



DISEÑO DE INTERFACES WEB

Unidad 2: USOS DE ESTILOS. MODELO DE CAJAS

Departamento de Informática
Curso 2020/2021



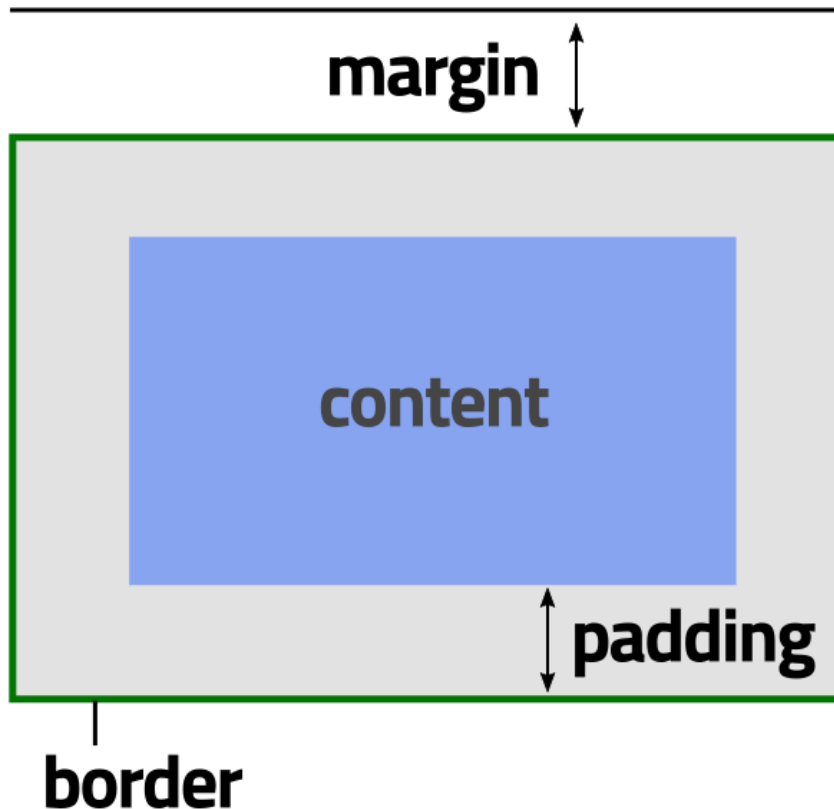


CONTENIDOS DE LA UNIDAD

1. Representación básica del modelo de cajas.
2. Zonas de un elemento.
3. Propiedades de un elemento.
4. Cálculo del tamaño de un elemento.
5. Modelos de posicionamiento de elementos.
6. Modelo de bloques.
7. Modelo flexible. Flexbox.
8. Modelo Rejilla.
9. Media Queries.
10. Diseño adaptativo con Media Queries. Modelo flexible.
11. Diseño adaptativo con Media Queries. Modelo Rejilla.
12. Preprocesadores CSS.



1. REPRESENTACIÓN BÁSICA DEL MODELO DE CAJAS

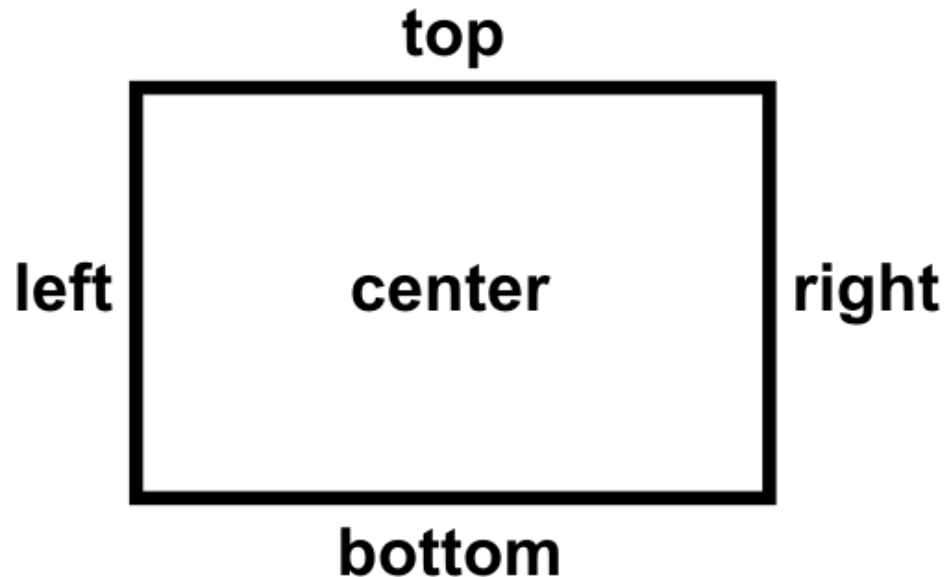


- Los **márgenes** (*margin*) son los espacios que hay entre los bordes del elemento en cuestión y los elementos externos.
- Los **espaciados** (*padding*) son los espacios que hay entre los bordes del elemento en cuestión y el contenido del elemento
- El **borde** (*border*) son las líneas gráficas existentes alrededor de la caja.

Hay que diferenciar bien los **márgenes** de los **espaciados**, puesto que no son la misma cosa. Los **espaciados** (*padding*) son los espacios que hay entre los bordes del elemento en cuestión y el contenido del elemento. Mientras que los márgenes (*margin*) son los espacios que hay entre los bordes del elemento en cuestión y sus elementos adyacentes.



2. ZONAS DE UN ELEMENTO



Top: Se refiere a la parte superior del elemento.

Left: Se refiere a la parte izquierda del elemento.

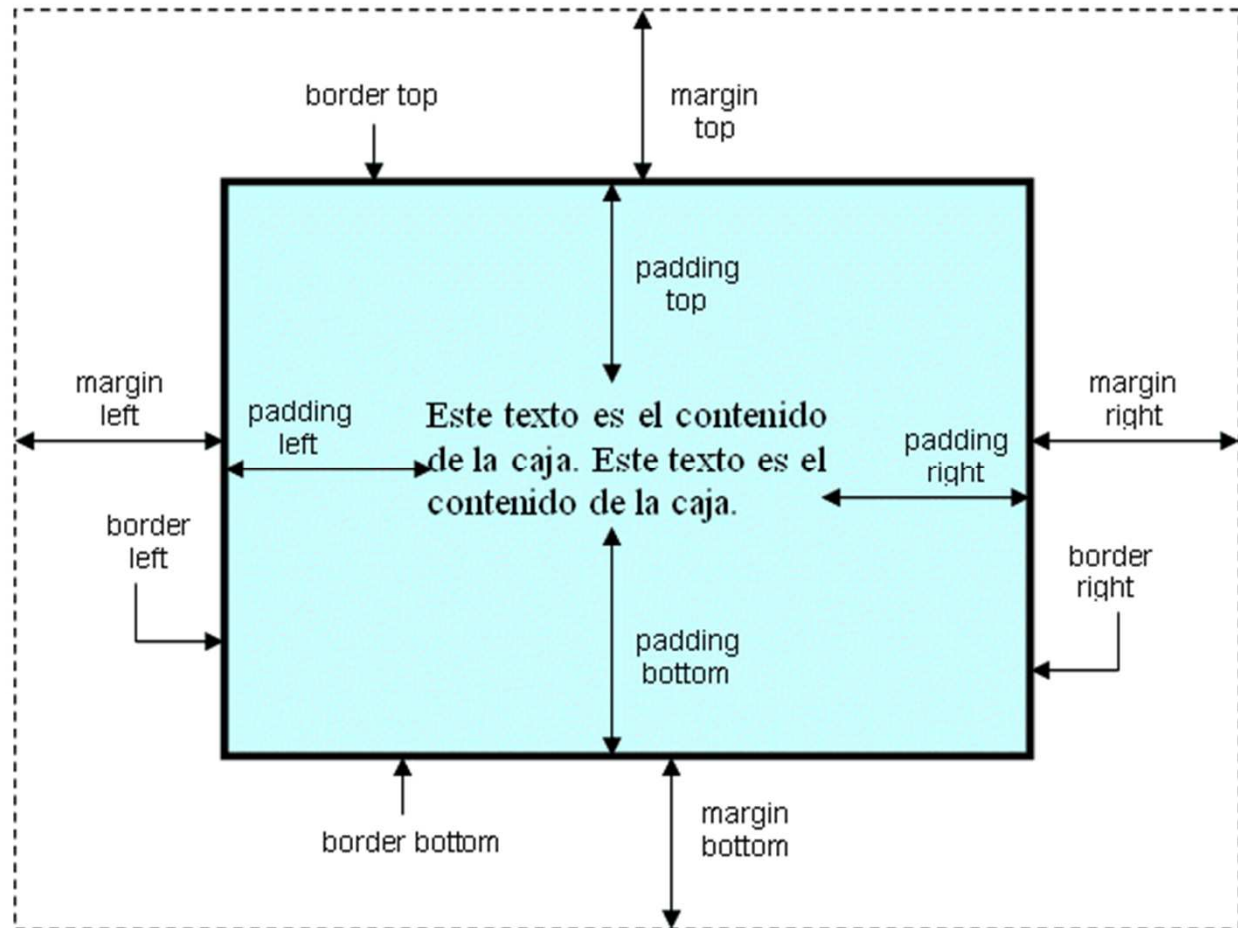
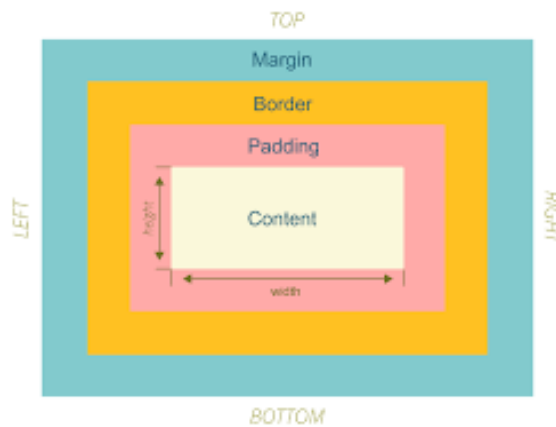
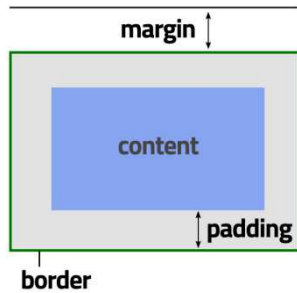
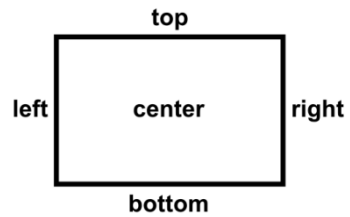
Right: Se refiere a la parte derecha del elemento.

