# PREPROCESADOR SASS

# METODOLOGÍA BEM

Y FLEX

Tabla de contenido

[PREPROCESADOR SASS 1](#_Toc27679972)

[METODOLOGÍA BEM 1](#_Toc27679973)

[1. Preprocesadores SASS 2](#_Toc27679974)

[1.1 Primeros pasos con SASS. 2](#_Toc27679975)

[1.2 Creación de proyectos scss. 3](#_Toc27679976)

[1.3 Empezamos a codificar en SASS 4](#_Toc27679977)

[1.4 Variables en SCSS 6](#_Toc27679978)

[1.5 Importaciones 7](#_Toc27679979)

[1.6 Operadores 8](#_Toc27679980)

[1.7 Reutilización de código 10](#_Toc27679981)

[**1.7.1** **Mixings** 10](#_Toc27679982)

[**1.7.2** **Herencia** 12](#_Toc27679983)

[**1.7.3** **Referencia al selector padre** 12](#_Toc27679984)

[2. CONTENEDORES FLEX 14](#_Toc27679985)

[**Propiedad align-items | align-self | align-content:** 19](#_Toc27679986)

[3. Media queries 27](#_Toc27679987)

[4. Contadores 28](#_Toc27679988)

[5. Metodología BEM 29](#_Toc27679989)

[6. Github Pages 30](#_Toc27679990)

[7. Bibliografía: 31](#_Toc27679991)

# Preprocesadores SASS

Los preprocesadores son herramientas que nos permiten conseguir código limpio que previamente es procesado por la herramienta. Nos dan una serie de nuevas funcionalidades que nos permitirán desarrollar código de una manera mucho más potente.

El preprocesador que hemos utilizado en clase es SASS, pero existen multitud de ellos como por ejemplo LESS.

## **Primeros pasos con SASS.**

Lo primero que hemos de hacer para poder utilizar SASS es instalarlo, para ello hay diferentes formas, como ahora estoy con mi equipo voy a realizar la explicación de instalación para Mac OS.

La instalación en Mac también se puede realizar de diferentes formas, la que yo he realizado es la instalación por la terminal usando el gestor de paquetes homebrew.

Para instalarlo es necesario ejecutar el siguiente comando:

**Brew install sass/sass/sass**

Imagen que contiene captura de pantalla, monitor, laptop, pantalla

Descripción generada automáticamente

Una vez ejecutado este comando y suponiendo que no se ha producido ningún error se habrá instalado correctamente el preprocesador SASS.

Para comprobar que se ha instalado correctamente podemos ejecutar el comando: **sass –versión**

Captura de pantalla de computadora

Descripción generada automáticamente

## **Creación de proyectos scss.**

Para poder utilizar sass hay que crear un fichero con extensión scss. En esencia el código que generamos tiene la misma sintaxis que CSS pero tendremos más funcionalidades.

Por una razón de orden, vamos a crear siempre una jerarquía de directorios dentro de nuestros proyectos. En ese sentido la jerarquía será la siguiente, una carpeta procesar donde irán nuestros ficheros con extensión scss y otra carpeta subir que tendrá los ficheros de css que han sido preprocesados y el html.

Voy a hacer un tree para ver de una manera visual la jerarquía:

Imagen que contiene dibujo

Descripción generada automáticamente

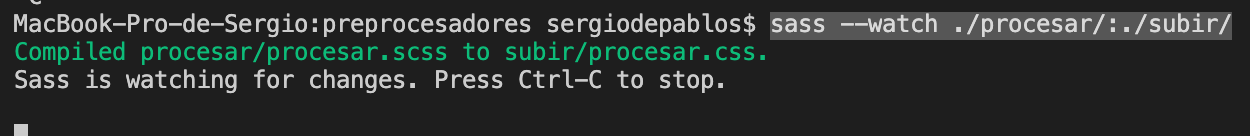
Ya hemos explicado la jerarquía que vamos a seguir, ahora es necesario explicar el comando necesario para procesar el fichero scss y generar el css.

Es posible procesar por ficheros concretos o por directorios, entonces procesará todos los ficheros scss que haya en ese directorio.

Nosotros hemos utilizado siempre la opción de procesar directorios.

Para poder hacerlo hemos utilizado el siguiente comando:

**sass --watch ./procesar/:./subir/**



Como vemos ahí nos avisa de que sass esta vigilando cambios.

## **Empezamos a codificar en SASS**

Lo primero que vamos a realizar en SASS va a ser las anidaciones. Es un uso muy habitual que a la hora de escribir código en sass realicemos **anidaciones**, pudiendo hacer estas anidaciones con distintos tipos de selectores y de resultados.

Vamos con una **anidación sencilla**:

div {

color: burlywood;

width: 80%;

height: auto;

.cajaroja {

background-color: crimson;

color: #000000;

width: 50%;

height: 400px;

p {

color: yellow;

}

}

}

Como vemos en el código de ejemplo, seleccionamos todos los div, luego todos los elementos .cajaroja cuyo padre es un div y después todos los p cuyo padre es .cajaroja y su abuelo es un div.

