Css

Display

`display` es la propiedad más importante para controlar estructuras. Cada elemento tiene un valor de display por defecto dependiendo de qué tipo de elemento sea. El valor por defecto para la mayoría de los elementos es usualmente `block` (de bloque) o `inline` (en línea). Un elemento que es block es comunmente llamado elemento block-level. Un elemento inline siempre es llamado elemento inline.

block

inline

El `span` es el elemento inline estándar. Un elemento inline puede contener algo de texto dentro de un párrafo como esto sin interrumpir el flujo del párrafo. El elemento `a` es el elemento inline más común, ya que se usa para links.

none

Otro valor común de display es `none`. Algunos elementos especializados como `script` usan este por defecto. Es comúnmente usado en JavaScript para ocultar o mostrar elementos sin eliminarlos ni recrearlos.

Esto es diferente de `visibility`. Usar `display: none` no dejará espacio donde el elemento se encontraba, pero `visibility: hidden;` dejará un espacio vacío.

otros valores de display

Hay bastantes valores exóticos de display, como `list-item` y `table`. Aquí hay una lista exhaustiva. Discutiremos `inline-block` y `flex` después.

```
/ * Valores <display-outside> * /

display: block;
display: inline;
display: run-in;

/ * Valores <display-inside> * /
display: flow;
display: flow-root;
display: table;
display: flex;
display: grid;
display: ruby;
display: subgrid;

/ * Valores <display-outside> más valores <display-inside> * /
display: block flow;
display: inline table;
display: flex run-in;
```

```
/ * Valores <display-listitem> * /
display: list-item;
display: list-item block;
display: list-item inline;
display: list-item flow;
display: list-item flow-root;
display: list-item block flow;
display: list-item block flow-root;
display: flow list-item block;

/ * Valores <display-internal> * /
display: table-row-group;
display: table-header-group;
display: table-footer-group;
display: table-row;
display: table-cell;
display: table-column-group;
display: table-column;
display: table-caption;
display: ruby-base;
display: ruby-text;
display: ruby-base-container;
display: ruby-text-container;

/ * Valores <display-box> * /
display: contents;
display: none;

/ * Valores <display-legacy> * /
display: inline-block;
display: inline-table;
display: inline-flex;
display: inline-grid;

/ * Valores globales * /
display: heredar;
display: initial;
display: unset;
```

```css
#main {
  width: 600px;
  margin: 0 auto;
}
```

```
<div id="main">
  Ajustar la propiedad width de un elemento block-level lo previene de estirarse hasta los bordes de su contenedor a la izquierda y derecha. Después puedes establecer la propiedad margin left y right para que sea auto y centrar de manera horizontal ese elemento en su contenedor. El elemento tomará el ancho que especificaste, después el espacio sobrante será dividido equitativamente entre los dos márgenes.

  El único problema ocurre cuando el ancho del navegador es menor que el de tu elemento. El navegador resuelve esto creando una barra de desplazamiento (scrollbar) horizontal en la página. Arreglemos esa situación...
</div>
```

Border

Ya que estamos hablando de la propiedad width, deberíamos hablar del *box model* (modelo de caja). Cuando ajustas la propiedad width de un elemento, este puede parecer mas grande de lo que es: la propiedad border (borde) del elemento y padding (relleno) estirarán el elemento más allá del ancho especificado. Mira el siguiente ejemplo, donde dos elementos con el mismo valor width terminan teniendo diferente tamaño al final.

```css
.simple {
  width: 500px;
  margin: 20px auto;
}

.fancy {
  width: 500px;
  margin: 20px auto;
  padding: 50px;
  border-width: 10px;
}
```

```
<div class="simple">
  Soy más chico...
</div>
```

```
<div class="fancy">
  ¡Y yo soy más grande!
</div>
```