Bottom: Se refiere a la parte inferior del elemento.

Center: En algunos casos se puede especificar el valor center para referirse a la posición central entre dos extremos.

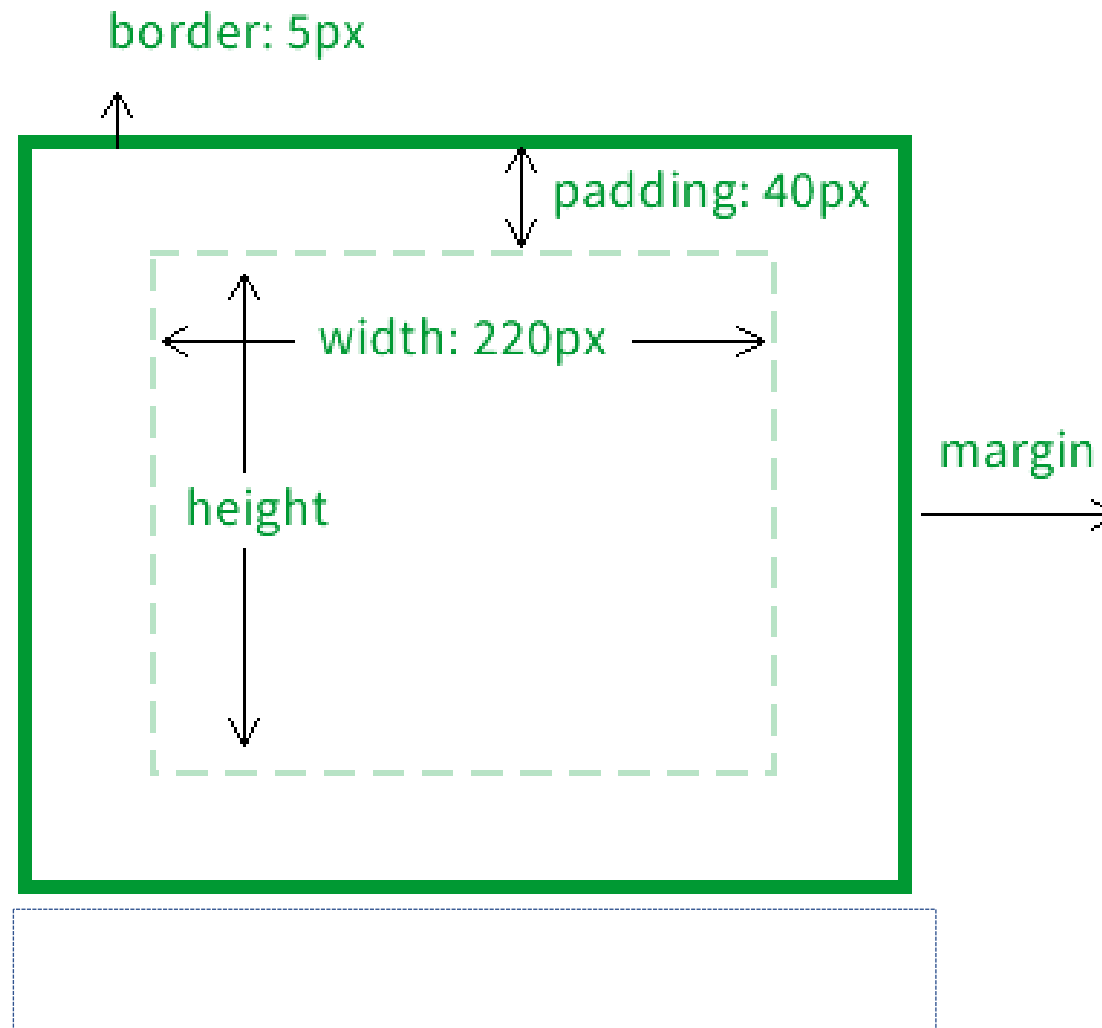


3. PROPIEDADES UN ELEMENTO





4. CÁLCULO DEL TAMAÑO DE UN ELEMENTO





5. MODELO DE CAJAS O BOX MODEL

THE CSS BOX MODEL HIERARCHY

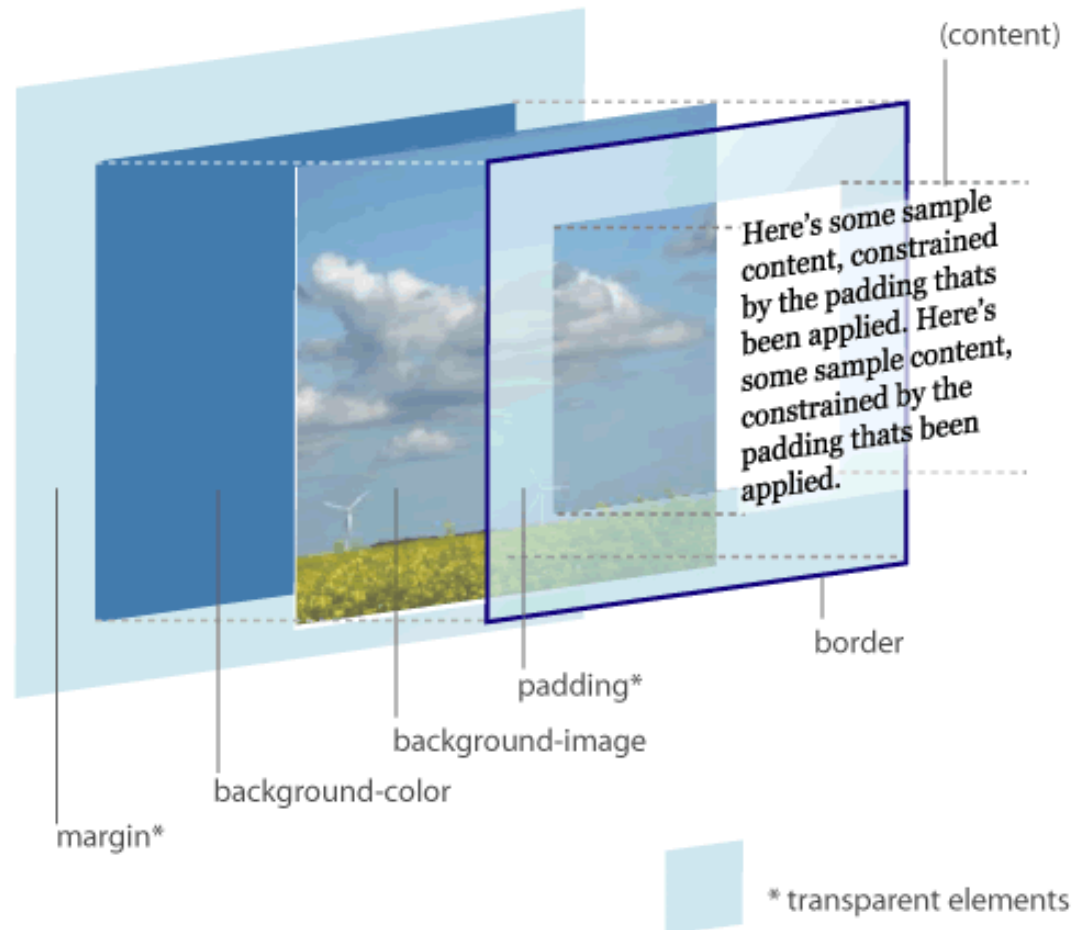


Figura 4.3 Representación tridimensional del box model de CSS



5. MODELOS DE POSICIONAMIENTO DE ELEMENTOS

Modelo de bloques.

- Los elementos se van colocando hasta rellenar el espacio de la página de izquierda a derecha y de arriba abajo.
- El problema de esta opción es que no se adapta fácilmente a los dispositivos móviles, ya que no permite reordenar y reajustar los elementos.

Modelo flexible o *flexbox*.

- Permite acomodar los elementos de una página según cambien las dimensiones y orientación de la página. Por tanto, nos permite hacer un diseño *adaptativo* y que los contenidos se visualicen correctamente en cualquier dispositivo.
- Es muy reciente y no es soportado por todos los navegadores.

Modelo de rejilla.

- Es el modelo que se está imponiendo, y permite crear sitios *adaptativos* gracias a las *media queries*.
- La pantalla se divide en una especie de rejilla virtual y a continuación se indica cuántas celdas de dicha rejilla ocupa cada elemento dependiendo del tamaño y la orientación.

Bootstrap.



