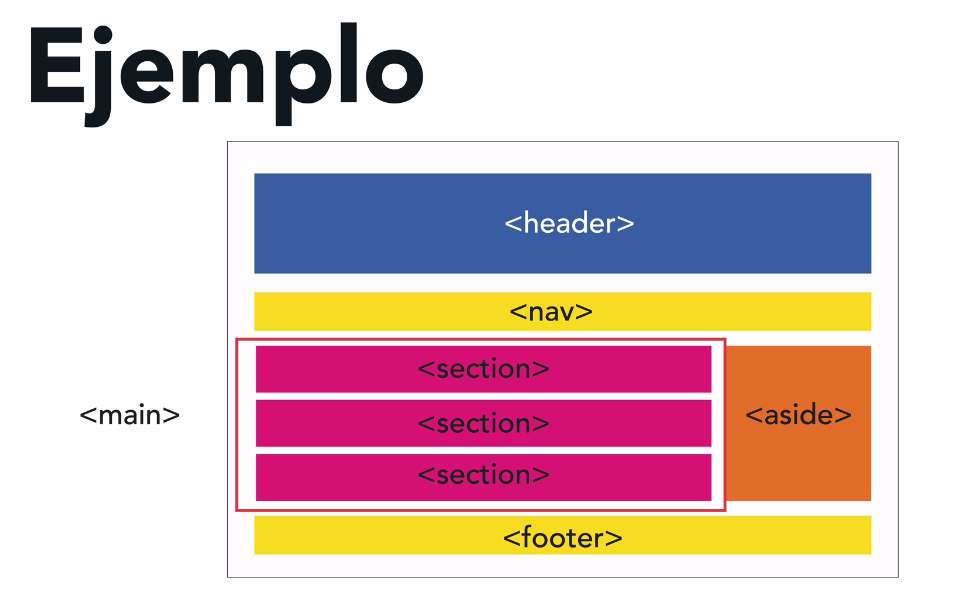
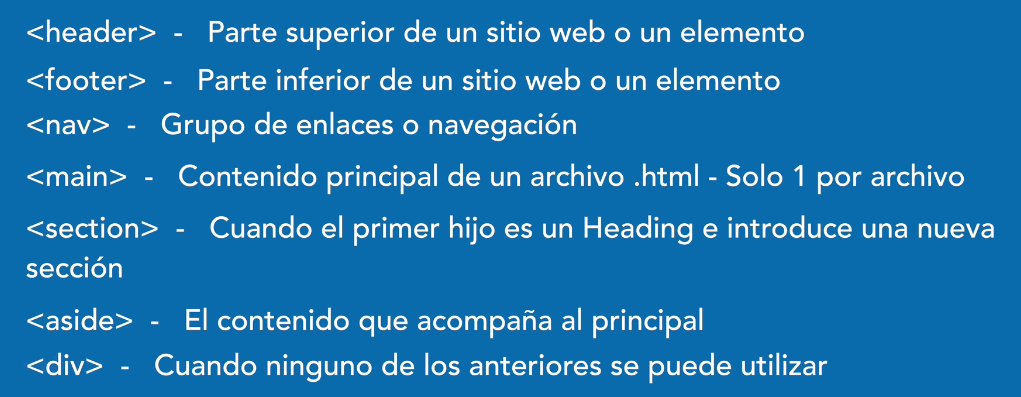
**HTML & CSS**

|  |
| --- |
| Estructurar HTML |





**Validate HTML 🡪** [**https://validator.w3.org/#validate\_by\_input**](https://validator.w3.org/#validate_by_input)

|  |
| --- |
| CSS |

**MODULE, BEM, SMACSS**

**Box model**

* **The size is determined by: content + padding + border + margin**
* **box-sizing: border-box (recommended way, box-sizing: content-box is default)**

[**https://www.paulirish.com/2012/box-sizing-border-box-ftw/**](https://www.paulirish.com/2012/box-sizing-border-box-ftw/)