Este código scss tiene la siguiente salida procesada en el archivo css:

div {

color: burlywood;

width: 80%;

height: auto;

}

div .cajaroja {

background-color: crimson;

color: #000000;

width: 50%;

height: 400px;

}

div .cajaroja p {

color: yellow;

}

Este código es el que se genera al procesar nuestro archivo scss.

Para completar el ejemplo he realizado un simple html con el que probarlo. La salida en el navegador en la [siguiente](preprocesadores/subir/ejercicio1.html).

Vamos ahora con otro ejemplo de anidaciones:

.contenedor {

background-color: dodgerblue;

> div {

background-color: #f1f1f1;

width: 140px;

margin: 10px;

padding: 20px;

font-size: 30px;

line-height: 75px;

text-align: center;

}

}

Aquí estamos realizando una anidación con un selector de descendientes con el símbolo mayor.

Estamos diciendo que todos los div que sean hijos directos de un elemento con la clase .contenedor se les apliquen todos esos estilos.

Esto está escrito en SCSS, una vez procesado el resultado sería el siguiente:

.contenedor {

background-color: dodgerblue;

}

.contenedor > div {

background-color: #f1f1f1;

width: 140px;

margin: 10px;

padding: 20px;

font-size: 30px;

line-height: 75px;

text-align: center;

}

He creado un html sencillo para probar este ejemplo. Podemos comprobar la salida en el navegador en el siguiente [enlace](anidaciones/subir/ejercicio.html).

## **Variables en SCSS**

Las variables en scss nos permiten almacenar multitud de contenido que se puede reutilizar, podemos así almacenar colores, fuentes, valores…

Vamos con un ejemplo en el que vamos a guardar una fuente en una variable.

Este es el código scss:

@font-face{

font-family: mifuente1;

src: url("fonts/IndieFlower-Regular.ttf");

}

$fuente1 : 'Gill Sans', 'Gill Sans MT', 'Calibri', 'Trebuchet MS', sans-serif;

$fuente2 : mifuente1;

div{

width: 80%;

min-height: 400px;

background-color: azure;

margin-left: auto;

margin-right: auto;

border: 1px solid blueviolet;

.cajaroja{

width: 60%;

min-height: 200px;

background-color: red;

margin-left: auto;

margin-right: auto;

font-family: $fuente2;

}

}

El código una vez procesado queda de la siguiente forma:

@font-face {

font-family: mifuente1;

src: url("fonts/IndieFlower-Regular.ttf");

}

div {

width: 80%;

min-height: 400px;

background-color: azure;

margin-left: auto;

margin-right: auto;

border: 1px solid blueviolet;

}

div .cajaroja {

width: 60%;

min-height: 200px;

background-color: red;

margin-left: auto;

margin-right: auto;

font-family: mifuente1;

}

/\*# sourceMappingURL=hojaestilo1.css.map \*/

Por último he generado un html para probarlo, puedes ver la salida en el navegador del ejemplo en el siguiente [enlace](variables/subir/ejercicio1.html).

Con esto ya hemos visto como funcionan las variables, pero veremos más ejemplos según vaya desarrollándose el miniproyecto.

## **Importaciones**

Mediante Sass podemos realizar importaciones de ficheros. Esto es útil pues podemos importar ficheros pequeños que una vez importados en otro fichero compondrán un fichero scss de mayor tamaño. Esto beneficia el mantenimiento del código, ya que modificar un fichero pequeño siempre será más sencillo que una de gran tamaño.

Es importante tener en cuenta que a la hora de procesar los ficheros dentro de un directorio, es posible indicar al watch de sass que no procese ese fichero. Esto se consigue nombrando al fichero con el carácter \_ y el nombre del fichero que queramos dar.

Vamos a hacer un ejemplo. Creamos el fichero \_importado.scss y le vamos a introducir el siguiente contenido:

html,

body,

ul,

ol {

margin: 0;

padding: 0;

letter-spacing: 1em;

}

Ahora vamos a crear otro fichero scss donde vamos a importar el contenido del fichero que acabamos de crear.

Para importar un fichero utilizamos el siguiente código: **@import nombredefichero.scss**

@import "importaciones.scss";

$fuente1: Helvetica, sans-serif;

$colorprincipal: rgb(187, 57, 57); /\*En este ejemplo vamos a aprender una nueva propiedad: font\*/

body {

font: 1em $fuente1;

color: $colorprincipal;

}

Ahora vamos a hacer un archivo html para poder probar el código. Podemos ver la salida en el navegador en el siguiente [enlace](importaciones/subir/ejercicio.html).

## **Operadores**

Las operaciones matemáticas en css son de gran utilidad. Por ejemplo podemos hacer que las proporciones de un contenedor sean exactas y no queden bailando pequeñas cantidades.