6. MODELO DE BLOQUES

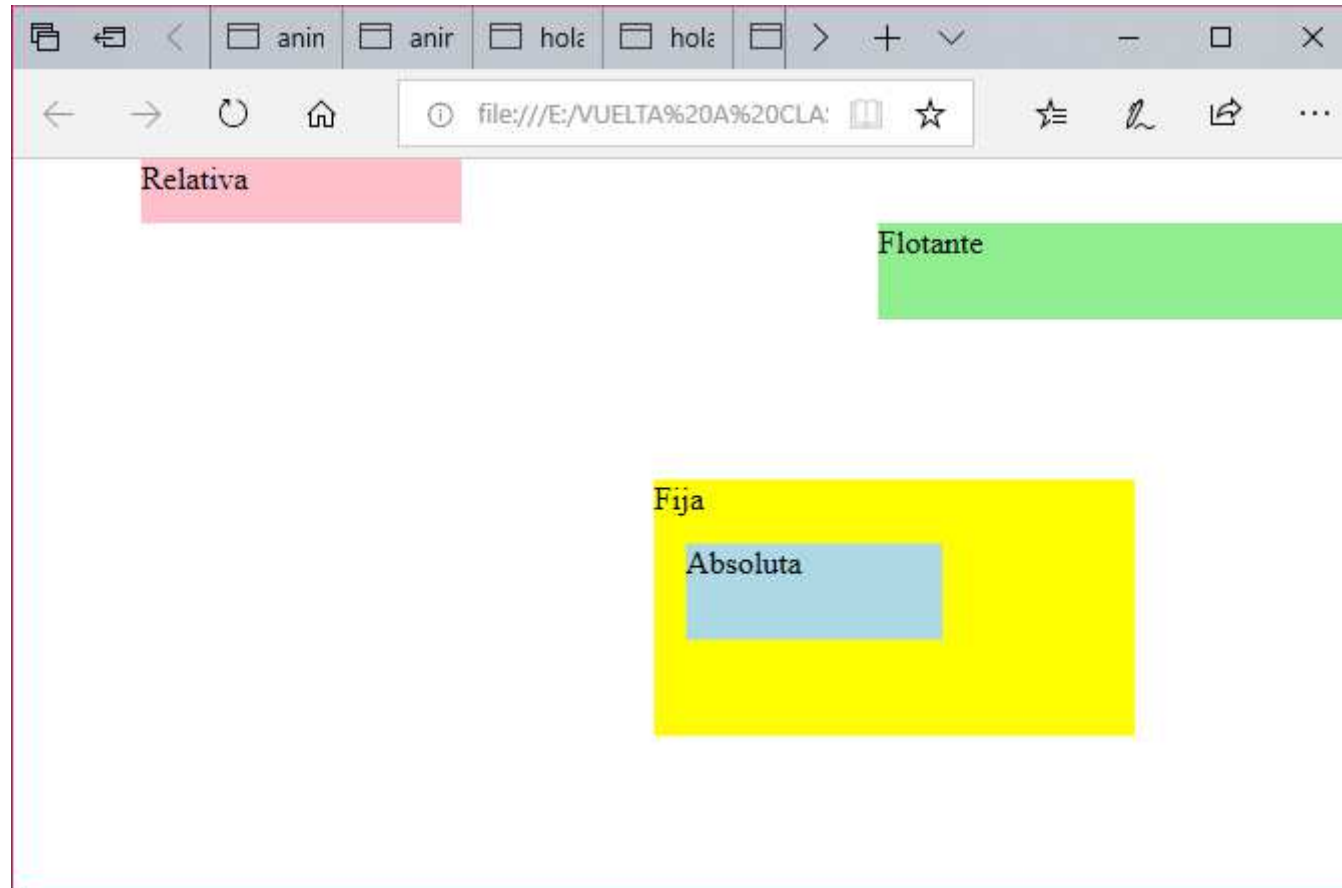
Los elementos se van colocando hasta rellenar el espacio de la página de izquierda a derecha y de arriba abajo.

Podemos cambiar el flujo normal, haciendo que las cajas se posicionen según el valor de la propiedad `position`:

- **Estática** (`position: static`): es el modo de posicionamiento por defecto. Cada caja ocupará la posición en la que quedaría en un “flujo normal”.
- **Relativa** (`position: relative`): se desplaza de la posición estática usando las propiedades `top`, `right`, `bottom` y `left`. El resto de las cajas no se ven afectadas.
- **Absoluta** (`position: absolute`): elimina y flujo normal de la caja y la posiciona en una posición fija de manera absoluta con respecto a la caja contenedora. Se usan de nuevo las propiedades `top`, `right`, `bottom` y `left`.
- **Fija** (`position: fixed`): igual que el posicionamiento absoluto, pero especificando la posición respecto a la ventana.
- **Flotante** (`float:`): la caja se desplaza hacia un lado (`float: left` o `float: right`) llevándola a uno de los extremos de la caja contenedora o hasta la primera caja flotante que se encuentre en esa dirección. Esto implica que se puede superponer a otras cajas no flotantes.



6. MODELO DE BLOQUES. EJEMPLO 1



Código HTML: [cap03_ejemplo02.HTML](#)

Código CSS: [cap03_ejemplo02.CSS](#)



6. MODELO DE BLOQUES. EJEMPLO 1

```
<html>
  <head>
    <title>Posicionamiento</title>
    <meta charset="utf-8"/>
    <link rel="stylesheet" type="text/css" href="cap03_ejemplo02.css" />
  </head>
  <body>
    <div id="relativa">Relativa</div>
    <div id="fija">
      Fija
      <div id="absoluta">Absoluta</div>
    </div>
    <div id="flotante">Flotante</div>
  </body>
</html>
```



6. MODELO DE BLOQUES. EJEMPLO 1

```
body{
  margin: 0;
}

div#relativa {
  background: ■pink;
  position: relative;
  width: 10em;
  height: 2em;
  left: 4em;
}

div#fija {
  background: ■yellow;
  position: fixed;
  width: 15em;
  height: 8em;
  left: 20em;
  top: 10em;
}
```

```
div#absoluta {
  background: ■lightblue;
  position: absolute;
  width: 8em;
  height: 3em;
  left: 1em;
  top: 2em;
}

div#flotante {
  background: ■lightgreen;
  position: float;
  width: 15em;
  height: 3em;
  float: right;
}
```



6. MODELO DE BLOQUES. EJEMPLO 2

Copiar los siguientes códigos:

Código HTML: [cap03_ejerciciobloques.HTML](#)

Código CSS: [cap03_ejerciciobloques.CSS](#)

```
<html>
  <head>
    <meta charset="utf-8"/>
    <link rel="stylesheet" type="text/css" href="cap03_ejerciciobloques.css" />
  </head>
  <body>
    <div id="a"></div>
    <div id="b">
      <div id="c"></div>
    </div>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Maiores est voluptas dolorem quod id neque esse doloribus saepe ut reprehenderit possimus fugit. Laudantium distinctio doloribus repellendus quibusdam voluptatem recusandae aut.</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Doloremque nam illum eveniet iste commodi culpa vel quibusdam sint veniam itaque provident sed ipsum reiciendis labore beatae impedit vitae earum aut.</p>
  </body>
</html>
```



6. MODELO DE BLOQUES. EJEMPLO 2

```
div#a {  
    background: blue;  
    width: 2em;  
    height: 4em;  
-}  
  
div#b {  
    background: red;  
    width: 4em;  
    height: 2em;  
-}  
  
div#c {  
    background: pink;  
    width: 1em;  
    height: 1em;  
-}  
  
p {  
    font-size: 80px;  
    background: yellow;  
    margin: 0;  
    padding 0;  
-}
```



6. MODELO DE BLOQUES. EJEMPLO 2

El tamaño de la fuente es intencionadamente grande para que necesitemos hacer *scroll* en la página. Ahora prueba a realizar las siguientes acciones para ver el resultado:

1. Cambia el posicionamiento del bloque *a* a *relative* y con un espaciado superior de 1em e izquierdo de 2em. Verás que el bloque azul se superpone al rojo, pero se reserva el espacio que debería haber ocupado el bloque azul.
2. Cambia el posicionamiento del bloque *c* a *relative* y con un espaciado superior de 1em e izquierdo de 2em. Verás que, en este caso, el bloque rosa cambia su posición *con respecto a su contenedor* (es decir, el bloque rojo).
3. Cambia el posicionamiento del bloque *a* a *absolute* y con un espaciado superior de 1em e izquierdo de 2em. Verás que el bloque azul se superpone al rojo pero que este último *no* ha reservado el espacio que debería haber ocupado el bloque azul, por lo que el rojo se coloca en la esquina superior izquierda de la pantalla. Verás que, si haces *scroll*, los bloques dejan de verse.
4. Cambia el posicionamiento del bloque *a* a *fixed* y con un espaciado superior de 1em e izquierdo de 2em. Como antes, el bloque rojo no ha reservado el espacio del bloque azul. En cambio, si hacemos *scroll*



6. MODELO DE BLOQUES. EJEMPLO 2

veremos que el bloque azul sigue en una posición fija con respecto a la ventana.

5. Cambia el posicionamiento del bloque *a* a `float` y añade la propiedad `float:right`. Verás que el bloque azul se ha colocado junto al borde derecho de la ventana y que no se le ha reservado espacio (por lo que se superpone al resto de los elementos).
6. Cambia el posicionamiento del bloque *c* a `float` y añade la propiedad `float:right`. Verás que el bloque rosa se ha pegado al lado derecho de su contenedor, el bloque rojo.
7. Cambia el posicionamiento del bloque *b* a `float` y añade la propiedad `float:right`. Verás que el bloque rojo se ha pegado al lado derecho hasta que ha chocado con el elemento flotante anterior, el bloque azul.
8. Cambia el posicionamiento del bloque *a* a `float` y añade la propiedad `float:left`. Añade a *p* la propiedad `clear:right`. Verás que el párrafo se coloca debajo de la caja roja, pero se superpone a la caja azul.
9. Cambia en *p* la propiedad `clear:both`. Verás que ahora el párrafo se ha colocado justo debajo de la caja azul.