**Normalize 🡪** [**https://necolas.github.io/normalize.css/**](https://necolas.github.io/normalize.css/)

**“normalize” is useful for to have a starting point equal in all the browsers.**

**Display “inline-block” allows us give an width, height, margin to an element with behavior of a “inline element”, which is not possible do it with a inline element.**

**Each element has a display by default.**

**Don’t use so many selectors, as max, 3.**

**You cant center a imagen with text-align: center, then use flex!**

**It is a bad practice to use “all” in transitions**

**MARGIN**

    margin: 0 auto;

**is equivalent to (new way)**

    margin-inline: auto;

|  |
| --- |
| FONTS |

**No more than 2 or three imported fonts.**

**¿link vs import?**

|  |
| --- |
| Units in CSS (px, rem, em) and why to use “REM” |

**Absolute**

* **Example. centimeters or pixels**
* **The problem is that there are computers with screen at 1080px and smartphones with screens at 4k pixels (this problem is solved with ‘relative’ units)**
* **They don’t scale very well for different screens.**

**Relatives**

* **Take their value based on other element**
* **Examples: em, rem, vh, vw, %**
* **They are adapted better with different resolutions and pixel densities, which make a better option for a big variety of devices.**

[**https://fonts.google.com/knowledge/glossary/em**](https://fonts.google.com/knowledge/glossary/em)

**An em is a unit of measurement, relative to the size of the font; therefore, in a typeface set at a font-size of 16px, one em is 16px. There are better alternatives for to use than “ems”.**

**em is a CSS unit relative to the font size of the parent element, while rem is a CSS unit relative to the font size of an html element.**

**A key reason to use scalable units like em and rem is accessibility. Accessibility enables all users, particularly those with disabilities, to successfully interact with a website. Using fixed units like px to set the values of elements, fonts, and space sizes does not give us this accessibility because fixed units do not scale.**

[**https://blog.logrocket.com/using-em-vs-rem-css/**](https://blog.logrocket.com/using-em-vs-rem-css/)

**REM CALCULATOR 🡪** [**https://nekocalc.com/px-to-rem-converter**](https://nekocalc.com/px-to-rem-converter)

**The default is 16px. So a good hack for the HTML element is font-size: 62.5%**

html {

    box-sizing: border-box;

    font-size: 62.5%; /\* 1 REM => 10px \*/

}

**Why 62.5%?** [**https://www.aleksandrhovhannisyan.com/blog/62-5-percent-font-size-trick/**](https://www.aleksandrhovhannisyan.com/blog/62-5-percent-font-size-trick/)

[**https://stackoverflow.com/questions/28988445/css-62-5-why-do-developers-use-it**](https://stackoverflow.com/questions/28988445/css-62-5-why-do-developers-use-it)

[**https://platzi.com/discusiones/2008-html-css/111887-al-hacer-el-truco-de-emfont-size-625em-625-16px-10px-si-la-etiqueta-html-tendra-ese-valor-10px-de-font-size-entonces/**](https://platzi.com/discusiones/2008-html-css/111887-al-hacer-el-truco-de-emfont-size-625em-625-16px-10px-si-la-etiqueta-html-tendra-ese-valor-10px-de-font-size-entonces/)

**The nesting in CSS is too hard to maintain and reuse (?), the recommendation is max 3 selectors.**

**.selector1 .selector2 .selector3 {**

**}**

|  |
| --- |
| Ways to write code in CSS |

**BEM, Modules, Utilities, SMACSS**

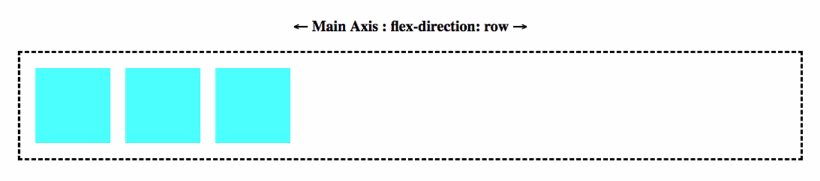
**BEM 🡪 Block element modifier**

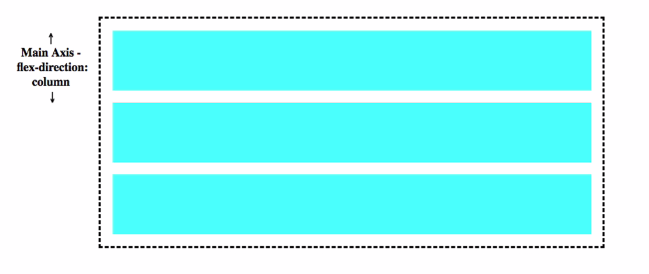
**We can use more than one approach in the same project.**