Los operadores que tenemos son los siguientes: **+, -, \*, / y %**

Hay que tener en cuenta que las operaciones que podemos hacer solo se pueden realizar con unidades de medida que sean compatibles

Ejemplo de operadores:

h2 {

font-size: 5px + 2em; /\*Aquí hay un error. Unidades incompatibles\*/

font-size: 5px + 2; /\*El resultado sería 7px\*/

}

h3 {

font-size: 5px \* 2px; /\*Esto arrojaría un error. No se pueden multiplicar dos números que tengan la misma unidad\*/

font-size: 5px \* 2;

}

Podemos ver la salida de este código en el navegador en el siguiente [enlace](operadores/subir/ejercicio.html).

Además de operadores, vamos a tratar sobre roles, que es otra forma de acceder a un elemento concreto del html que tenga asociado ese mismo rol. Para ello en el html hace falta poner: role = “nombreRol”

Tras eso en el archivo scss hay que acceder con el nombre del elemento y [role=”nombreRol”].

Vamos con un ejemplo para verlo más claro.

Este es el código scss

.container {

width: 100%;

}

h2[role="titulo"] {

float: left;

width: 600px / 960px \* 100%;

}

h3[role="subtitulo"] {

float: right;

width: 360px / 960px \* 100%;

}

El código CSS generado es el siguiente:

.container {

width: 100%;

}

h2[role=titulo] {

float: left;

width: 62.5%;

}

h3[role=subtitulo] {

float: right;

width: 37.5%;

}

/\*# sourceMappingURL=hojaestilos2.css.map \*/

Como vemos, se han creado dos roles, título y subtítulo. Así es como seleccionamos esos elementos.

La salida en el navegador de este ejercicio esta en el siguiente [enlace.](operadores/subir/ejercicio2.html)

## **Reutilización de código**

Una de las principales funcionalidades de Sass es la capacidad que nos da para poder reutilizar código. Así nos da la opción de evitar escribir tanto código.

Las dos opciones que conocemos para reutilizar código son **mixings y herencias.**

Vamos a tratar cada uno de estos conceptos por separado:

### **Mixings**

Son similares a una función en un lenguaje de programación, se declaran en nuestro documento y luego podemos incluir en cualquier sección.

La gran diferencia con la herencia es que estos mixings pueden recibir parámetros de entrada, ya sean estos valores, variables, colores, fuentes…

Normalmente usaremos la herencia porque el código que genera es más óptimo que el código generado por los mixings. Pero será de utilidad cuando necesitemos dar valores como parámetro de entrada o hacer cálculos más complejos.

Vamos ahora a ver algún ejemplo de como se utilizan los mixings:

@mixin borderradius($radio, $color) {

-webkit-border-radius: $radio;

-moz-border-radius: $radio;

-ms-border-radius: $radio;

border-radius: $radio;

width: 300px / 900px \*100%;

float: left;

box-sizing: border-box;

border: 1px solid $color;

}

%texto{

font-family: Verdana, Geneva, Tahoma, sans-serif;

font-size: 1.5rem;

color: royalblue;

text-decoration: underline red;

}

.capaborderradius1{

@include borderradius(5px, blue);

&:hover{

$color: random(99);

$color1: random(99);

$color2: random(99);

background-color: rgb($color, $color1, $color2);

}

&\_\_texto1{

@extend %texto;

}

}

.capaborderradius2{

@include borderradius(10px, red);

&\_\_texto1{

@extend %texto;

}

}

.capaborderradius3{

@include borderradius(15px, green);

&\_\_texto1{

@extend %texto;

}

}

Lo primero que hacemos es declarar un mixing y luego una herencia, este ejemplo nos servirá para ambos ejemplos de reutilización de código. Pero de momento nos centramos en el mixing

Para declarar el mixing hay que usar la clausula **@mixin nombreMixing (parametros){}**

Una vez que lo tenemos declarado ya lo podemos usar en cualquier parte de nuestro código utilizando la clausula **@include nombreMixing(parametros);**

Podemos utilizar este mixing las veces que sea necesario.

La parte concreta del ejemplo que es el mixing es esta:

@mixin borderradius($radio, $color) {

-webkit-border-radius: $radio;

-moz-border-radius: $radio;

-ms-border-radius: $radio;

border-radius: $radio;

width: 300px / 900px \*100%;

float: left;

box-sizing: border-box;

border: 1px solid $color;

}

### **Herencia**

Como ya introduje anteriormente la herencia sirve para reutilizar código, pero de una manera más óptima que el mixing. Lo usaremos siempre que no necesitemos poner parámetros de entrada.

Para declarar una herencia solo es necesario el carácter **%nombreDeLaHerencia{}**

%texto{

font-family: Verdana, Geneva, Tahoma, sans-serif;

font-size: 1.5rem;

color: royalblue;

text-decoration: underline red;

}

Así ya habremos declarado una herencia.

