maqueta, no obstante, este no desaparece del código, o sea que sí abrimos el inspector de elementos veremos que el elemento aún se encuentra ahí.

Por otra parte, existe una propiedad que podremos usar para esconder un elemento sin necesidad de eliminar el elemento desde la maqueta usando `visibility`.

¿Qué hace esta propiedad?

La propiedad `visibiity` nos ayudará a esconder los elementos que no queramos mostrar hacia los usuarios dentro de la interfaz.

Al usar esta regla el elemento se esconderá dejando un espacio en donde este se encontraba.

Probemos esta propiedad agregando al párrafo con el `<span>` un `visibiity:hidden;`, recarguemos y veamos el resultado.

Cómo vemos el elemento desapareció del lugar, pero mantuvo el espacio que agregamos con la propiedad `display:inline-block`.

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
        <meta charset="UTF-8">
        <title>Modelo de cajas</title>
        <style media="screen">
                span {
                        display: inline-block;
                        visibility: hidden;
                        background-color: coral;
                        width: 200px;
                        height: 200px;
                }
        </style>
</head>
<body>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Natus
repellendus, quo praesentium <span>officia provident tempore</span>, accusamus
vel, atque, repudiandae culpa doloribus minus maiores consequuntur. Magnam
sapiente harum veniam iste esse!</p>
</body>
</html>
```

Si quisiéramos mostrar el contenido usando esta propiedad el valor deberá ser `visiblity: visible;`

¿Cuál es la diferencia entre *visibility: hidden* y *display: none*?

Otra tema a tener en cuenta acerca de la propiedad `visibility: hidden;` es la diferencia que tiene con `display: none;`.

Estos dos valores, aunque similares, tienen una gran diferencia la que radica en el hecho que la propiedad `visibility: hidden;` esconde los elementos y display `display: none;` los elimina del documento.

Para ver esta diferencia revisemos el siguiente ejemplo.

Primero comentemos la propiedad `visibility: hidden;` incluida dentro del selector `<span>` y luego cambiemos el valor de `display: inline-block;` por `none` y veamos que sucede.

```
<!DOCTYPE html>
<html lang="es">
<head>
        <meta charset="UTF-8">
        <title>Modelo de cajas</title>
        <style media="screen">
                span {
                        display: none;
                        /* visibility: hidden; */
                        background-color: coral;
                        width: 200px;
                        height: 200px;
                }
        </style>
</head>
<body>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Natus
```