|  |
| --- |
| Flexbox & css grid |

**Flexbox (display: flex)**

* **Not better or worse than Grid, it is different and both are complemented.**
* **Unidirectional model for create layouts**
* **Before used for positioning: tables and floats**
* **Display: flex 🡪 only affect the first level of child**
* **Axis: how to distribute the elements**
  + **Row or column**
  + **Default: row (one element together to other, from left to right)**
  + **Also known as “main axis”**
  + **Specified for flex-direction**





**Example:**

.nav-principal {

    padding: 20px 0;

    display: flex;

    flex-direction: row;

}

|  |
| --- |
| Flexbox align |

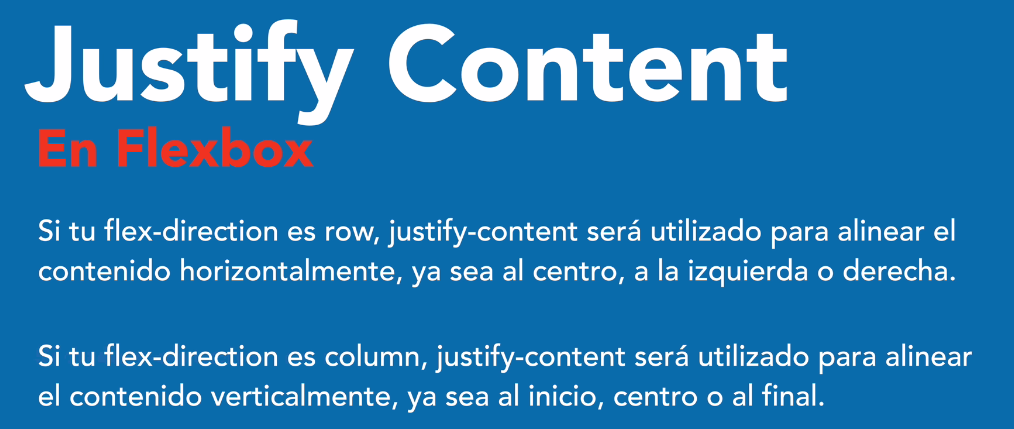
**Key properties**

**flex-direction: row, column, row-reverse, column-reverse**

**justify-content: depending if flex-direction is “row” or “column” is how the content will be aligned. If is “row” this property will be used for align the content HORIZONTALLY.**

**align-items: depending if flex-direction is “row” or “column” is how the content will be aligned. If is “row” this property will be used for align the content VERTICALLY.**

**When you have a flex-direction: row, for align HORIZONTALLY you use “justify-content” and for align VERTICALLY you use “align-items”, and if you have flex-direction: column it is used in inverse order.**





**Example.**



.informacion-nosotros {

    display: flex;

    flex-direction: column;

    justify-content: center;

}

|  |
| --- |
| **Flexbox – RESUME** |

**Position relative & absolute in contrast with flexbox and css grid, any child in the hierarchy (not just the first level) will take the element with position relative for take its positioning.**

**display: flex** 🡪 **Only apply to the first level of child**

**flex-direction: row -> it is the value by default (left to right in the screen)**

**The following code is equivalent**

.d-flex-2 {

    display: flex;

    flex-direction: row;

}

.d-flex-2 {

    display: flex;