Durante generaciones, la gente se ha dado cuenta de que las matemáticas no son divertidas, así que una nueva propiedad CSS llamada `box-sizing` fue creada. Cuando ajustas un elemento con `box-sizing: border-box;`, el padding y border de ese elemento no incrementan su ancho. Aquí está el mismo ejemplo de la página anterior, pero usando `box-sizing: border-box;` en los dos elementos:

```css
.simple {
  width: 500px;
  margin: 20px auto;
  -webkit-box-sizing: border-box;
     -moz-box-sizing: border-box;
          box-sizing: border-box;
}

.fancy {
  width: 500px;
  margin: 20px auto;
  padding: 50px;
  border: solid blue 10px;
  -webkit-box-sizing: border-box;
     -moz-box-sizing: border-box;
          box-sizing: border-box;
}
```

```
<div class="simple">
  ¡Ahora somos del mismo tamaño!
</div>
```

```
<div class="fancy">

        ¡Hurra!

</div>
```

Ya que esto es mucho mejor, algunos autores quieren que todos los elementos de sus páginas trabajen de la misma manera. Estos autores ponen lo siguiente en sus páginas:

```css
* {
  -webkit-box-sizing: border-box;
     -moz-box-sizing: border-box;
          box-sizing: border-box;
}
```

Esto asegura que el tamaño de todos los elementos siempre será modificado de la manera más intuitiva.

Ya que `box-sizing` es bastante nuevo, te recomiendo usar los prefijos `-webkit-` y `-moz-` por ahora, como yo los uso en el ejemplo. Esta técnica permite funciones experimentales en navegadores específicos. También, ten en cuenta que es compatible con IE8+.

Para poder tener estructuras mas complejas, tenemos que discutir la propiedad `position` (posición). Esta propiedad tiene un montón de posibles valores, sus nombres no tienen sentido y son casi imposibles de memorizar. Veámoslos uno a uno, pero antes deberías guardar esta página en tus marcadores para futuras referencias.

Static

```css
.static {
  position: static;
}
```

---

relative

```css
.relative1 {
  position: relative;
}
.relative2 {
  position: relative;
  top: -20px;
  left: 20px;
  background-color: white;
  width: 500px;
}
```

```
<div class="relative1">
  relative (relativo) se comporta de la misma manera que static a menos que le agregues otras propiedades.
  <div class="relative2">
    Establecer las propiedades top, right, bottom, y left de un elemento con position: relative; causará que su posición normal se reajuste. Otro contenido no se puede ajustar para adaptarse a cualquier hueco dejado por el elemento.
  </div>
</div>
```

Fixed

```
<div class="fixed">

Hola! No me prestes atención todavía

</div>
```

Un elemento fixed (fijo) se posiciona a la ventana del navegador de manera relativa, lo que significa que se mantendrá en el mismo lugar incluso después de hacer scroll en la página. Al igual que con `relative`, las propiedades `top`, `right`, `bottom`, y `left` también son usadas.

Estoy seguro que ya notaste el elemento fijo en la parte de abajo a la derecha de la página. Te doy permiso de que lo veas ahora. Aquí está el CSS que lo posiciona donde está:

```css
.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 200px;
  background-color: white;
}
```

Un elemento con valor fixed no deja espacio en el lugar de la página donde estaba ubicado normalmente.

Los buscadores móviles, sorprendentemente, no tienen muy buen soporte para el valor `fixed`. Aprende más al respecto aquí.