```
repellendus, quo praesentium <span>officia provident tempore</span>, accusamus
vel, atque, repudiandae culpa doloribus minus maiores consequuntur. Magnam
sapiente harum veniam iste esse!</p>
</body>
</html>
```

Cómo vemos el texto incluido dentro del `<span>` desapareció de la página e hizo que el layout de nuestra página de prueba cambiara su distribución.

Conociendo el posicionamiento de elementos

Cómo hablamos anteriormente, los elementos de un documento tienen diferentes comportamientos dependiendo del tipo de elemento que sea este, como un tipo *inline* o *block*.

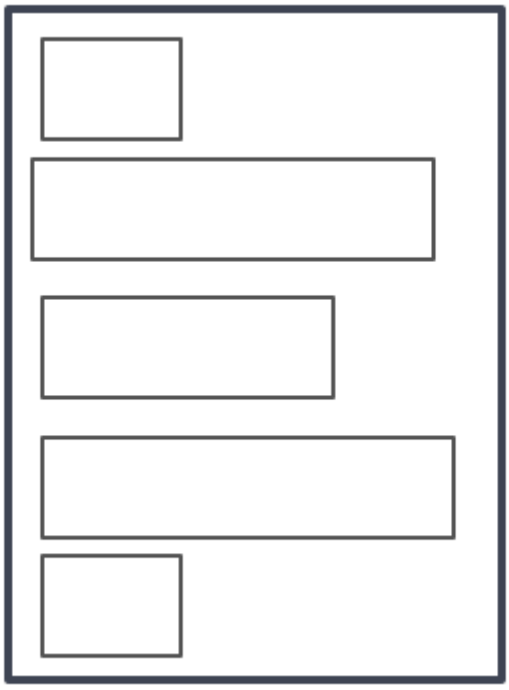
Estos tipos de elementos se pueden incluir dentro de otras cajas, que como sabemos por *BEM* se transforman en un bloque, o también conocido como elementos padres o contenedores.

Al diagramar al interior de contenedores es importante para una correcta distribución controlar la posición en la que se encuentran los elementos dentro del diseño.

Tipos de posicionamiento

Para hacerlo *CSS* tiene una propiedad llamada `position`, con la cual podremos posicionar elementos uno por sobre otro, especificando el tipo de posicionamiento que se utilizará en el elemento.

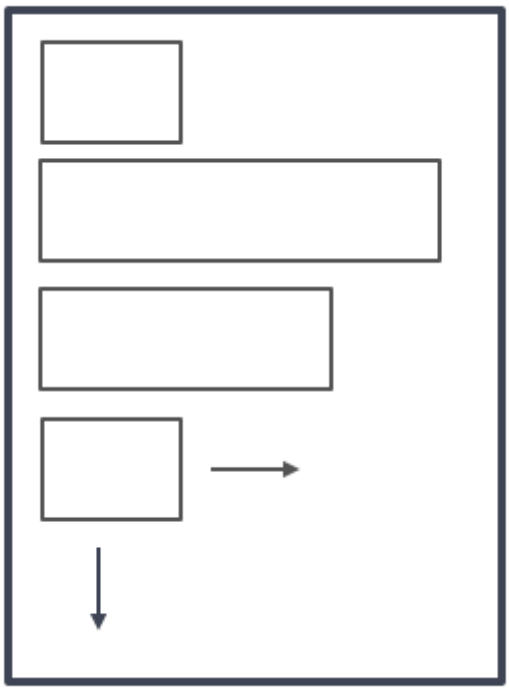
Posiciones estáticas



Por defecto los navegadores utilizan `position: static;` por defecto, de modo que todos los elementos que hemos trabajado hasta el momento se encuentran estáticos, esto quiere decir que los elementos se renderizan o dibujan de acuerdo al flujo normal del *HTML*. Asimismo, cualquier elemento el cual no se especifique su posición será *static*.

Si vamos al ejemplo de las cajas podremos ver que los elementos se posicionan uno por sobre otro siguiendo el flujo normal del *HTML*.

Posiciones relativas



Con los valores relativos podemos renderizar un elemento en el documento y luego de haberse dibujado cambiar su posición respecto al flujo normal del *HTML*. Para reposicionar los elementos se pueden usar las propiedades `top`, `right`, `bottom` o `left`.

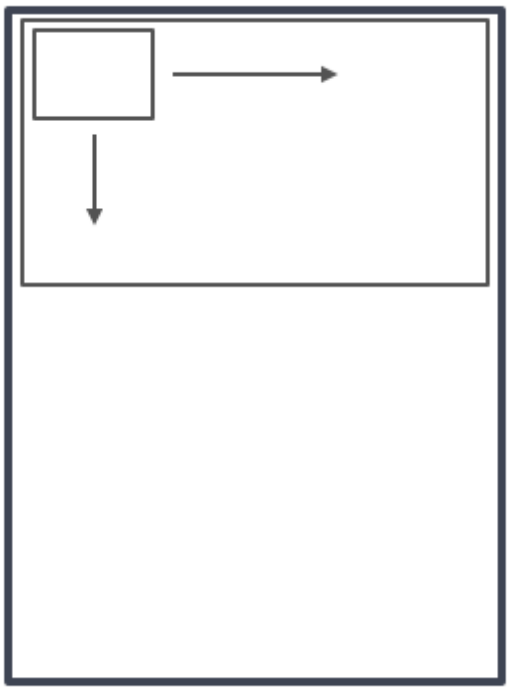
Para probar este valor agreguemos en la segunda caja un `position:relative;` y luego incluyamos la propiedad `left` con un valor de `300px`.

Cómo podemos ver el valor la caja se movió a la izquierda. Si quisiéramos bajar el contenedor podríamos agregar un `top` con un valor de `100px` y se moverá hacia abajo.

```
.container_blue {
        position: relative;
        left: 300px;
        top: 100px;
}
```

Además las propiedades `top`, `right`, `bottom` y `left` además de usar píxeles, pueden usar otras unidades como *porcentajes*, *em*, *rem*, *vw*, *vh*, entre otras.

Posiciones absolutas



Los valores absolutos nos permiten posicionar elementos en función a un elemento padre, siempre y cuando este tenga una posición distinta a *static*. En el caso de ser así el elemento con posición absoluta tomará como padre al documento y no al *viewport*.

Los elementos con posiciones absolutas se encuentran fuera del flujo normal del *HTML*, de modo que este no afecta a otros elementos que posicionados en el documento.

Para situar el contenido en estas posiciones usaremos las propiedades `top`, `right`, `bottom` y `left`.

Veamos un ejemplo de estas posiciones creando un `<div>` con clase `.position-relative` qué será el elemento padre de las dos cajas que creamos anteriormente. Luego agreguemos dentro del contenedor las cajas y por último escribamos una clase que contendrá el *position: absolute* llamado `.position-absolute`

```
<body>
        <h1>Soy una caja</h1>
        <div class="position-relative">
                <div class="container_blue position-absolute">
                        Lorem ipsum dolor sit amet, consectetur adipisicing
elit. Ea repellat veniam commodi aut ut perferendis alias rem natus facere.
Autem ex illo quam nisi, accusamus voluptates obcaecati vel dolore harum!
                </div>
                <div class="container_blue position-absolute">
                        Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
                </div>
        </div>
</body>
```

Ahora demos los estilos a la clase `.position-relative`. Agreguemos un ancho y un alto al contenedor de `1000px` de ancho y `1200px` de ancho. Después pondremos un `position:relative` y finalmente usaremos un `border: 5px solid black;` para ver el borde del elemento padre.