Para hacer referencia a esa herencia usamos la clausula **@extend %nombreHerencia**

&\_\_texto1{

@extend %texto;

}

### **Referencia al selector padre**

Antes de abandonar la sección de reutilización de código vamos a tratar una notación que nos sirve para hacer referencia al selector padre dentro de una anidación. El carácter utilizado para este cometido es &

Por ejemplo:

.capaborderradius1{

@include borderradius(5px, blue);

&:hover{

$color: random(99);

$color1: random(99);

$color2: random(99);

background-color: rgb($color, $color1, $color2);

}

&\_\_texto1{

@extend %texto;

}

}

Podemos ver que hacemos un hover del padre de la anidación que es la clase capaborderradius.

Además esta parte del código es curiosa porque cada vez que se procesa se genera un nuevo color de fondo.

Ahora vamos a ver el css que se genera tras el ejemplo completo que pusimos anteriormente:

.capaborderradius3\_\_texto1, .capaborderradius2\_\_texto1, .capaborderradius1\_\_texto1 {

font-family: Verdana, Geneva, Tahoma, sans-serif;

font-size: 1.5rem;

color: royalblue;

text-decoration: underline red;

}

.capaborderradius1 {

-webkit-border-radius: 5px;

-moz-border-radius: 5px;

-ms-border-radius: 5px;

border-radius: 5px;

width: 33.3333333333%;

float: left;

box-sizing: border-box;

border: 1px solid blue;

}

.capaborderradius1:hover {

background-color: #282c58;

}

.capaborderradius2 {

-webkit-border-radius: 10px;

-moz-border-radius: 10px;

-ms-border-radius: 10px;

border-radius: 10px;

width: 33.3333333333%;

float: left;

box-sizing: border-box;

border: 1px solid red;

}

.capaborderradius3 {

-webkit-border-radius: 15px;

-moz-border-radius: 15px;

-ms-border-radius: 15px;

border-radius: 15px;

width: 33.3333333333%;

float: left;

box-sizing: border-box;

border: 1px solid green;

}

/\*# sourceMappingURL=hojaEstilo.css.map \*/

Como podemos observar la herencia agrupa código, mientras que los mixing lo generan para cada elemento.

La salida en el navegador de todo este ejemplo está en el siguiente [enlace](reutilizaciondecodigo/subirServidor/ejercicio1.html).

## **CONTENEDORES FLEX**

Flexbox viene de “Flexible Box Layout“, que se puede traducir como “Diseño de caja flexible“, y nos aporta una magnífica solución para todos nuestros desarrollos “responsive”. Lo que nos permite es crear un conjunto de elementos flexibles que se adaptan automáticamente a su contenedor y con el que podemos controlar parámetros tales como la alineación, dirección (horizontal/vertical), ajuste de la fila según tamaños y multitud de posibilidades que vamos a presentar en este artículo.

En cuanto a compatibilidad, Flexbox es actualmente **compatible con los navegadores web más importantes** y diferentes versiones de los mismos, si bien en algunos debemos utilizar prefijos css para su buen funcionamiento

Vamos a empezar a ver ejemplos de flex.

Lo primero que se tiene que ver es como poner un elemento como flex. Para ello usamos **display:flex**;

Vamos con el primer ejemplo de un contenedor flex compuesto por más elementos:

.container{

display:flex;

background-color: brown;

> div{

width:25%;

background-color: aqua;

border: 1px solid black;

text-align: center;

}

}

Como vemos, a la clase container le ponemos el **display flex**. A los hijos directos le damos un ancho fijo, pero al estar dentro de un contenedor flex van a ocupar el espacio necesario para que los elementos ocupen el ancho completo del navegador independientemente del ancho que tienen fijado. La salida en pantalla es la siguiente:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Puede verse en el navegador en el siguiente [enlace](flex1/subir/ejercicio.html).

**Propiedad flex-direction:**

Puede tener los siguientes valores.

– **flex-direction:row;** -> Los elementos se visualizan **de izquierda a derecha** (valor por defecto, similar al ejemplo 1)Captura de pantalla de un celular con letras

Descripción generada automáticamente

– **flex-direction:row-reverse;** -> Los elementos se visualizan **de derecha a izquierda**.Imagen que contiene captura de pantalla

Descripción generada automáticamente

– **flex-direction:column;** -> Los elementos se visualizan **de arriba hacia abajo**.Captura de pantalla de un celular con letras

Descripción generada automáticamente

– **flex-direction:column-reverse;** -> Los elementos se visualizan **de abajo hacia arriba**.Captura de pantalla de un celular con letras

Descripción generada automáticamente

**Propiedad flex-wrap**

Esta propiedad afecta a cómo se distribuyen los elementos en fila y, por consiguiente, a su tamaño. Los posibles valores son:

– **flex-wrap:nowrap;** -> Los elementos se muestran en línea, en una sola fila, y su tamaño se ajusta al contenedor siempre y cuando la suma de todos ellos sea mayor o igual que el 100% de la anchura del contenedor. Si es inferior, se siguen mostrando en línea pero conservan su tamaño. Este es el valor por defecto, y como veíamos en el *ejemplo 1*, aunque la anchura de los elementos es el 25% del contenedor, todos se muestran en línea modificando su tamaño para que la suma total no sea superior al 100% de su contenedor.