7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*.

Permite acomodar los elementos de una página según cambien las dimensiones y orientación de la página. Por tanto, nos permite hacer un diseño *adaptativo* y que los contenidos se visualicen correctamente en cualquier dispositivo.

Es muy reciente y no es soportado por todos los navegadores.

La idea principal es que un contenedor flexible expande o comprime sus elementos para rellenar el espacio libre o ajustarse al área disponible. De esta forma tendremos un *contenedor flex (flex container)* y una serie de *elementos flex contenidos (flex item)*.

Para definir un contenedor utilizaremos la propiedad

```
display: flex
```

```
display: -webkit-flex (para compatibilidad con safari)
```

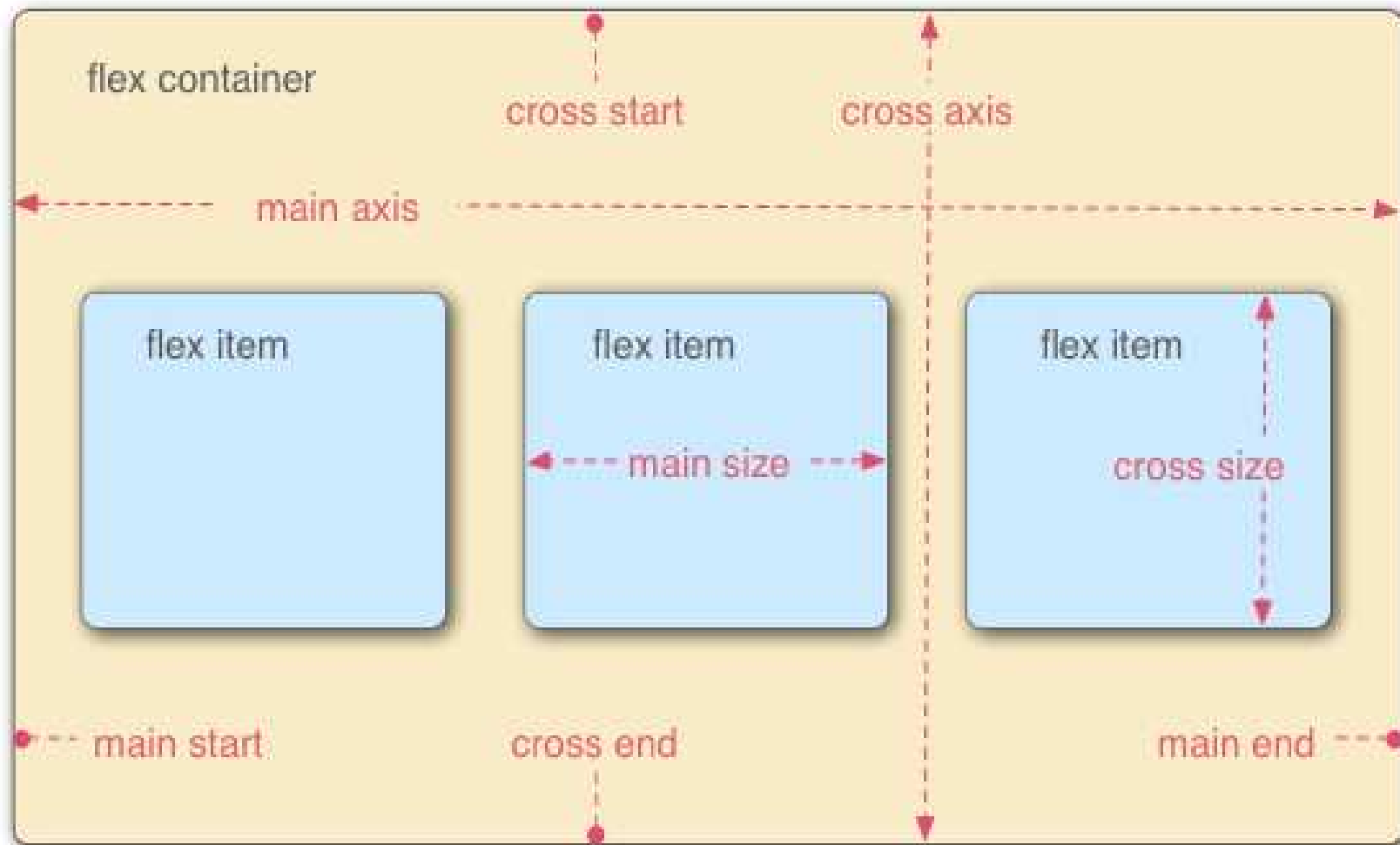
Todos los elementos contenidos pasarán a tener la condición de elementos flex.

En este modelo no se utilizan conceptos como ordenamiento horizontal o vertical, sino que se define un eje principal (*main axis*) y otro secundario (*cross axis*). De esta forma, el diseño se ajusta fácilmente a los cambios de orientación del dispositivo.

Por cada eje podremos situar componentes en su inicio (`main-start` o `cross-start`) y final (`main-end` o `cross-end`)



7. MODELO FLEXIBLE o *FlexBox*.





7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*.

```
<html>
  <head>
    <title>Flexbox</title>
    <meta charset="utf-8"/>
    <link rel="stylesheet" type="text/css" href="cap03_ejemplo03.css" />
  </head>
  <body>

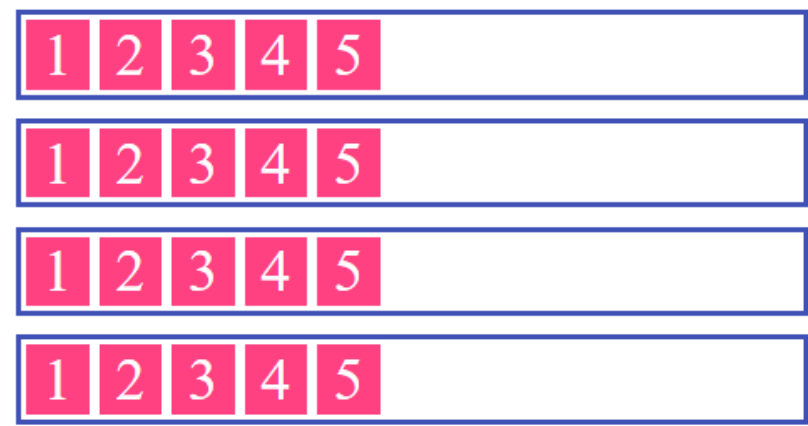
    <div id="cont01" class="contenedor">
      <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
    </div>
    <div id="cont02" class="contenedor">
      <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
    </div>
    <div id="cont03" class="contenedor">
      <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
    </div>
    <div id="cont04" class="contenedor">
      <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
    </div>

  </body>
</html>
```



7. MODELO FLEXIBLE o *FlexBox*.

```
.contenedor{  
  width: 100%;  
  border-style: solid;  
  border-color: #3F51B5;  
  border-width: 4px;  
  margin-top: 15px;  
  display: flex;  
}  
  
.contenedor>div{  
  background-color: #FF4081;  
  color: white;  
  text-align: center;  
  width: 50px;  
  font-size: 3em;  
  margin: 4px;  
}
```





7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*. DIRECCIÓN DE LOS ELEMENTOS

La propiedad `flex-direction` especifica la dirección del eje principal y, por tanto, cómo se colocan los elementos dentro del contenedor.

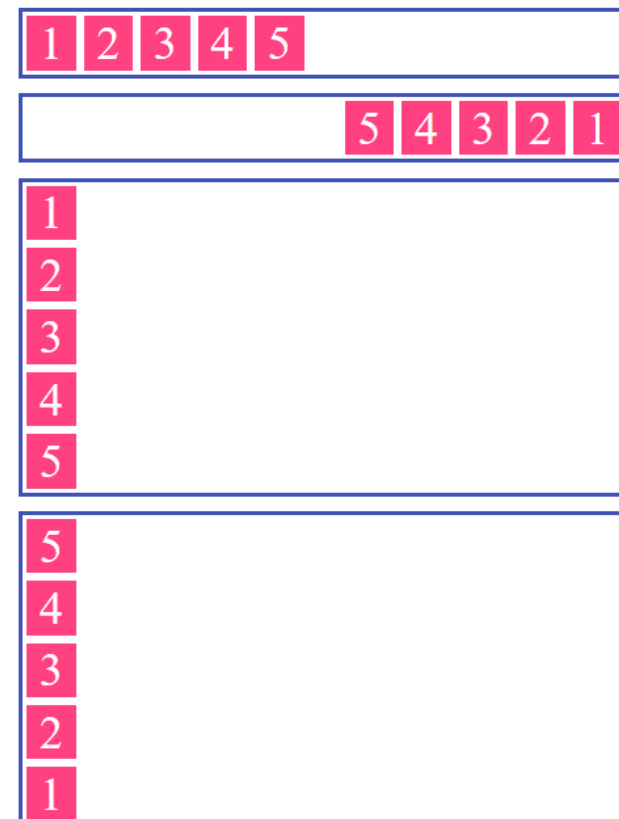
Podemos utilizar los valores: `row` (fila), `row-reverse` (fila invertida), `column` (columna) o `column-reverse` (columna invertida)

Añadid el siguiente código en la hoja CSS

```

]#cont01{
    flex-direction: row;
-}
]#cont02{
    flex-direction: row-reverse;
-}
]#cont03{
    flex-direction: column;
-}
]#cont04{
    flex-direction: column-reverse;
-}

```





7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*. AJUSTE EN EL EJE PRINCIPAL

Esta propiedad `flex-wrap` establece si es necesario ajustar los elementos para que quepan en una sola fila (o columna).