```
.position-relative {
        height: 1000px;
        width: 1200px;
        position: relative;
        border: 5px solid black;
}
```

Sigamos con la clase `.position-absolute` agregando la propiedad `position: absolute;`

```
.position-absolute {
        position: absolute;
}
```

Si guardamos y recargamos la página podremos ver que sólo se puede notar una sola caja, no obstante, la otra caja se encuentra debajo de esta. Esto que fenómeno ocurre debido a que aún no hemos configurado una posición específica para las cajas dejando a estas en la posición arriba izquierda del elemento padre.

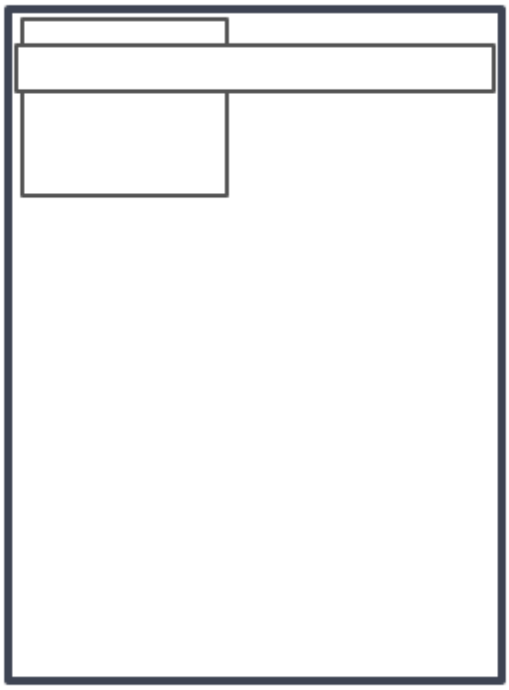
Si agregamos un párrafo con un texto preformateado debajo de la última caja podremos ver que el texto pasa por debajo de estas.

Para posicionar estas cajas usaremos una a modo de ejemplo las posiciones `top`, `right`, `bottom` y `left` en el inspector de elementos.

Dentro de la clase `.container_blue` agregaremos un `top: 130px;` y de manera inline pondremos un `left: 600px;` a la segundo caja para que no se superponga con la primera caja.

Cómo vemos la caja se movió hacia la derecha `600px` haciendo que la segunda caja se moviera.

Posiciones fijas



El último valor que veremos antes de usar las posiciones en nuestra maqueta serán las posiciones fijas.

Los *position: fixed* sirven para fijar contenido en función del *viewport*, o sea, de la ventana del navegador. Cuando usamos estás posiciones y hacemos scroll los elementos afectados por este valor quedarán fijos en el mismo lugar mientras los otros elementos se moverán normalmente.

Veamos un ejemplo de este particular valor usando el ejemplo anterior, pero esta vez no usaremos a las cajas, si no más bien crearemos un *header* usando el título principal de la página.

ingresemos al editor de texto y agreguemos un contenedor con una clase `.position-fixed` qué hará que se posicione el *header*. Luego agreguemos en este el título principal de la página.

```
<header class="position-fixed">
        <h1>Soy una caja</h1>
</header>
```

Ahora añadamos los estilos.

Primero comencemos con `<header>`. Iniciemos agregando un ancho para el contenedor. Como el posicionamiento se basa en el tamaño de la ventana del navegador, configuraremos su tamaño a su `100%`. Después le daremos un color negro de fondo y un color de fuente blanca para el contraste.

```
header {
        width: 100%;
        background-color: black;
        color: white;
}
```

Ahora, añadamos en `.position-fixed` la propiedad `position: fixed;`. Guardemos los cambios y revisemos los resultados en el navegador.

```
.position-fixed {
        position: fixed;
}
```

Si hacemos scroll veremos que el *header* queda fijo en la ventana del navegador por debajo de las cajas posicionadas con `absolute`. De igual manera es importante mencionar que esta posición no respeta el flujo normal del HTML, por lo tanto se superpondrá ante cualquier elemento anterior o posterior a ella.

Para dejar el título por arriba de la caja usaremos una propiedad llama *z-index*.

¿Qué es *z-index*?

Esta propiedad nos ayudará a definir si los elementos estarán encima o debajo de otros. Para hacerlo deberemos dar valores positivos para que estén arriba y negativos para que estén debajo.

Agreguemos un `z-index` que afecte al elemento `header` con un valor de `1`.

```
header {
        width: 100%;
        background-color: black;
        color: white;
        z-index: 1;
}
```

Como vemos el *header* se sobrepuso a la caja usando haciendo que está este encima de todos los elementos de la página. Por lo general fixed es usado para crear barras de navegación fija o toolbars que acompañan al usuario.

Aplicando posicionamiento en proyecto

Ahora que conocemos los tipos de valores que podremos usar para posicionar elementos dentro nuestras páginas comenzaremos a usarlos dentro de la maqueta de la entrada de blog.

Para ello iremos al mockup para conocer qué elementos de la interfaz se sobreponen.

Si nos fijamos veremos que todas las imágenes incluidas en la entrada tienen un *caption* que describe el contenido de la foto y que se superponen a esta imagen.

Para posicionar el *caption* de la imagen de esta manera deberemos hacer que el caption use al padre, que en este caso sería la etiqueta `<figure>`, para que se posicione en relación a este elemento.

En concreto, lo que haremos será agregar una clase que funcione como un estado, o sea, crear una clase llamada `.position-relative` que sirva posicione a todos los contenedores en *relative* a modo de transformar a este en el elemento padre y no al documento.

Agreguemos en el parcial `_base` la clase `.position-relative` a modo de organizar esta en un lugar que no difiera con componentes o partes del layout. Dentro incluiremos la propiedad `position:relative`.