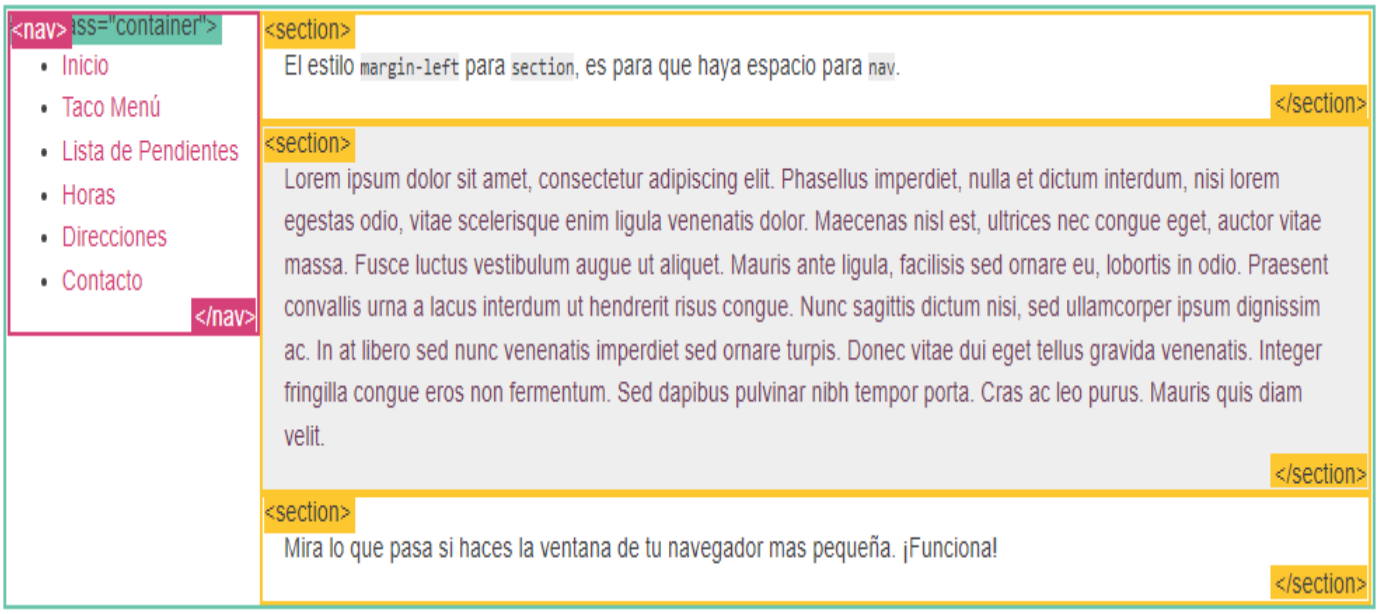
absolute

`absolute` (absoluto) es el valor más mañoso. `absolute` se comporta como `fixed` pero es relativo a *su ancestro posicionado más cercano* en lugar de ser relativo a la ventana del navegador. Si un elemento con `position: absolute;` no tiene ancestros posicionados, usará el elemento body del documento, y se seguirá moviendo al hacer scroll en la página. Recuerda, un elemento "posicionado" es aquel cuyo valor es cualquiera excepto `static`.

```css
.relative {
  position: relative;
  width: 600px;
  height: 400px;
}
.absolute {
  position: absolute;
  top: 120px;
  right: 0;
  width: 300px;
  height: 200px;
}
```

Todo esto de `position` tal vez tenga un poco más de sentido con un ejemplo práctico. Abajo está la estructura real de una página.

```css
.container {
  position: relative;
}
nav {
  position: absolute;
  left: 0px;
  width: 200px;
}
section {
  /* position is static by default */
  margin-left: 200px;
}
footer {
  position: fixed;
  bottom: 0;
  left: 0;
  height: 70px;
  background-color: white;
  width: 100%;
}
body {
  margin-bottom: 120px;
}
```



Este ejemplo funciona porque el contenedor es más alto que el nav. Si no lo fuera, el nav se saldría de su contenedor. En las páginas que vienen a continuación discutiremos otras técnicas para estructurar y hablaremos de sus pros y sus contras.

Float

Otra propiedad CSS para estructurar es `float` (flotar). Float se usa para envolver imágenes con texto, como aquí:

```css
img {
  float: right;
  margin: 0 0 1em 1em;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.

Clear

La propiedad `clear` (despejar) es importante para controlar el comportamiento de los floats. Compara estos dos ejemplos:

```
<section>
  <div class="box">
    ¡Siento que estoy
    flotando!
  </div>
  En este caso, el elemento section está después del div. Sin embargo, ya que el div está flotado a la izquierda, esto es lo que pasa: el texto en el elemento section flota alrededor del div y el elemento section rodea todo. ¿Qué pasaría si quisiéramos que section apareciera despues del elemento que está flotando?
</section>
```

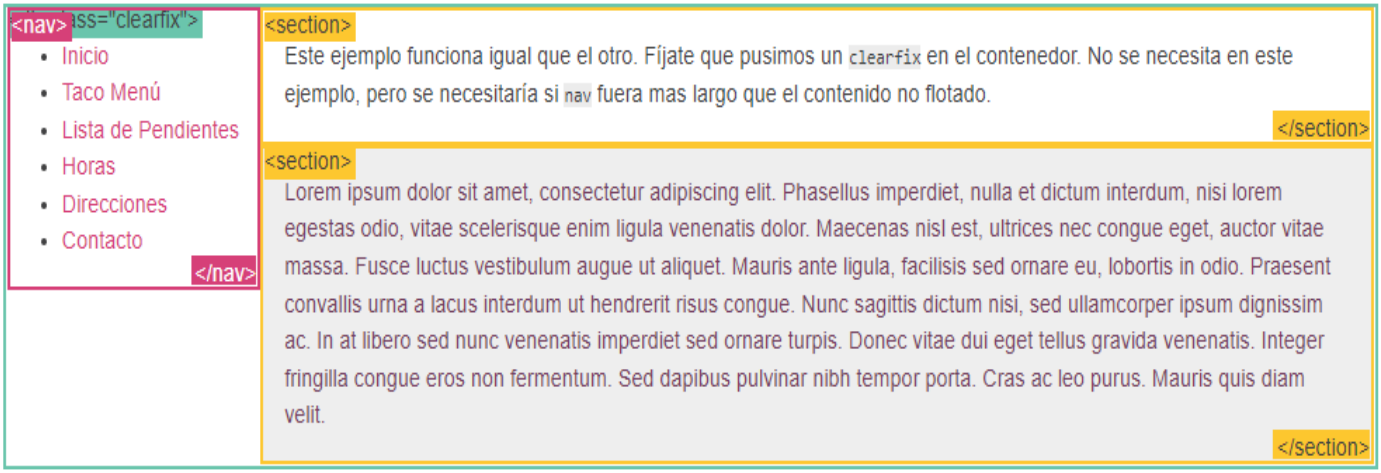
```css
.box {
  float: left;
  width: 200px;
  height: 100px;
  margin: 1em;
}
.after-box {
  clear: left;
}
```

```
<div class="box">
  ¡Siento que estoy
  flotando!
</div>

<section class="after-box">
  Usando clear hemos movido esta sección hacia abajo del div que está flotando. Para despejar elementos que están flotando a la izquierda debes usar el valor left. También puedes despejar hacia la derecha usando right y hacia ambos lados usando both.
</section>
```

Es muy común hacer estructuras completas usando `float`. Aquí está la misma estructura que usamos con `position` hace rato, pero usando `float`.

```css
nav {
  float: left;
  width: 200px;
}
section {
  margin-left: 200px;
}
```



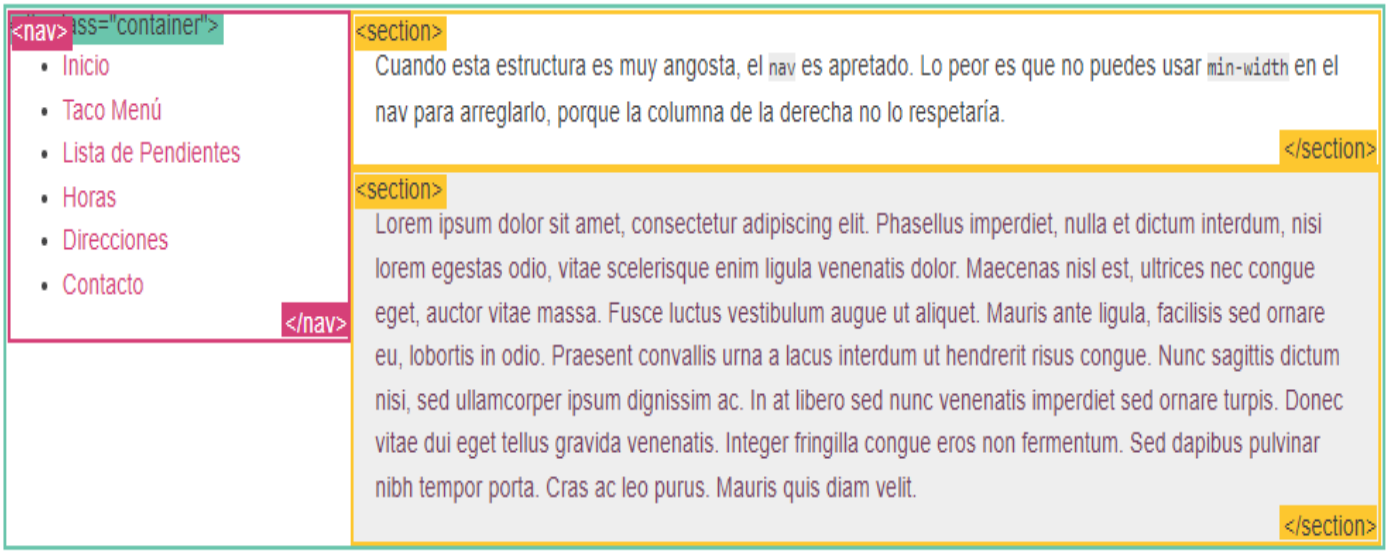
Porcentaje

Porcentaje es una unidad de medida relativa al bloque contenedor. Es muy útil para imágenes: aquí tenemos una imagen que ocupa el 50% de su contenedor en todo momento. ¡Haz la página mas pequeña para que veas lo que pasa!