}

**justify-content**

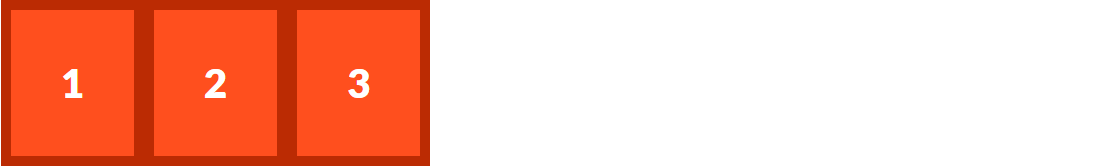
**the behavior is different depending on the value of flex-direction.**

**The default is justify-content: flex-start**

**With flex-direction: row, justify content allow us align HORIZONTALLY**

**flex-direction: row (HORIZONTALLY)**

**justify-content: flex-start**



**justify-content: flex-end**



**justify-content: center**



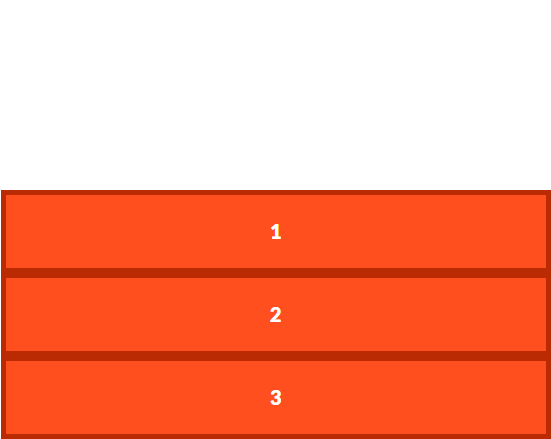
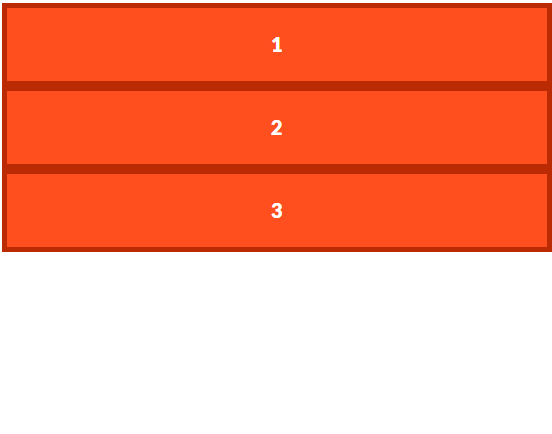
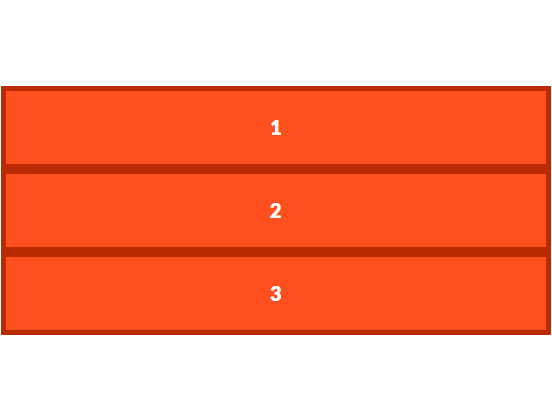
**justify-content: space-between**



**Other values: space-around, space-evenly, space-between**

**flex-direction: column (VERTICALLY)**

**justify-content: flex-start, flex-end, flex-center, etc**

**align-items**

**the default value is align-items: stretch (it is not align-items: flex-start as justify-content). Because this value, the content is “stretched” by default taking up all the available space.**

**the behavior is different depending on the value of flex-direction.**

**flex-direction: row (VERTICALLY) 🡪 the content is aligned from the top to the bottom or vertically.**

**Possible values:** flex-start, stretch, flex-end, center, baseline (doesn’t have the property space-around and space between)

**flex-direction: column (HORIZONTALLY) 🡪 the content is aligned from the left to the right or HORIZONTALLY.**

**justify-content & align items**

**Default values**

    flex-direction: row;

    justify-content: flex-start;

    align-items: stretch;

|  |  |  |
| --- | --- | --- |
| **flex-direction** | **justify-content** | **align-items** |
| **ROW** | **HORIZONTALLY** | **VERTICALLY** |
| **COLUMN** | **VERTICALLY** | **HORIZONTALLY** |