```
.position-relative {
        position: relative;
}
```

Luego agregaremos esta clase en cada uno de los `<figure>` de la página.

```
<figure class="single-post__featured-image position-relative">
  <img src="assets/images/featured-image.jpg" alt="Vista de la cordillera de los
Andes vista desde Santiago">
  <figcaption class="image-caption">Vista de la cordillera de los Andes vista
desde Santiago.</figcaption>
</figure>
```

Ahora incluiremos los estilos visuales y de posición del *caption* de la imagen. Ingresemos al parcial `_image-caption` y agreguemos la clase `.image-caption`.

Dentro de esta clase definiremos en primer lugar los estilos visuales de este cómo los colores de fondo y el tipo y color de fuente que tiene el *caption*.

Agreguemos el fondo del *caption* con `backgroud-color` y con un color `$jet`. Sigamos con el color de la fuente que será de color `white` y el tipo de letra que tendrá será `$lato`.

```
.image-caption {
        background-color: $jet;
        font-family: $lato;
        color: $white;
}
```

Ahora que el *caption* tiene los estilos base posicionaremos arriba de la imagen usando un `position: absolute;`. Debajo de este pondremos un `z-index` con un valor positivo para que el caption quede arriba de la foto. Además subiremos el caption hacia arriba usando un `bottom: 4px`.

Si revisamos el resultado veremos que este se posicionó correctamente en la imagen.

```
.image-caption {
        position: absolute;
        bottom: 5px;
        z-index: 1;
        background-color: $jet;
        font-family: $lato;
        color: $white;
}
```

A este aún le falta un poco de relleno dentro, de modo que agregaremos un *padding* de `10px` para mejorar este aspecto.

Además de esto al *caption* le falta un poco de transparencia para que se vea por ella parte de la imagen, en otras palabras, usaremos una propiedad que crea este efecto visual llamada `opacity`.

La propiedad `opacity` nos ayudará a darle transparencia al fondo de un elemento especificado. los valores usados son de `1` - sin opacidad - hasta `0.1` - transparencia casi total -.

En este caso nosotros deseamos que nuestro *caption* se vea sólo un poco de la imagen, de modo que usaremos un valor de `.8`.

```
.image-caption {
        padding: 10px;
        position: absolute;
        bottom: 5px;
        z-index: 1;
        background-color: $jet;
        opacity: .8;
        font-family: $lato;
        color: $white;
}
```

Si vemos ahora el *caption* veremos que está casi listo.

Lo que nos falta por arreglar es el tamaño total del *caption*. En este caso queremos que nuestro caption tenga el tamaño completo del contenedor, esto se traduce al 100%, así que agregaremos un `width` del `100%` para lograrlo.

Si recargamos veremos que el *caption* tomó el 100% del tamaño del contenedor, el cual es su padre, pero la imagen es más grande que el tamaño total de `<figure>`, para solucionar esto haremos que todas las imágenes máximo un 100% del lugar en donde esten contenidas.

Para eso iremos al parcial `_reset` y pondremos en él un `max-width` de `100%`.

```
//_reset.scss
img {
        max-width: 100%;
}
```

```
//_image-caption.scss
.image-caption {
        width: 100%;
        padding: 10px;
        position: absolute;
        bottom: 5px;
        z-index: 1;
        background-color: $jet;
        opacity: .8;
        font-family: $lato;
        color: $white;
}
```

Sí recargamos veremos que la imagen tomo el `100%` del `<figure>`, pero aún no queda bien el *caption* con la imagen. El problema que ocurre en este caso es debido a que el total del *caption* es mayor al contenedor, debido a que el *padding* aumenta el tamaño total del elemento. La solución a este problema es simple, sólo debemos agregar un `box-sizing: border-box;` y el problema se solucionará, ya que el tamaño total se sumará automáticamente a la caja.