Captura de pantalla de un celular con letras

Descripción generada automáticamente

– **flex-wrap:wrap; ->** Los elementos se muestran en línea, pero si su anchura supera la delcontenedor, se distribuyen en varias filas.Captura de pantalla de un celular

Descripción generada automáticamente

– **flex-wrap:wrap-reverse; ->** Los elementos se muestran en línea, pero si su anchura supera la del contenedor, se distribuyen en varias filas, y además lo hacen en orden inverso al de maquetación.

Captura de pantalla de un celular

Descripción generada automáticamente

**Propiedad justify-content**

Esta propiedad la vamos a usar para alinear elementos horizontalmente. Los valores más utilizados son los siguientes:

* – **justify-content:flex-start;** -> Alinea los elementos en horizontal desde el inicio de la dirección del eje principal de su contenedor (partiendo desde el inicio de la línea). Este es el valor por defecto. Es importante destacar que, como veremos más adelante, **la dirección establecida en “flex direction” afecta a la alineación**. Con alineación row.Captura de pantalla de un celular

  Descripción generada automáticamente

Ahora si ponemos un row-reverse:

Captura de pantalla de un celular

Descripción generada automáticamente

* – **justify-content:flex-end;** -> Alinea los elementos en horizontal desde el final de la dirección del eje principal de su contenedor (partiendo desde el final de la línea) Captura de pantalla con letras y números

  Descripción generada automáticamente
* – **justify-content:center;** -> Alinea los elementos al centro del eje principal de su contenedor. Similar a un texto alineado al centro.Imagen que contiene captura de pantalla, dibujo, jugador

  Descripción generada automáticamente
* – **justify-content:space-between;** -> Alinea los elementos justificándolos a lo largo del eje principal de su contenedor. Similar a un texto justificado. Los elementos laterales se pegan a los extremos y el resto se distribuyen a lo largo del eje principal dejando el mismo espacio entre ellos.Captura de pantalla de un celular

  Descripción generada automáticamente
* – **justify-content:space-around;** -> Alinea los elementos distribuyendo sus centros de forma horizontal a lo largo del eje principal de su contenedor, dejando el mismo espacio lateral de separación al comienzo, al final y entre ellos.

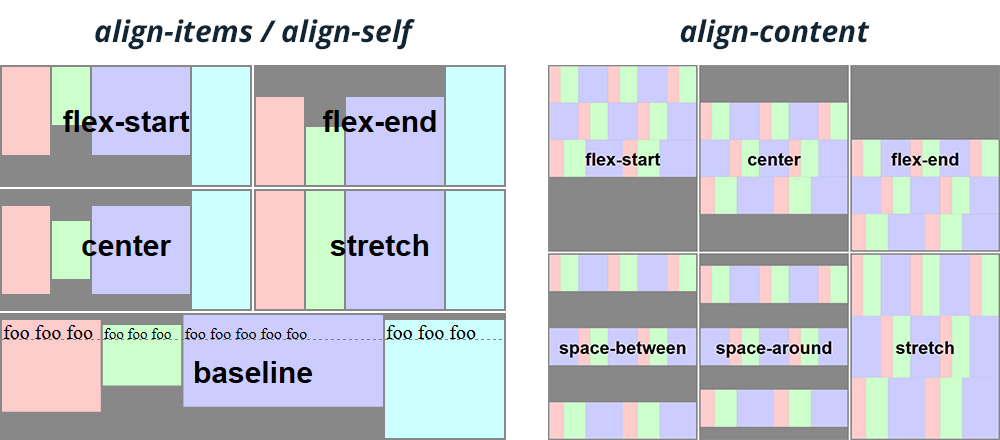
Captura de pantalla de un celular

Descripción generada automáticamente

### **Propiedad align-items | align-self | align-content:**

La **alineación vertical** se realiza a través del llamado **“eje transversal” (cross axis)**, y para ello contamos con tres propiedades diferentes, “align-items”, “align-self” y “align-content”. Aquí viene el lío, ya que si no se entiende el concepto muchas veces se obtienen resultados inesperados. Vamos a ver qué nos cuenta el W3C sobre estas propiedades:

* – **“align-items”** establece la alineación predeterminada para todos los elementos del contenedor, incluidos los elementos independientes.
* – **“align-self”** permite alinear elementos independientes del contenedor.
* – **“align-content”** alinea las líneas/filas de elementos de un contenedor.