Los valores posibles son:

`nowrap`, que ajusta los elementos a una línea.

`wrap`. Que los distribuye sin cambiar su tamaño

`wrap-reverse`, que es similar a la anterior, pero en orden inverso.

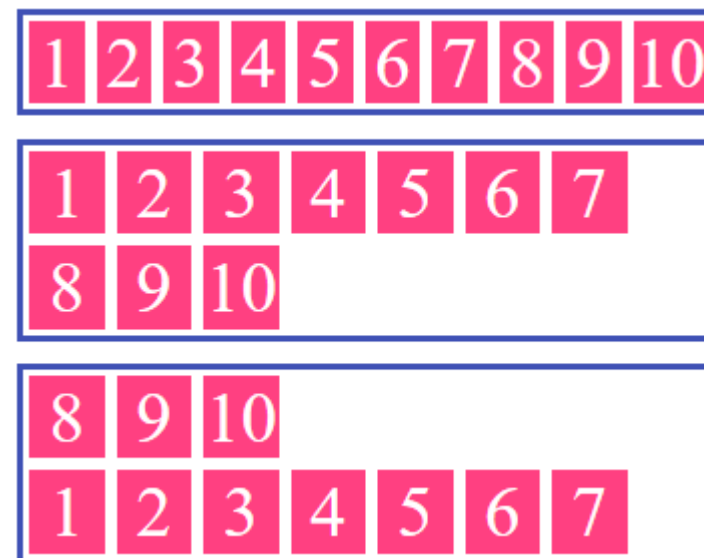
Para comprobarlo Añadid el siguiente código en la hoja CSS y html



7. MODELO FLEXIBLE o *FlexBox*.

```
<div id="cont05" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
  <div>6</div><div>7</div><div>8</div><div>9</div><div>10</div>
</div>
<div id="cont06" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
  <div>6</div><div>7</div><div>8</div><div>9</div><div>10</div>
</div>
<div id="cont07" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
  <div>6</div><div>7</div><div>8</div><div>9</div><div>10</div>
</div>
```

```
#cont05{
  flex-wrap: nowrap;
}
#cont06{
  flex-wrap: wrap;
}
#cont07{
  flex-wrap: wrap-reverse;
}
```





7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*. **ALINEACIÓN DEL CONTENIDO**

Con la propiedad `justify-content` podemos alinear los elementos respecto al eje principal del contenedor. De esta forma decidimos qué hacer con el espacio restante.

Los posibles valores son:

- `flex-start`, que junta los elementos al principio del eje.

- `flex-end`, que los junta al final.

- `center`, para centrarlos.

- `space-between`, que reparte el espacio entre los elementos dejando los extremos pegados al borde.

- `space-around`, donde todo el espacio restante se reparte alrededor de cada elemento.

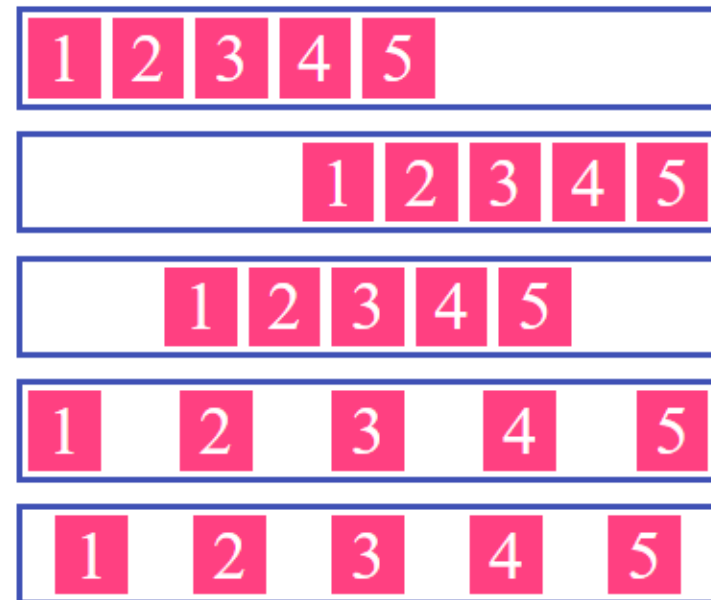
Para comprobarlo Añadid el siguiente código en la hoja CSS y html



7. MODELO FLEXIBLE o *FlexBox*.

```
<div id="cont08" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont09" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont10" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont11" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont12" class="contenedor">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
```

```
##cont08{
  justify-content: flex-start;
}
##cont09{
  justify-content: flex-end;
}
##cont10{
  justify-content: center;
}
##cont11{
  justify-content: space-between;
}
##cont12{
  justify-content: space-around;
}
```





7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*. AJUSTE EN EL EJE SECUNDARIO

Para ajustar los elementos respecto al eje secundario contamos con la propiedad `align-items`.

En este caso los posibles valores son:

`stretch`, para ocupar todo el alto

`flex-start`, que alinea los elementos al principio del eje.

`flex-end`, que los alinea al final.

`center`, para centrarlos.

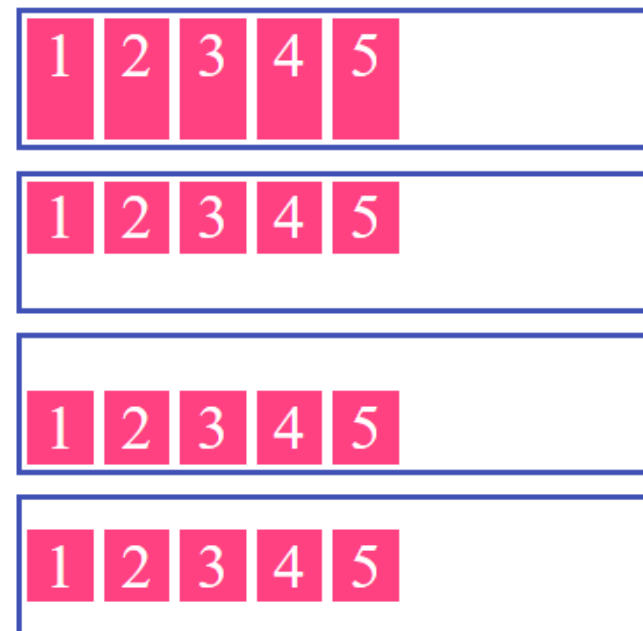
Para comprobarlo Añadid el siguiente código en la hoja CSS y html



7. MODELO FLEXIBLE o *FlexBox*.

```
<div id="cont13" class="contenedor contenedorAlto">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont14" class="contenedor contenedorAlto">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont15" class="contenedor contenedorAlto">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
<div id="cont16" class="contenedor contenedorAlto">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div>
</div>
```

```
.contenedorAlto{
  height: 100px;
}
#cont13 {
  align-items: stretch;
}
#cont14 {
  align-items: flex-start;
}
#cont15 {
  align-items: flex-end;
}
#cont16 {
  align-items: center;
}
```





7. MODELO FLEXIBLE o *FlexBox*.

Modelo flexible o *flexbox*. PROPIEDADES DE LOS ELEMENTOS

Hasta ahora hemos visto propiedades para configurar el contenedor principal. Pero también hay propiedades interesantes para los elementos que nos van a permitir ordenarlos e indicar como van a ocupar el espacio.

Las propiedades son:

`order`, indica el orden en que aparece el elemento dentro del contenedor.

`flex-grow`, establece el *factor de crecimiento* del elemento, es decir, como crece el elemento en relación a otros.

`flex-shrink`, análoga a la propiedad anterior, en este caso define el *factor de encogimiento*.

`flex-basis`, indica el tamaño por defecto (ancho) del elemento antes de que el espacio libre sea distribuido. Normalmente se deja el valor por defecto `flex-basis: auto`

Para comprobarlo Añadid el siguiente código en la hoja CSS y html



7. MODELO FLEXIBLE o *FlexBox*.

```
<div class="contenedor">
  <div id="item1">1</div>
  <div id="item2">2</div>
  <div id="item3">3</div>
  <div id="item4">4</div>
  <div id="item5">5</div>
</div>
```

```
#item1{
  order: 3;
}
#item2{
  order: 1;
  flex-grow: 2;
}
#item3{
  order: 2;
}
#item4{
  order: 5;
}
#item5{
  order: 4;
}
```





7. MODELO FLEXIBLE o *FlexBox*.


ENLACES DE INTERÉS

<https://www.emenia.es/flexbox-la-caja-flexible-css3/>

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Usando_las_cajas_flexibles_CSS

Browser Support

The flexbox properties are supported in all modern browsers.

				
29.0	11.0	22.0	10	48



8. MODELO REJILLA

Modelo de rejilla.

- Es el modelo que se está imponiendo, y permite crear sitios *adaptativos* gracias a las *media queries*.
- La pantalla se divide en una especie de rejilla virtual y a continuación se indica cuántas celdas de dicha rejilla ocupa cada elemento dependiendo del tamaño y la orientación.
- Podemos decir que la librería que ha popularizado este tipo de modelo es **Bootstrap** (<http://getbootstrap.com>), creada por los desarrolladores de Twitter.

ACTIVIDAD1: Descargad la librería Bootstrap de la dirección <http://getbootstrap.com>

ACTIVIDAD2: Comentar la hoja de estilo [bootstrap.css](#)



8. MODELO REJILLA

FUNCIONAMIENTO.