```
.image-caption {
        width: 100%;
        padding: 10px;
        box-sizing: border-box;
        position: absolute;
        bottom: 5px;
        z-index: 1;
        background-color: $jet;
        opacity: .8;
        font-family: $lato;
        color: $white;
}
```

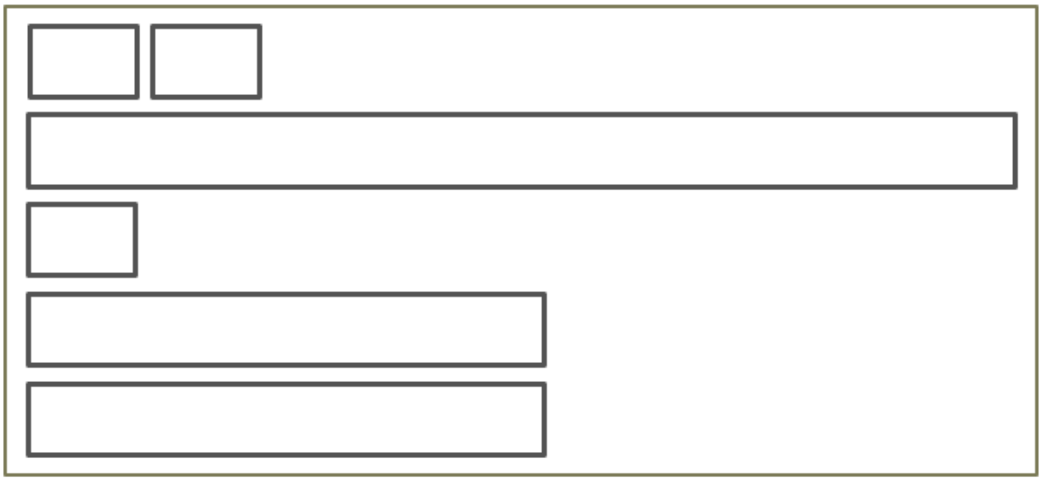
Guardemos y revisemos en el navegador el resultado.

Ya nos estamos acercando a terminar de traducir la representación visual entregada por nuestro equipo. Sólo nos falta el *navbar (barra de navegación)* y el *footer*.

Elementos flotantes

Para poder darle estilos a estos elementos deberemos conocer una propiedad de posicionamiento llamada *float*.

Para comprender mejor esto revisaremos nuevamente cómo funciona el dibujado de una página web.



Cuando tenemos elementos *inline* estos van apareciendo ocupando el mínimo espacio posible y aparecen uno al lado del otro. Cuando por ejemplo ponemos varias imágenes y estas no son muy grandes aparece una al lado de la otra.

Sin embargo si pongo una imagen y después agrego un párrafo, ese párrafo es un bloque, y aparece en un nuevo espacio, abajo, ocupando el `100%`.

Si pongo un inline debajo de un párrafo, el elemento *inline* aparecerá debajo del párrafo porque los elementos de bloque incluyen este quiebre o salto de línea, entonces automáticamente va a empujar al próximo *inline* hacia abajo, no sólo porque el bloque ocupa un `100%`, si no por que además el bloque automáticamente quiebra el contenido generando esta nueva línea.

Entonces sí agregamos un bloque de un `40%` de ancho, que no va a ocupar todo el espacio y si luego agregamos un inline o un bloque, independiente del tamaño el bloque quedará abajo.

Para poder posicionar un elemento al lado del otro podremos usar una propiedad de *CSS* llamada `float`

¿Qué es *float*?

La propiedad *float* nos permite sacar un elemento del flujo normal y llevarlo a la izquierda o derecha lo que nos permitirá posicionar mejor nuestro contenido.

En el caso anterior de dibujado, al principio se encontraba dos elementos *inline*, luego un bloque que ocupaba el `100%` , el que dío un quiebre de línea al siguiente elemento *inline* y finalmente había un bloque con un `40%` de ancho y otro con `50%` de ancho.

Si agregamos *float* en estos dos bloques los elementos quedarán uno al lado del otro y de esa forma podremos diagramar hacia el lado.

Ejemplo con *float*

Para ver de mejor manera el efecto de la propiedad *float* haremos un ejemplo reutilizando las cajas que nos han acompañado durante toda esta experiencia.

Para ello vamos a devolver todos elementos del documento al flujo normal de dibujado quitando las clases relacionadas a posicionamiento. Con esto los elementos volverán a tener las características del principio.

Ahora para probar cómo funciona la propiedad *float* haremos que las cajas se posicionen en la misma línea sacando a estos de su flujo normal.

Hacerlo es simple, sólo debemos agregar la propiedad `float` con valor `left` dentro de la clase `.container_blue`.

```
.container_blue {
        float: left;
        width: 500px;
        height: 500px;
        border: 10px solid coral;
        padding: 30px 20px;
        margin: 10px 20px 30px 40px;
        background-color: blue;
        color: white;
}
```

Si recargamos la página veremos que las dos cajas se posicionaron una al lado de la otra, logrando el efecto esperado, no obstante, al lado de la segunda caja los párrafos que se encontraban debajo quedaron a lado de este elemento. Esto se debe a que floats modifica el flujo normal de renderizado de los elementos haciendo que algunos elementos se afecten.

Para devolver el flujo a los elementos luego de hacer flotaciones debemos usar una propiedad llamada `clear`.

¿Qué es la propiedad *clear*?

`clear` nos permitirá limpiar un elemento que se encuentra afectado por un *float*. Las opciones que tendremos para limpiar dependerán del lugar en el cual se encuentre el elemento *float*. Los valores que podremos usar son `left`, `right` o `both`.

Limpiemos el flujo de los párrafos creando en el primer párrafo una clase llamada `.clear`. Cuando terminemos dentro de la etiqueta `<style>` agreguemos la clase `.clear` y dentro de ella escribamos la propiedad `clear`.

Es importante notar la dirección en la que limpiaremos los elementos, en este caso las flotaciones se encuentran hacia la izquierda, de modo que podríamos usar el valor `left` o `both`, o sea limpiar a la izquierda y derecha.

Teniendo claro esto, agreguemos el valor `left` a la propiedad `clear`.