Por lo tanto, “**align-items**” nos sirve para **alinear los elementos** y “**align-content**” para **alinear las filas** de éstos, mientras que “**align-self**” nos permite **alinear elementos de forma independiente**. Sabiendo esto, podemos deducir que “align-content” sólo funciona cuando tenemos más de una fila de elementos. La imagen superior representa muy bien cómo funcionan estas propiedades. A continuación veremos los valores que pueden tomar cada una de ellas (los valores de “align-items” son válidos para “align-self”).

Vamos ahora con los primeros ejemplos. Vamos a probar la propiedad align-items y su valor stretch.

– **align-items:stretch;** -> Valor por defecto. La altura de los elementos se ajusta al tamaño del contenedor (o fila), dividiendo el espacio sobrante entre todos los elementos por igual.

Imagen que contiene naranja, brillando, colorido, cuarto

Descripción generada automáticamente

– **align-items:flex-start;** -> Alinea en vertical los elementos desde el inicio de la dirección del eje transversal de su contenedor (al igual que ocurría en horizontal). También **afecta el valor de “flex-direction” al sentido de la alineación vertical**.

Captura de pantalla de un celular

Descripción generada automáticamente

– **align-items:flex-end;** -> Alinea en vertical los elementos desde el final de la dirección del eje transversal de su contenedor (al igual que ocurría en horizontal).

Captura de pantalla de un celular

Descripción generada automáticamente

– **align-items:center;** -> Alinea al centro vertical los elementos a lo largo del eje transversal de su contenedor.

Captura de pantalla de un celular

Descripción generada automáticamente

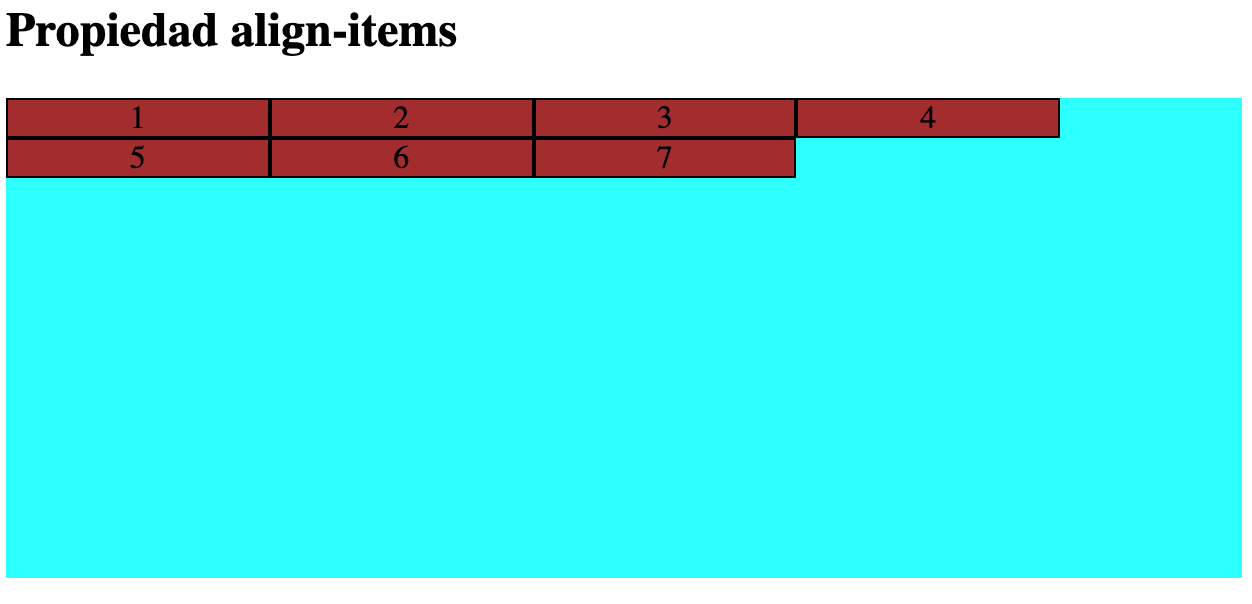
– **align-items:baseline;** -> Alinea en vertical las “líneas base” de los elementos a lo largo del eje transversal de su contenedor. Para poder comprender este ejemplo, vamos a añadir algunos estilos diferentes a los elementos:

Captura de pantalla de un celular con letras

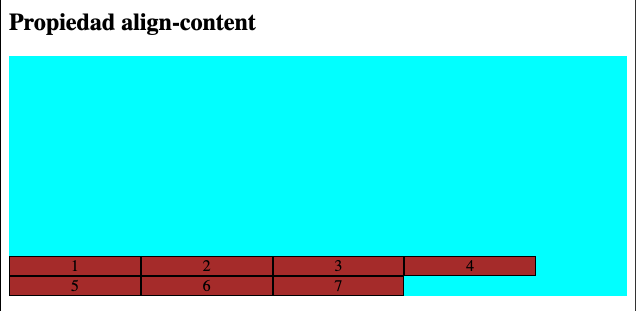
Descripción generada automáticamente

**Propiedad align-content**

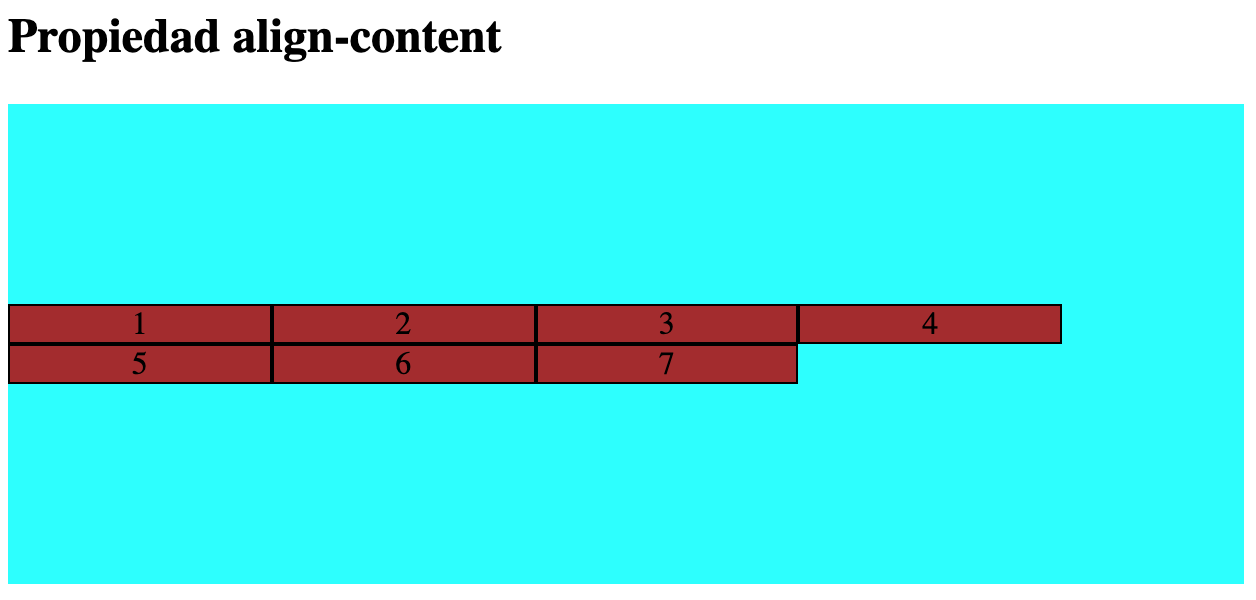
– **align-content:flex-start;** -> Alinea en vertical las filas desde el inicio de la dirección del eje transversal de su contenedor.



– **align-content:flex-end;** -> Alinea en vertical las filas desde el final de la dirección del eje transversal de su contenedor.



– **align-content:center;** -> Alinea al centro vertical las filas a lo largo del eje transversal de su contenedor.



– **align-content:space-between;** -> Alinea las filas justificándolas a lo largo del eje transversal de su contenedor. Similar a un texto justificado en vertical. Las filas superior e inferior se pegan a sus extremos y el resto se distribuyen a lo largo del eje transversal dejando el mismo espacio entre ellas.

Imagen que contiene captura de pantalla, dibujo

Descripción generada automáticamente

– **align-content:space-around;** -> Alinea las líneas distribuyendo sus centros de forma vertical a lo largo del eje transversal de su contenedor, dejando el mismo espacio vertical de separación en la parte superior, inferior y entre ellos.

Captura de pantalla con letras y números

Descripción generada automáticamente

**Propiedad align-self:**

“**align-self**” nos permite **alinear elementos de forma independiente**.

Imagen que contiene pantalla, televisión, monitor, negro

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

**flex:**

A continuación veremos la propiedad “**flex**” que se aplica a los elementos del contenedor y que a su vez se estructura de tres propiedades diferentes y que se pueden presentar por separado, **“flex- grow”, “flex-shrink” y “flex-basis”**, y se define de la siguiente forma:

* – **flex: none | <flex-grow> <flex-shrink> <flex-basis>**

Vamos a ver las propiedades por separado:

* – flex-grow:<número>; -> Determina el factor de incremento de tamaño de uno o varios elementos en relación al resto teniendo en cuenta el espacio “vacío” o “libre” entre ellos. Como espacio “vacío/libre” nos referimos al espacio que no ocupan los elementos en relación a su fila. Cuando este valor se omite, se establece en 1.
* – flex-shrink:<número>; -> Determina el factor de disminución de tamaño de uno o varios elementos en relación al resto teniendo en cuenta el espacio “vacío” o “libre” entre ellos. Cuando este valor se omite, se establece en 1.
* – flex-basis: auto | 0 | <width> -> Especifica el tamaño principal inicial de los elementos teniendo en cuenta el espacio “vacío” o “libre” entre ellos. Cuando este valor se omite, se establece en 0. Esta propiedad hace que, dependiendo del valor asignado, el tamaño “base” se interprete como “**todo el espacio**” del elemento o como el “**espacio sobrante**” del mismo. A continuación vemos un gráfico donde visualmente se explica cómo se interpretan estos valores en relación al tamaño del elemento.