**Practice**

[**https://flexboxfroggy.com/#es**](https://flexboxfroggy.com/#es)

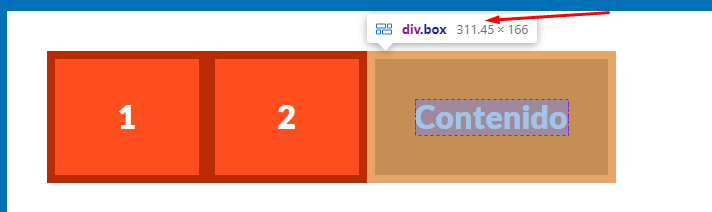
|  |
| --- |
| Flex-basis |

[**https://developer.mozilla.org/en-US/docs/Web/CSS/flex-basis**](https://developer.mozilla.org/en-US/docs/Web/CSS/flex-basis)

**It is a property that is applied in the child (to difference of display, flex-direction, justify-content, align-items).**

**Works similar to width**

**It is the “initial value”. For example, flex basis: 200px, if the content is more wide than that, then the box will grow and will be bigger than 200px.**



**Note: In case both flex-basis (other than auto) and width (or height in case of flex-direction: column) are set for an element, flex-basis has priority.**

|  |
| --- |
| Gap y calc |

**NOTE: the preferred way for create layout it to use CSS Grid, but you can do it with flex too.**

**With flex-basis & gap & calc, I can create layouts.**

**“gap” property for flexbox is a new property and it is well supported in NEW browsers. For better support, instead, I can use the “calc” property.**

.d-flex-9 {

    display: flex;

    gap: 2rem;

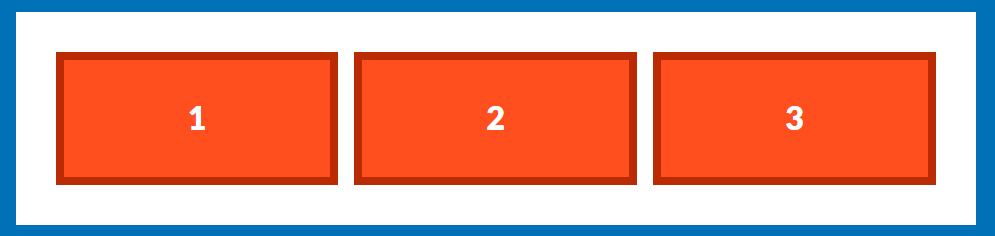
}

.d-flex-9 .box {

    flex-basis: 33.33%;

}

**This create a layout as the follow**



**Using calc (great support) instead gap**

.d-flex-9 {

    display: flex;

    justify-content: space-between;

}

.d-flex-9 .box {

    flex-basis: calc( 33.33% - 1rem );

}

**Other example:**

.d-flex-9 {

    display: flex;

    justify-content: space-between;

}

.d-flex-9 .box:nth-child(1) {

    flex-basis: calc( 50% - 1rem );

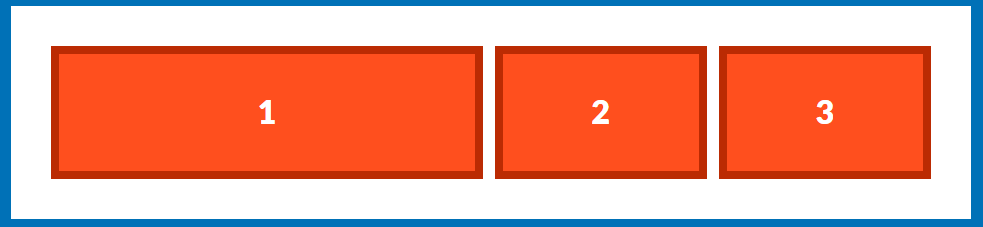
}

.d-flex-9 .box:nth-child(2),

.d-flex-9 .box:nth-child(3) {

    flex-basis: calc( 25% - 1rem );