```
<p class="clear">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Minus
voluptas ex eveniet laudantium iure aliquam, explicabo sed. Consequatur nihil
quaerat nulla esse, cupiditate corrupti sit enim odio, illo tenetur quia.</p>
```

```
.clear {
        clear: left;
}
```

Si recargamos la ventana podremos ver que los párrafos volvieron al flujo normal de renderizado.

Aplicando *float* en Maqueta

Ahora que ya conocemos como flotar elementos, es momento de traducir todo este conocimiento a nuestra maqueta, específicamente en las secciones del layout que faltan, o sea, la barra de navegación y el *footer*.

Comencemos por darle los estilos base a la barra de navegación. Esta como sabemos se encuentra contenida por el bloque `header`. Este bloque ya se encuentra con las dimensiones y estilos base, los que fueron realizados por el maquetador anterior.

Al interior de ella veremos una etiqueta `nav` que contiene a los elementos que componen a esta barra de navegación, los que se dividen a su vez en el logo y la navegación.

```
<header class="header">
        <nav class="navbar">
                <div class="navbar__logo">
                        <a href="#"><img src="assets/images/logo.png" alt="Logo
principal de Monttaña tour operador" class=""></a>
                </div>
                <div class="navbar__container">
                        <ul class="navbar__list">
                                <li><a href="#" class="link_primary">Tours</a>
</li>
                                <li><a href="#"
class="link_primary">Nosotros</a></li>
                                <li><a href="#" class="link_primary">Blog</a>
</li>
                                <li><a href="https://www.w3schools.com/jquery/"
class="link_primary">Contacto</a></li>
                        </ul>
                </div>
        </nav>
</header>
```

En la clase de bloque `.navbar` no agregaremos ningún valor para posicionar o dimensionar su espacio ya que `<header>` ya tiene dimensiones configuradas.

En cuanto a los contenedores del logo y la navegación haremos que estos floten uno por sobre el otro. Agreguemos la clase `.navbar__logo` con un `float:left;`. Hagamos lo mismo con la clase `.navbar_container`.

```
.navbar__logo {
        float: left;
}

.navbar__container {
        float: left;
}
```

Configuremos ahora el tamaño del logo usando *nesting*. Dentro de la clase `.navbar__logo` agreguemos el elemento `img` y en él pongamos un ancho `90px`.

```
.navbar__logo {
        float: left;

        img {
                width: 90px;
        }
}
```

Sigamos con los estilos del navegador. Para ello primero definiremos un tamaño en su interior a modo de mover los ítems de la navegación a la derecha, como aparece en el mockup. Le daremos un ancho de `900px` y alinearemos la navegación hacia la derecha.

```
.navbar__container {
        float: left;
        width: 900px;
        text-align: right;
}
```

Luego usando nesting le daremos un `display: inline-block` a `<ul>`, a modo de cambiar el comportamiento de modificar el comportamiento de bloque que viene por defecto en esta etiqueta. Además, cambiaremos el tipo de fuente de la navegación por lato.

```
ul {
        display: inline-block;
        font-family: $lato;
}
```

Continuemos con los ítems de la navegación, los cuáles deberemos flotar para conseguir que estén de manera horizontal dentro de la barra. Luego le daremos un poco de padding a la derecha de `10px`. Finalmente, para quitar el bullet que lleva cada uno de los item de la navegación, usaremos la propiedad `list-style-type` con un valor `none`.

```
li {
        float: left;
        padding-right: 10px;
        list-style-type: none;
}
```

Dentro este `<li>` se encuentra un link que forma parte del item. Para darle los estilos a esta etiqueta anidaremos nuevamente (*nesting*).

Agreguemos el selector `a` y luego quitemos la línea que sigue al link usando la propiedad `text-decoration`. El valor que tendrá será `none`.

Seguido agregaremos la propiedad `text-transform` la que nos permitirá transformar el texto en otro tipo como por ejemplo, un texto capitalizado, con mayúsculas o minúsculas. En este caso el valor será con mayúsculas, o sea, `uppercase`.

```
li {
        float: left;
        padding-right: 10px;
        list-style-type: none;

        a {
                text-decoration: none;
                text-transform: uppercase;
        }
}
```

Si recargamos la página podremos ver el resultado final de la barra de navegación, la cual se ve similar a la vista en el mockup.

TOURS NOSOTROS BLOG CONTACTO

Con respecto al *footer* del sitio, este está compuesto por una el bloque principal `footer`, el cual contiene en su interior dos contenedores con clase `.footer__container`, los que contienen a su vez, el logo y copyright del sitio y la información de contacto de la empresa.