- El espacio disponible se divide en 12 columnas de igual tamaño.
- Por cada elemento indicaremos cuántas columnas ocupa dependiendo del tamaño del dispositivo.

Ejemplo: Para una barra lateral podemos indicar que ocupe dos columnas en tamaño escritorio, seis columnas (la mitad del ancho total) para *tablets* y 12 columnas (todo el ancho) para dispositivos móviles.

Para esto se utilizan una serie de clases que en este caso serían: `.col-md-2`, `.col-sm-6` y `.col-xs-12` respectivamente.

Como puede verse, en estas clases se indica el número de columnas deseado cuando el navegador tenga un tamaño concreto. Estos tamaños se definen así en **Bootstrap**:

xs (*extra small*): Pantallas con ancho inferior a 768px. La gran mayoría de los smartphones.

sm (*small*): Pantallas con ancho inferior a 992px. Si se ha definido la clase `xs` y el tamaño es inferior a 768px, entonces se aplicará esa clase.

md (*medium*): Pantallas entre 992px y 1200px. La mayoría de los navegadores de escritorio.

Lg (*large*): Pantallas muy grandes con ancho superior a 1200px. Pantallas grandes de alta definición.

https://ajgallego.gitbooks.io/bootstrap-3/content/capitulo_rejilla.html



8. MODELO REJILLA

```
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```

En la siguiente imagen se puede ver el resultado para una pantalla mediana (de escritorio):

.col-md-8		.col-md-4
.col-md-4	.col-md-4	.col-md-4
.col-md-6		.col-md-6



8. MODELO REJILLA

```
<!-- En pantallas pequeñas aparecerá una columna que ocupará todo el ancho  
y otra que ocupará la mitad de la pantalla -->
```

```
<div class="row">  
  <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>  
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>  
</div>
```

```
<!-- En pantallas medianas se indica que ocupe cada columna la mitad  
del ancho disponible -->
```

```
<div class="row">  
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>  
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>  
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>  
</div>
```

```
<!-- Como no se indica el tamaño para pantallas grandes las columnas  
siempre ocuparán el 50% -->
```

```
<div class="row">  
  <div class="col-xs-6">.col-xs-6</div>  
  <div class="col-xs-6">.col-xs-6</div>  
</div>
```

En la siguiente imagen se puede ver como quedaría el código de ejemplo para pantallas medianas (md) y grandes (lg):

.col-xs-12 .col-md-8		.col-xs-6 .col-md-4	
.col-xs-6 .col-md-4	.col-xs-6 .col-md-4	.col-xs-6 .col-md-4	
.col-xs-6		.col-xs-6	

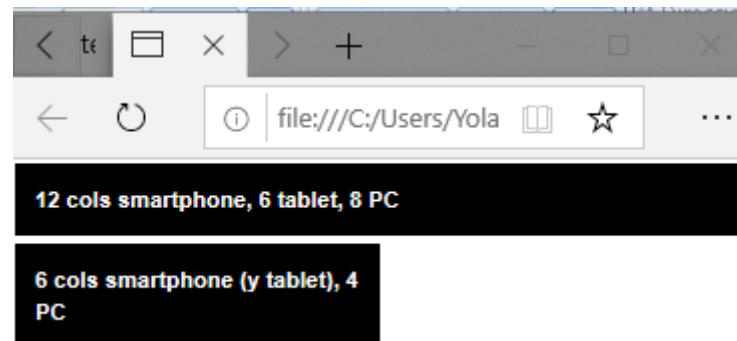
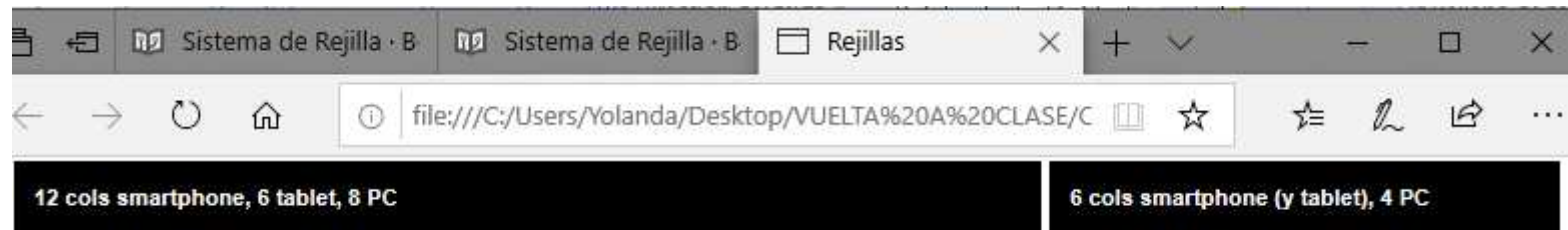


8. MODELO REJILLA

```
<html>
  <head>
    <title>Rejillas</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="bootstrap.css">
    <style>
      .row>div {
        background-color: black;
        color:white;
        font-weight: bolder;
        padding: 12px;
        border-style: solid;
        border-color: white;
        border-width: 3px
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-xs-12 col-sm-6 col-md-8">
          12 cols smartphone, 6 tablet, 8 PC
        </div>
        <div class="col-xs-6 col-md-4">
          6 cols smartphone (y tablet), 4 PC
        </div>
      </div>
    </div>
  </body>
</html>
```



8. MODELO REJILLA





9. MEDIA QUERIES

Las Media Queries son una de las grandes ventajas de CSS3, ya que permiten saber qué sistema se está visualizando en una página web y, en función de ello, aplicar unas reglas de estilo u otras. Así, podemos servir un CSS personalizado, acorde las condiciones del navegador o dispositivo que nos visita. Se han convertido en **uno de los mejores recursos con los que cuentan los diseñadores para hacer sitios responsive**.

El **Html4 y CSS2** ya se soportaban hojas de estilo a medida para los siguientes tipos de medios (media types):

- HTML4: *mediatype*
'print', 'screen', 'aural', 'braille', 'handheld', 'print', 'projection', 'tty', 'tv'.
- CSS2: define la misma lista menos 'aural' y añade 'embossed' y 'speech'.
También, 'all' es usado para aplicar la hoja de estilo a todas las media types.

Ejemplo:

```
<link rel="stylesheet" type="text/css" media="screen" href="pantalla.css">
```

```
<link rel = "stylesheet" type = "text / css" media = "print" href = "impresora.css">
```



9. USO DE @ MEDIA QUERIES

En desarrollo web, las **media queries** son un módulo CSS3 que permite adaptar la representación del contenido a características del dispositivo como la resolución de pantalla (por ejemplo, un *smartphone* frente a pantallas de alta definición) o la presencia de características de accesibilidad como el braille. Se convirtió en un estándar recomendado por la W3C en junio de 2012 y es un principio básico de la tecnología de Diseño web adaptativo.

Desde la especificación de CSS 2.1, ha sido posible modificar el aspecto de los documentos HTML en función del [tipo de dispositivo](#) en el que se mostraban. El caso más común es el de crear una hoja de estilos que se aplica al imprimir los documentos:

INCLUIRLO EN EL DOCUMENTO HTML mediatype

```
<link rel="stylesheet" type="text/css" href="core.css" media="screen" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px)" href="shetland.css" />
```

Media type admitidos HTML5 https://www.w3schools.com/tags/att_link_media.asp

INCLUIRLO EN LA HOJA CSS

```
@media screen and (min-width: 1024px) { body { font-size: 100%; } }
```

O incluso utilizando la sentencia @import:

```
@import url("wide.css") screen and (min-width: 1024px);
```



9. REGLAS @ MEDIA

Las **reglas @media** nos permiten indicar sobre qué tipo de medio deben aplicarse ciertas reglas CSS. La sintaxis general es:

```
@media [media type and] (media feature) {  
    CSS-Code  
}
```

Las **media feature** nos van a permitir indicar las características del dispositivo y, podemos utilizar los operadores lógicos **and**, **not** y **only**.

Ejemplos:

```
@media (min-width: 700px) { ..... }  
@media (min-width: 700px) and (orientation: landscape) { ..... }  
@media screen and (min-width: 700px) and (orientation: landscape) {  
    ..... }
```

https://www.w3schools.com/css/css_rwd_mediaqueries.asp



9. REGLAS @ MEDIA *media feature*

Existen diferentes *media features* que nos van a permitir, principalmente, especificar el tamaño del dispositivo, su orientación o resolución, así como el número de colores.

Las más destacadas son:

- **aspect-ratio, min-aspect-ratio, max-aspect-ratio**
Ancho/alto (exacto, mínimo o máximo) del dispositivo
- **color-index**: número de colores que el dispositivo puede mostrar.
- **width, min-width, max-width**
Ancho (exacto, mínimo o máximo) disponible
- **height, min-height, max-height**
Alto (exacto, mínimo o máximo) disponible
- **orientation**: Orientación *landscape* o *portrait*



9. MEDIA QUERIES

ENLACES DE INTERÉS

<https://www.adictosaltrabajo.com/2012/10/11/css3-media-queries/>

https://www.w3schools.com/css/css_rwd_mediaqueries.asp

<https://www.arsys.es/blog/programacion/disenio-web/media-queries-css3/>

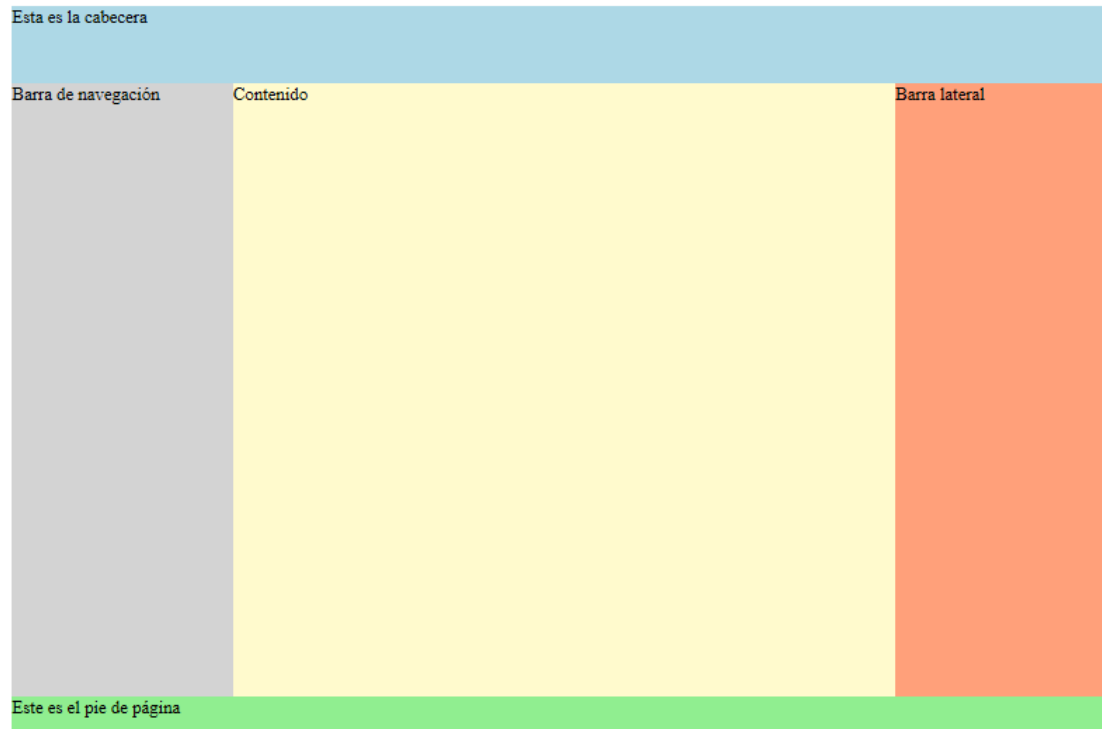


10. DISEÑO ADAPTATIVO CON MEDIA QUERIES

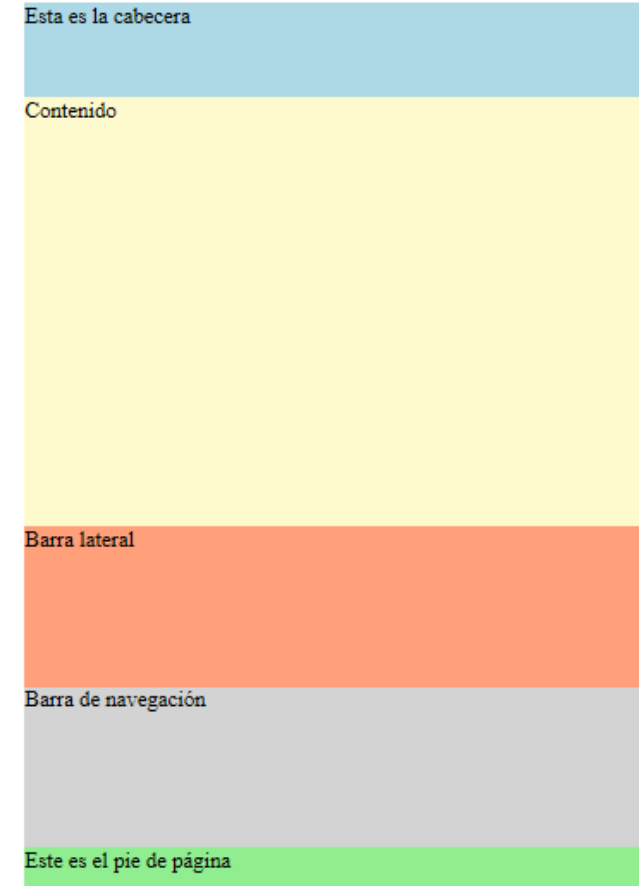
Modelo Flexible

Crear un diseño adaptativo con el modelo flexible es muy fácil. Sobre todo deberemos de **aprovechar la posibilidad de reordenar los elementos que nos ofrece esta opción.**

Ancho mayor que 640 px



Ancho menor que 640 px





10. DISEÑO ADAPTATIVO CON MEDIA QUERIES

Modelo Flexible

```
<html>
  <head>
    <title>Responsive</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" type="text/css" href="cap03_ejemplo04.css" />
  </head>
  <body>
    <header>Esta es la cabecera</header>
    <div id="principal">
      <article>Contenido</article>
      <aside>Barra lateral</aside>
      <nav>Barra de navegación</nav>
    </div>
    <footer>Este es el pie de página</footer>
  </body>
</html>
```



10. DISEÑO ADAPTATIVO CON MEDIA QUERIES

Modelo Flexible

```
header {  
    height: 10%;  
    background-color: lightblue;  
}  
#principal {  
    display: flex;  
    flex-flow: row;  
    align-items: stretch;  
    min-height: 80%;  
}  
nav {  
    background-color: lightgray;  
    flex: 1 6 20%;  
    order: 1;  
}  
article {  
    background-color: lemonchiffon;  
    flex: 3 1 60%;  
    order: 2;  
}  
aside {  
    background-color: lightsalmon;  
    flex: 1 6 20%;  
    order: 3;  
}  
footer {  
    background-color: lightgreen;  
    height: 5%;  
}
```

```
@media all and (max-width: 640px) {  
  
    #principal {  
        flex-flow: column;  
    }  
  
    article {  
        order: 1;  
    }  
    aside {  
        order: 2;  
    }  
    nav {  
        order: 3;  
    }  
}
```

flex: flex-grow flex-shrink flex-basis | auto | initial | inherit;

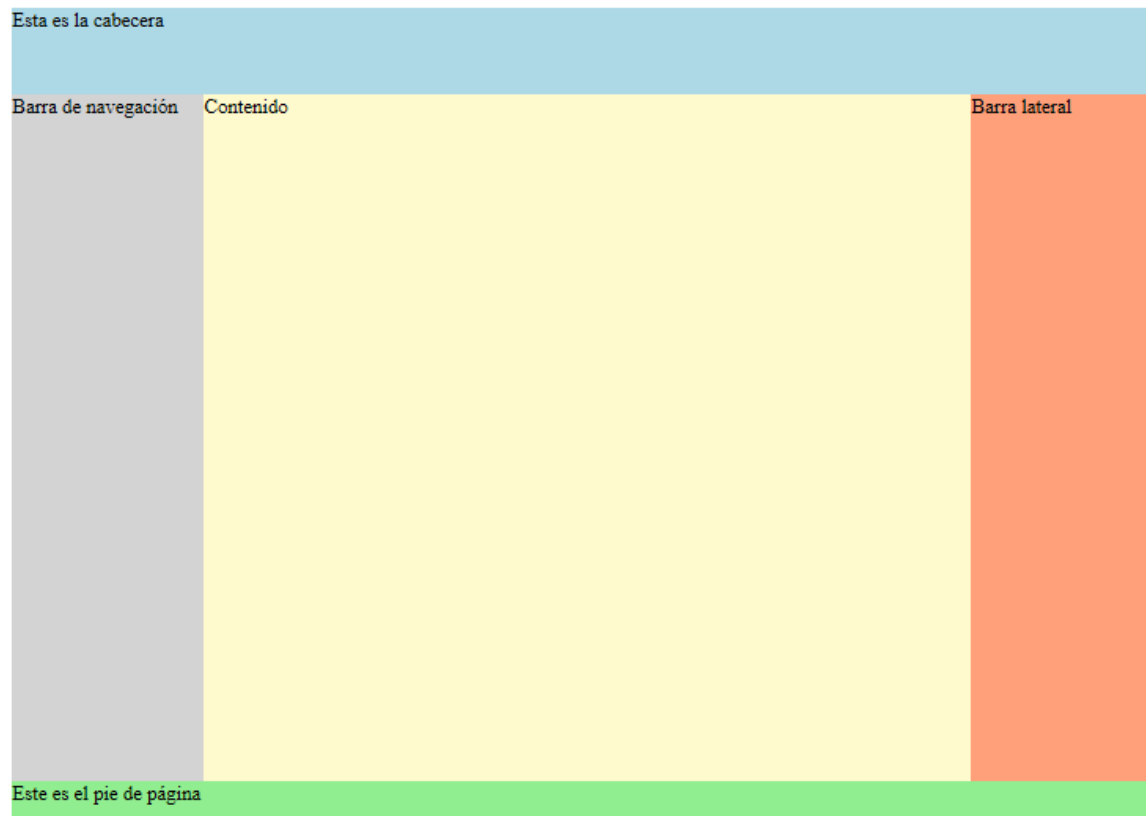


11. DISEÑO ADAPTATIVO CON MEDIA QUERIES

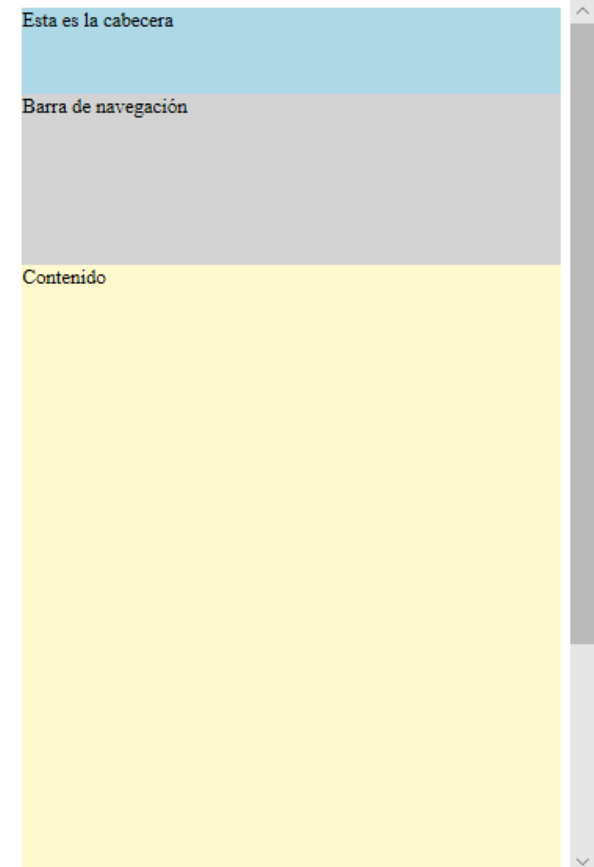
Modelo Rejilla

Veamos como podemos hacer lo mismo con el modelo rejilla. La principal limitación es que no podemos reordenar los elementos. Utilizaremos una hoja CSS tipo *bootstrap*

Ancho mayor que 768 px



Ancho menor que 768px





11. DISEÑO ADAPTATIVO CON MEDIA QUERIES

Modelo Rejilla