```css
article img {
  float: right;
  width: 50%;
}
```

estructuras con porcentaje de ancho

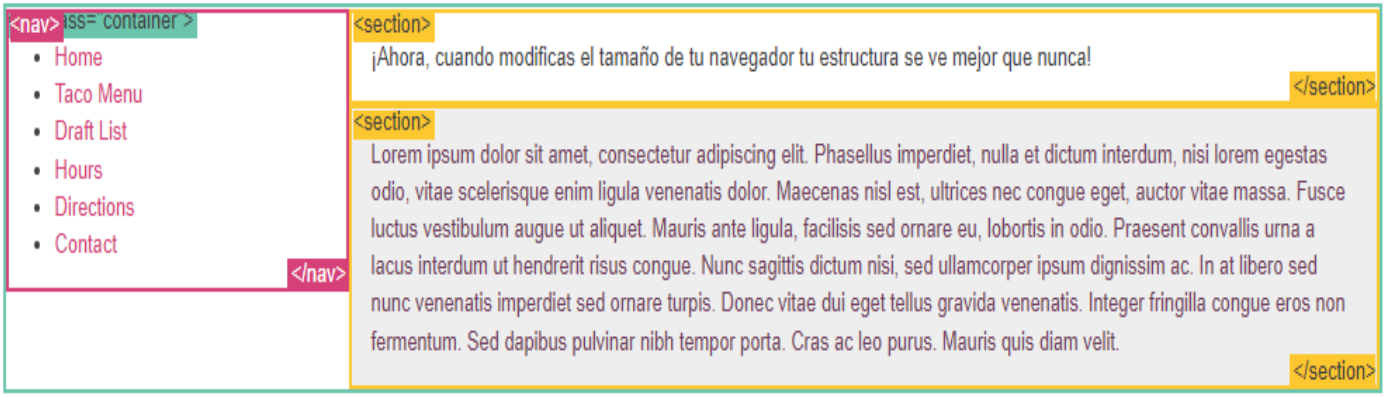
Puedes usar porcentaje para estructuras, pero esto puede requerir más trabajo. En este ejemplo, el contendido de `nav` empieza a desenvolverse de una forma desagradable cuando la ventana es muy angosta. Así que todo se reduce a lo que funcione mejor en tu sitio.



"Responsive Design" (Diseño Responsivo) es la estrategia para hacer que un sitio "responda" al navegador y dispositivo en el que se muestra... haciendo que se vea increible pase lo que pase.

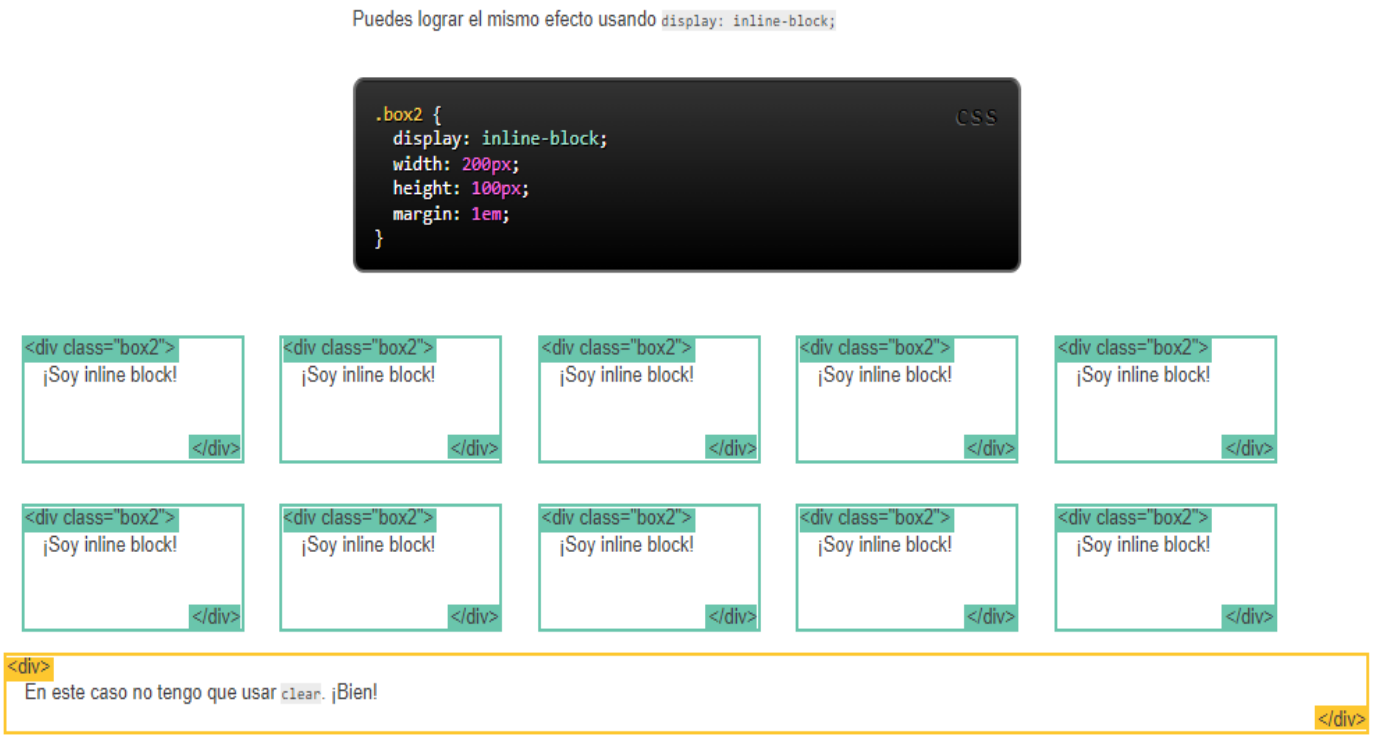
Los media queries son la herramienta más poderosa para hacer esto. Tomemos la estructura donde usamos porcentajes y transformémosla en una columna cuando la ventana del navegador es muy pequeña para mantener el menú en la barra lateral:

```css
@media (min-width:600px) {
  nav {
    float: left;
    width: 25%;
  }
  section {
    margin-left: 25%;
  }
}
@media (max-width:599px) {
  nav li {
    display: inline;
  }
}
```



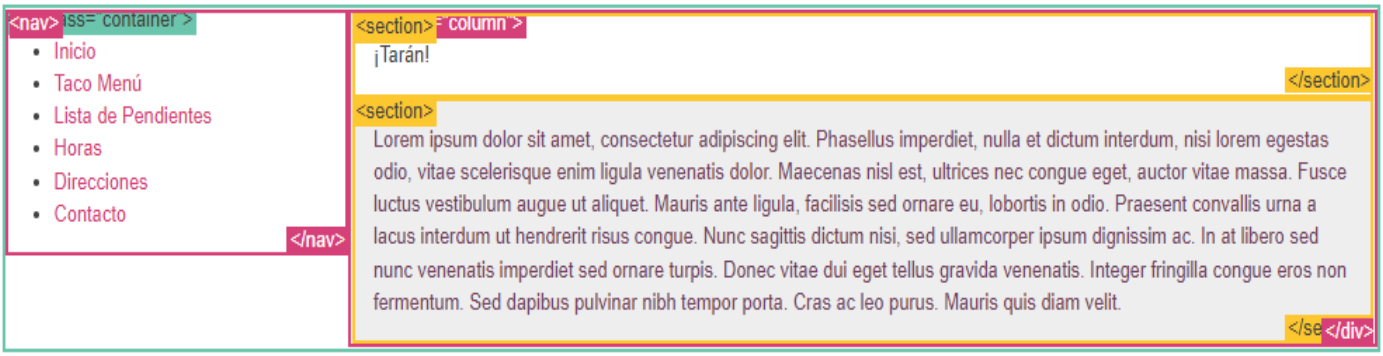
Inline-Block

Puedes crear una cuadrícula de cajas que llene el navegador armoniosamente. Esto ha sido posible por mucho tiempo usando `float`, pero ahora con `inline-block` es aún más fácil. Veamos ejemplos con las dos opciones.



También puedes usar el valor `inline-block` para estructurar un sitio, pero hay algunas cosas que debes tener en cuenta:

- Los elementos `inline-block` son influenciados por la propiedad `vertical-align`, la cual quizas quieras establecer como `top`.
- Necesitas establecer el ancho de cada columna.
- Si hay un espacio en blanco entre columnas en tu HTML, habrá un vacío entre ellas.



Multiples Columnas

Hay un nuevo set de propiedades CSS que te permiten tener múltiples columnas de texto. Echa un vistazo:

```css
.three-column {
  padding: 1em;
  -moz-column-count: 3;
  -moz-column-gap: 1em;
  -webkit-column-count: 3;
  -webkit-column-gap: 1em;
  column-count: 3;
  column-gap: 1em;
}
```

```
<div class="three-column">
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.
</div>
```

Flexbox

La nueva estructura con `flexbox` está lista para redefinir la manera en la que hacemos estructuras con CSS. Desafortunadamente las especificaciones han cambiado bastante recientemente y su implementación es distinta dependiendo del navegador. De cualquier modo, me gustaría compartir algunos ejemplos, así podrás saber lo que está por venir. Estos ejemplos actualmente funcionan únicamente en Chrome, y se basan en la última versión estandar.

Estructura Simple usando Flexbox

```css
.container {
  display: -webkit-flex;
  display: flex;
}
nav {
  width: 200px;
}
.flex-column {
  -webkit-flex: 1;
          flex: 1;
}
```

```
<nav> ss= container >
  • Inicio
  • Taco Menú
  • Lista de Pendientes
  • Horas
  • Direcciones
  • Contacto
</nav>
<section> "flex-column" >
  ¡Flexbox es muy fácil!
</section>
<section>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.
</se</div>
```

Estructura fantástica usando Flexbox

```css
.container {
  display: -webkit-flex;
  display: flex;
}
.initial {
  -webkit-flex: initial;
          flex: initial;
  width: 200px;
  min-width: 100px;
}
.none {
  -webkit-flex: none;
          flex: none;
  width: 200px;
}
.flex1 {
  -webkit-flex: 1;
          flex: 1;
}
.flex2 {
  -webkit-flex: 2;
          flex: 2;
}
```

```
<div class="initial">
  Tendré 200px si hay espacio y me reduciré a 100px si no lo hay, pero nunca menos de eso.
</div>
<div class="none">
  Tendré siempre 200px, sin importar lo que pase.
</div>
<div class="flex1">
  Llenaré 1/3 del ancho que quede.
</div>
<div class="flex2">
  Llenaré 2/3 del ancho que quede.
</div>
```

Centrando con Flexbox

```css
.vertical-container {
  height: 300px;
  display: -webkit-flex;
  display:         flex;
  -webkit-align-items: center;
          align-items: center;
  -webkit-justify-content: center;
          justify-content: center;
}
```

```
<div class="vertical-container">
  <div>
    ¡Finalmente es fácil centrar verticalmente con CSS!
  </div>
</div>
```