```
<footer>
        <div class="footer__container">
                <div class="footer__logo">
                        <a href="#"><img src="assets/images/logo-footer.png"
alt="Logo Monttaña tour operador"></a>
                </div>
                <small>Monttaña Tour Operador Limitada, &copy; Todos los
derechos reservados 2018.</small>
        </div>
        <div class="footer__container">
                <address class="footer__contact-info">
                        <p><span class="footer_text-emphasis">Dirección:</span>
Manuel Montt 007, Providencia, Región Metropolitana, Chile.</p>
                        <p><span class="footer_text-emphasis">Teléfono:</span>
<a href="tel:+56 98007123123">+56 9 8 007 123 123</a></p>
                        <p><span class="footer_text-emphasis">Correo:</span> <a
href="mailto:hola@tourmonttana.cl"> hola@tourmonttana.cl</a></p>
                </address>
        </div>
</footer>
```

Comencemos a darle los estilos principales al *footer* agregando el elemento `footer` dentro del parcial `_footer`. En su interior le daremos un alto de `100px` y un padding de `50px` para dimensionarlo. Ahora le daremos un color de fondo `$jet` y un color de fuente `$white`.

```
footer {
        height: 100px;
        padding: 50px;
        background-color: $jet;
        color: $white;
}
```

Seguiremos con los estilos del contenedor, con los cuáles posicionaremos a su izquierda usando la propiedad `float: left`. Además demos un poco de relleno a los contenedores usando un `padding` de `0` de alto y `30px` de ancho y cambiaremos el tipo de fuente de bloque usando un `font-family: $lato;`. Además, usando nesting le daremos estilos a la etiqueta `<small>` un tamaño de fuente extra pequeño y transformaremos el copyright a mayúsculas con `text-transform: uppercase;`.

```
.footer__container {
        float: left;
        padding: 0 30px;
        font-family: $lato;

        small {
                font-size: $x-small;
                text-transform: uppercase;
        }
}
```

Con esto listo, le daremos un `padding-bottom` de `15px` a la clase `.footer__logo`, así como también reduciremos el tamaño del logo usando un `width: 300px`.

```
.footer__logo {
        padding-bottom: 15px;

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

Ahora demos los estilos al otro contenedor en donde se encuentra la información de contacto de la empresa.

Para poder dar los estilos específicos a estos elementos deberemos sobreescribir algunas propiedades que vienen heredadas, como por ejemplo la cursiva que llevan las tipografías los colores de los links.

Comencemos agregando dentro de la clase `.footer__contact-info` otro selector que afecte al párrafo de este contenedor, y dentro de este pondremos un `margin: 0;` para resetear las dimensiones heredadas del párrafo. Luego de eso para separar cada uno de los párrafos usaremos un `padding` con `10px`.

Cambiemos también la fuente heredada por los párrafos usando la variable `$lato`. Asimismo quitemos el estilo cursiva de las tipografías usando la propiedad `font-style: normal;` y finalmente cambiemos el color de las fuentes a un color `$white`.

```
.footer__contact-info {
        p {
                margin: 0;
                padding-bottom: 10px;
                font-family: $lato;
                font-style: normal;
                color: $white;
        }
}
```

Con estos estilos ya casi estamos terminando la maqueta que nos encomendaron. Lo que nos falta agregar son los colores que componen a los links del *footer* y aumentar el grosor del título de la información.

Agreguemos debajo del selector de párrafo el selector `a` el cual contendrá los colores de los links. El color en estado normal será `$coral` y el color al pasar por arriba de él será `$nickel`.

```
a {
        color: $coral;

        &:hover {
                color: $nickel;
        }
}
```

Finalmente, le daremos énfasis a los títulos de información usando un `font-weight` con un valor de `$bold` y para separar el título de la información le daremos un poco de `padding`. `5px` esta bien.

```
.footer_text-emphasis {
        padding-right: 3px;
        font-weight: $bold;
}
```

Si guardamos los estilos y los revisamos en el navegador podremos ver la traducción del mockup ha finalizado.

Layout estático v/s fluído

Aunque parezca que hayamos terminado nuestra maqueta y que esta se vea genial, aún nos falta resolver un tema muy importante a la hora crear maquetas.

Desde hace un buen tiempo hasta ahora el uso de dispositivos móviles ha aumentado al nivel de superar a los computadores personales haciendo que los tamaños de pantalla sean variados.

Por ejemplo, el contenido de un sitio dispuesto en un smartphone se verá diferente al verlo en un computador.

Así mismo las resoluciones de pantalla también conllevan un problema a la hora de definir una resolución exacta, ya que existen casos en que las pantallas de dispositivos móviles tienen mayor resolución que dispositivos de escritorio.

Esto en concreto crea un gran problema a la hora de decidir cuál será el tamaño adecuado para el diseño de la maqueta, sin embargo existen estrategias que nos ayudarán a construir maquetas que se adecúen a a estas diferencias de tamaño y resoluciones.

Estas estrategias hacen relación al tipo de unidad de medida que contiene el layout, o sea, si creamos un diseño con unidades de medida absolutas como los píxeles obtendremos un diseño estático que se mantendrá fijo y si usamos unidades de medidas relativas como los porcentajes, obtendremos cajas que serán relativas al tamaño del *viewport* del navegador, de modo que es importante identificar el tipo de diseño que usaremos.