Captura de pantalla de un celular

Descripción generada automáticamente

Esta propiedad “flex” puede funcionar de diferentes formas dependiendo del valor que establezcamos y no podemos presentar todos los posibles ejemplos que pueden darse.

.contenedorflex{

display: flex;

width: 500px;

background-color: moccasin;

padding: 10px;

min-height: 300px;

> .elemento1{

flex-basis: 200px;

flex-grow: 1;

background-color: lightgreen;

}

> .elemento2{

flex-basis: auto;

flex-grow: 2;

background-color: lightgreen;

}

> .elemento3{

flex-basis: auto;

flex-grow: 2;

background-color: lightgreen;

}

Captura de pantalla de un celular

Descripción generada automáticamente

## **Media queries**

Las media queries se pueden definir como código CSS condicional, es decir, que solo se aplicará si cumple con una determinada condición.

Su mayor aplicación es la de adaptar el contenido de la página para diseño responsivo.

Lo normal es usar medidas en pixeles para determinar cuando se aplica el código y se utilizará respectivamente las clausulas max-width y min-width.

**Max-width** hace referencia a cuando el tamaño de la pantalla es tantos pixeles de ancho o menos.

**Min-width** hace referencia a cuando el tamaño es tantos pixeles de ancho o más.

Para declarar una media querie es necesario usar @media (condición a cumplir){}

Con esto ya se aplicrará nuestro código en función de la condición que hayamos puesto.

Vamos con un ejemplo sencillo de uso de media querie

.contenedor {

overflow: auto;

}

#principal {

margin-left: 4px;

}

#barralateral {

float: none;

width: auto;

}

#listamenu {

margin: 0;

padding: 0;

}

.elementomenu {

background: #cdf0f6;

border: 1px solid #d4d4d4;

border-radius: 4px;

list-style-type: none;

margin: 4px;

padding: 2px;

}

@media screen and (min-width: 480px) {

#barralateral {

width: 200px;

float: left;

}

#principal {

margin-left: 216px;

}

}

Aquí creamos un menú que el ancho de la pantalla desciende por debajo de los 480px entonces el menú deja de flotar a la izquierda y se coloca en su posición natural.

Podemos ver la salida por pantalla de este ejemplo en el siguiente [enlace](mediaqueries/subir/ejercicio.html).

## **Contadores**

Los contadores son variables en CSS que nos pueden ayudar a la hora de organizar los contenidos de la web y mostrar listas personalizadas con gran detalle a la hora de enumerar sus contenidos. Tenemos varias propiedades para utilizar este tipo de variables:

. counter-reset – Crea o reinicia el valor de un contador.

· counter-increment – Incrementa el valor de un contador.

· content – Inserta un contenido personalizado por el desarrollador.

· counter() – Añade el valor de un contador a un elemento.

Ejemplo:

Captura de pantalla de un celular

Descripción generada automáticamente

Salida en el navegador en el primer [enlace](contadores/subir/ejercicio.html).

## Metodología BEM

La metodología BEM es una forma de notación que facilita la lectura y entendimiento del código.

Se caracteriza sobretodo por 3 grandes reglas:

/\* Bloque de contenido reutilizable \*/

.btn {

} /\* Elemento hijo que depende de un bloque de contenido \*/

.btn\_\_precio {

} /\* Modificador que cambia el estilo de un bloque \*/

.btn--naranja {

}

.btn--grande {

}

<a class="btn btn--grande btn--naranja" href="https://css-tricks.com">

<span class="btn\_\_precio">9.99€</span>

<span class="btn\_\_text">Pagar</span>

</a>

Tiene gran uso con Sass porque se puede anidar el código de una manera sencilla. Por ejemplo:

.btn{

&--modify{

}

&\_\_dependiente{

}

Sass creará las clases:

* + .btn—modify
  + .btn\_\_dependiente

## **Github Pages**

Una de las nuevas funcionalidades que ha sido añadido a github es la posibilidad de testear nuestras páginas web y poder ver el resultado. Sería como subir nuestros archivos a un hosting.

Para poder llevar a cabo esto, es necesario crear un repositorio nuevo pero con un nombre concreto que va a ser: nombreUsurio.github.io

Captura de pantalla de un celular

Descripción generada automáticamente

En mi caso sería serdpa08.github.io

Para subir los archivos es necesario hacer un push al repositorio, teniendo en cuenta que será necesario tener un archivo index.html para visualizar el contenido.

Una vez subido solo tendremos que buscar en nuestro navegador la url que será el nombre del repositorio que creamos previamente.

Captura de pantalla de un celular

Descripción generada automáticamente

## Bibliografía:

* + Apuntes de clase.
  + <https://www.yunbitsoftware.com/blog/2017/03/30/flexbox-css3-tutorial-descripcion-ejemplos-html/>