```
<html>
  <head>
    <title>Responsive</title>
    <meta charset="utf-8"/>
    <link rel="stylesheet" type="text/css" href="cap03_ejemplo05.css" />
  </head>
  <body>
    <div class="row">
      <header class="col-xs-12 col-sm-12">Esta es la cabecera</header>
    </div>
    <div class="row" id="principal">
      <nav class="col-xs-12 col-sm-2">Barra de navegación</nav>
      <article class="col-xs-12 col-sm-8">Contenido</article>
      <aside class="col-xs-12 col-sm-2">Barra lateral</aside>
    </div>
    <div class="row">
      <footer class="col-xs-12 col-sm-12">Este es el pie de página</footer>
    </div>
  </body>
</html>
```

Utilizamos la clase **col-xs-12** para indicar que el tamaño pequeño emplee las 12 columnas. Las clases **col-sm-12**, **col-sm-8**, **col-sm-2** para que ocupen 12, 8 y 2 columnas respectivamente. La CSS empleada está copiada de *Bootstrap* pero eliminando las clases que no son necesarias.



11. DISEÑO ADAPTATIVO CON MEDIA QUERIES

Modelo Rejilla

```
header {  
    height: 10%;  
    background-color: lightblue;  
}  
#principal {  
    min-height: 80%;  
}  
nav {  
    background-color: lightgray;  
    height: 20%;  
}  
article {  
    background-color: lemonchiffon;  
    min-height: 80%;  
}  
aside {  
    background-color: lightsalmon;  
    height: 20%;  
}  
footer {  
    background-color: lightgreen;  
    height: 5%;  
}  
.row:after {  
    clear: both;  
}  
.col-xs-12 {  
    float: left;  
    width: 100%;  
}
```

```
@media (min-width: 768px) {  
    .col-sm-2, .col-sm-8, .col-sm-12 {  
        float: left;  
    }  
    .col-sm-12 {  
        width: 100%;  
    }  
    .col-sm-8 {  
        width: 66.66666667%;  
    }  
    .col-sm-2 {  
        width: 16.66666667%;  
    }  
    nav, aside {  
        height: 80%;  
    }  
}
```



12. PREPROCESADORES CSS

¿ QUÉ SON ?

Los preprocesadores son lenguajes para hacer hojas de estilos para sitios web, similares a CSS, pero que poseen ciertas características que este no tiene y cuando terminas se compilan a CSS puro.

¿ CUÁNTOS EXISTEN ?

Existen principalmente 3 pre-procesadores de CSS (hay más, pero no son tan conocidos), estos son **LESS**, **SASS** y **Stylus**. Cada uno tiene sus propias características que los hacen diferente de los demás y a su vez comparten varias muy comunes.

VENTAJAS

- ✓ Código más organizado
- ✓ Proyectos más fáciles de mantener
- ✓ Reutilización de código



12. PREPROCESADORES CSS - LESS

INSTALANDO LESS

Existen varias formas de instalar LESS:

<https://programandoointentandolo.com/2017/07/tutorial-less-1-introduccion-less.html>

Instalando [los plugins LESS en Sublime Text](#)

VARIABLES

En LESS las variables se definen mediante el símbolo @

Archivo .LESS

```
@colorPrincipal: #ABABAB;
@anchoSitio: 1024px;
@estiloBorde: dotted;

body{
  color: @colorPrincipal;
  border: 1px @estiloBorde @colorPrincipal;
  max-width: @anchoSitio;
}

button{
  background-color: @colorPrincipal;
}
```

Archivo .CSS

```
body {
  color: #ababab;
  border: 1px dotted #ababab;
  max-width: 1024px;
}

button {
  background-color: #ababab;
}
```



12. PREPROCESADORES CSS - LESS

MIXINS

Un mixin en LESS es un conjunto de propiedades CSS agrupadas. Esto nos permite utilizar el grupo en sí y no repetir mil veces las mismas propiedades.

Si estuviésemos hablando de programación (como tal), esto sería lo más parecido a un array. Otra forma de entenderlo podría ser una variable con múltiples valores.

Como dije en el capítulo anterior de este tutorial, los mixins son, junto a las variables y funciones, uno de los tres principales pilares de LESS.

```
.border-radius(@radius) {  
    -webkit-border-radius: @radius;  
    -moz-border-radius: @radius;  
    border-radius: @radius;  
}  
  
.sidebar {  
    .border-radius(4px);  
}
```

```
.border-radius(@radius: 6px) {  
    -webkit-border-radius: @radius;  
    -moz-border-radius: @radius;  
    border-radius: @radius;  
}  
  
.sidebar {  
    .border-radius;  
}  
  
.sidebar2 {  
    .border-radius(12px);  
}
```

```
.sidebar {  
    -webkit-border-radius: 6px;  
    -moz-border-radius: 6px;  
    border-radius: 6px;  
}  
.sidebar2 {  
    -webkit-border-radius: 12px;  
    -moz-border-radius: 12px;  
    border-radius: 12px;  
}
```



12. PREPROCESADORES CSS - LESS

CÓDIGO ANIDADO

Un problema común al generar hojas CSS surge cuando existen múltiples elementos con el mismo padre. Resulta muy tedioso tener que estar escribiendo el nombre del objeto padre repetidamente.

Código CSS

```
.boton-solitario {  
  color: blue;  
}  
.boton-solitario:hover {  
  background: red;  
}  
.contenedor .boton {  
  color: #114477;  
  width: 200px;  
  height: 50px;  
}  
.contenedor .boton:hover {  
  color: #774411;  
  background: green;  
}
```

Código LESS

```
.boton-solitario {  
  color: blue;  
  &:hover {  
    background: red;  
  }  
}  
.contenedor {  
  .boton {  
    color: #147;  
    width: 200px;  
    height: 50px;  
    &:hover {  
      color: #741;  
      background: green;  
    }  
  }  
}
```



12. PREPROCESADORES CSS - LESS

FUNCIONES DE COLORES

Las funciones de colores sirven para transformar el color en el momento de la compilación. Son bastante útiles para crear degradados, oscurecer un botón al pasar el ratón, etc..

lighten(@color, 10%);	<i>/* devuelve un color 10% más claro que @color */</i>
darken(@color, 10%);	<i>/* devuelve un color 10% más oscuro que @color */</i>
saturate(@color, 10%);	<i>/* devuelve un color 10% más saturado que @color */</i>
desaturate(@color, 10%);	<i>/* devuelve un color 10% menos saturado que @color */</i>
spin(@color, 10%);	<i>/* devuelve un color 10% más de tono que @color */</i>
desaturate(@color, -10%);	<i>/* devuelve un color 10% menos de tono que @color */</i>
mix(@color1, (@color2);	<i>/* devuelve un color mezcla de ambos */</i>

<https://programandoointentandolo.com/2017/10/tutorial-less-13-funciones-colores-less.html>



12. PREPROCESADORES CSS - LESS

OPERACIONES

Permite resolver operaciones matemáticas.

código LESS

```
body{  
  margin: (14px/2);  
  top: 50px + 100px;  
  right: 100px - 50px;  
  left: 10 * 10;  
}
```

código CSS

```
body {  
  margin: 7px;  
  top: 150px;  
  right: 50px;  
  Left: 100;  
}
```

MEDIA QUERIES

Permite anidar media queries

código LESS

```
.contenedor{  
  width: 75%;  
  @media (max-width: 800px){  
    width: 100%;  
  }  
}
```

código CSS

```
.contenedor {  
  width: 75%;  
}  
@media (max-width: 800px) {  
  .contenedor {  
    width: 100%;  
  }  
}
```



12. PREPROCESADORES CSS

ENLACES DE INTERÉS

<https://programandoointentandolo.com/2017/07/tutorial-less-1-introduccion-less.html>

<https://abalozz.es/que-es-un-preprocesador-de-css/>

<https://medium.com/@sergiodxa/ventas-y-desventajas-de-los-pre-procesadores-de-css-6528fdac9926>

<https://www.acens.com/wp-content/images/2016/07/wp-presprocesadores-css-acens.pdf>

<http://maquetando.com/sass-y-less-el-presente-de-css>

<https://amatellanes.github.io/lesscss.org/>