Los tipos de diseño más básicos que podremos crear son los:

Diseños estáticos:

En este tipo de diseño los elementos que lo componen tienen un ancho fijo que no varía y se mantiene fijo aún cuando reduzca el tamaño del *viewport*.

Por lo general este tipo de diseño esta construido pensando en un tamaño en específico. Sin ir mas lejos, antes de que aparecieran los smartphone los diseños de los sitios web eran totalmente estáticos, definidos por un ancho especificado en el `<body>`.

De hecho, si miramos el parcial `_base` podremos ver que el maquetador fijó un tamaño de `1280px` de ancho. Esto hizo que el diseño de la maqueta fuera estática. Si movemos reducimos la ventana del navegador veremos que aparece un scroll horizontal en la ventana. El que aparezca este tipo de scroll es un indicio de este tipo de diseño.

Ventajas de los diseños estáticos

Las ventajas que tiene este tipo de diseño son:

- El control del tamaño y posición de los elementos es más preciso.
    - Ya que sabremos realmente la cantidad de píxeles tendrán.
- El control de la cantidad de espacio usado por el contenido.
    - Al igual que el punto anterior, tendremos un control preciso de cuánto espacio existe entre un elemento y otro.
- El control del tamaño de las imágenes.
    - Bajo el mismo contexto, sabremos cual es el tamaño exacto de las imágenes.

Desventajas de los diseños estáticos

Las desventajas que tienen este tipo de diseño son:

- Problemas si el usuario tiene un dispositivo con mayor resolución a la definida por el maquetador.
    - Si ocurre el usuario verá el contenido pequeño haciendo que la usabilidad del sitio baje.
- Problemas si el usuario aumenta el tamaño de la fuente usando el navegador.
    - Al usar unidades estáticas el texto se mantendrá fijo, haciendo que estos no puedan ser modificados por el usuario, bajando la funcionalidad de la página.
- La página será totalmente usable sólo si el usuario navega en un dispositivo con el mismo tamaño al definido por el maquetador.

- Si por el contrario el dispositivo del usuario tiene una resolución mayor o menor a la definida existirán problemas en la legibilidad del contenido.

Diseños Fluidos

Los diseños fluidos o líquidos son aquellos que usan unidades de medidas relativas como lo porcentajes para estirarse o contraerse dependiendo del tamaño del *viewport*.

En el caso de nuestra maqueta si quisiéramos hacerla fluida deberemos agregar unidades relativas como *porcentajes*, *em* o *rem*.

Probemos esto eliminando del `<body>` el ancho fijo que configuró el maquetador anterior.

```
/* Base */
.position-relative {
        position: relative;
}
```

Sí recargamos y reducimos el tamaño del *viewport* veremos que el contenido fluye en base al tamaño del *viewport*. Asimismo si aumentamos el zoom de la página podremos distinguir que las fuentes aumentan su tamaño sin problemas y sin agregar un *scroll* horizontal.

Otro punto interesante de este tipo de diseño esta relacionado a las imágenes, ya que si estas se las configura con un tamaño fijo, aún cuando el diseño sea fluido estas no cambiarán sus dimensiones.

Veamos esto efecto comentando el `max-width` qué dejamos en el parcial `_reset`.

```
// img {
//              max-width: 100%;
// }
```

Si recargamos la página veremos que las imágenes se desbordaron y quedaron fuera del diseño. Esto se debe a que las imágenes tienen un tamaño absoluto, lo que es contrario a un diseño fluido.

Para solucionar este problema deberemos darle un tamaño que cubra un `100%`. En el caso de las imágenes de la maqueta serán contenidas por la etiqueta `<figure>`.

Descomentemos el `max-width: 100;` y eliminemos los tamaños agregados en el parcial `_images`.

```
img {
        max-width: 100%;
}
```

Si recargamos el navegador veremos que las imágenes fluyen en base al *viewport*.

Antes de terminar de conocer estos diseños es importante notar las ventajas y desventajas de este tipo de diseño:

Ventajas de los diseños fluidos

Algunas ventajas de los diseños fluidos son:

- El contenido del diseño se ajustará al tamaño del *viewport*.

    - O sea, que el contenido se contraerá si la pantalla es de un dispositivo móvil y no se agregará un scroll horizontal como si lo hacen los diseños estáticos.
- El diseño puede ser configurable por el usuario mejorando la usabilidad del sitio.
    - Esto en la práctica tiene que ver con el tamaño de la fuente.

Desventajas de lo diseños fluidos

Algunas desventajas de los diseños fluidos son:

- Si el usuario tiene pantalla ancha el texto se verá largo.
    - Esto provocará que el texto sea poco legible.
- Si tenemos una imagen con tamaño fijo esta se desbordará del diseño provocando que la diagramación no funcione.

El entender cómo funcionan estos dos tipos de diseños hará que la elección de las unidades sea base al tipo de layout que requiere la representación visual. Esto a grandes rasgos nos permitirá crear mejores elementos o cajas con tamaños absolutos estos no fluirán en base al *viewport*.

Como vemos el trabajar con uno u otro diseño conlleva entender cómo funcionan estos, asó como también entender cuales son sus ventajas y desventajas a modo de usarlos de manera adecuada en cualquier tipo de maquetas.