}



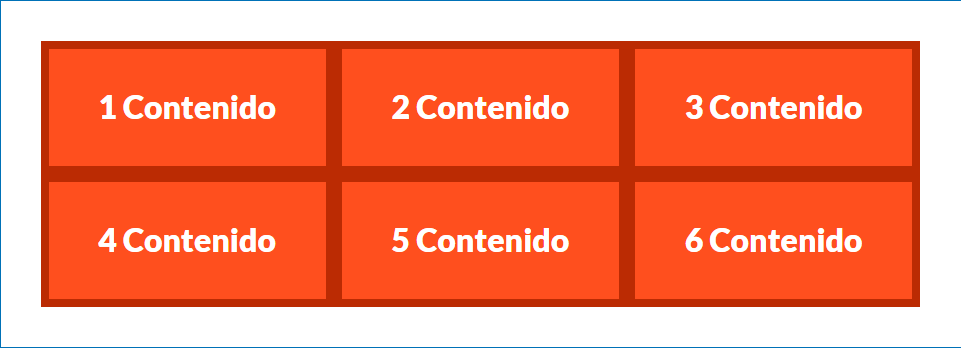
|  |
| --- |
| Flex wrap |

[**https://developer.mozilla.org/en-US/docs/Web/CSS/flex-wrap**](https://developer.mozilla.org/en-US/docs/Web/CSS/flex-wrap)

**flex-wrap: no-wrap (default value)**

**This property is setted to parent level.**

**The flex-wrap CSS property sets whether flex items are forced onto one line or can wrap onto multiple lines. If wrapping is allowed, it sets the direction that lines are stacked.**



|  |
| --- |
| Flex grow |

[**https://developer.mozilla.org/en-US/docs/Web/CSS/flex-grow**](https://developer.mozilla.org/en-US/docs/Web/CSS/flex-grow)

**It is a property that is applied in the child (to difference of display, flex-direction, justify-content, align-items).**

**The flex-grow CSS property sets the flex grow factor of a flex item's main size**

**This property specifies how much of the remaining space in the flex container should be assigned to the item (the flex grow factor).**

**The default value is flex-box: 0 then THIS IS WHY OUR CONTENT DOESN’T GROW WHEN IT IS NOT SPECIFIED.**

**Normally we apply different value to different elements that are child of the same parent.**

**Example**

.d-flex-11 {

    display: flex;

}

.d-flex-11 .box:nth-child(1) {

    flex-grow: 2;

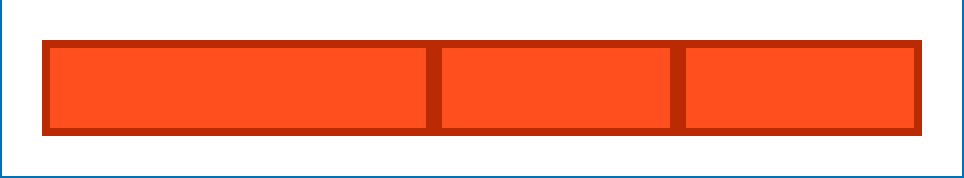
}

.d-flex-11 .box:nth-child(2),

.d-flex-11 .box:nth-child(3) {

    flex-grow: 1;

}



|  |
| --- |
| Flex shrink |

**https://developer.mozilla.org/en-US/docs/Web/CSS/flex-shrink**

**The flex-shrink CSS property sets the flex shrink factor of a flex item. If the size of all flex items is larger than the flex container, items shrink to fit according to flex-shrink.**

**In use, flex-shrink is used alongside the other flex properties flex-grow and flex-basis, and normally defined using the flex shorthand.**

**The default value is flex-shrink: 1**

**So, in the next example, the third box will start to shrink to a velocity more fast than other boxes.**

.d-flex-12 {

    display: flex;

}

.d-flex-12 .box {

    flex-grow: 1;

    flex-basis: 300px;

}

.d-flex-12 .box:nth-child(3) {

    flex-shrink: 2;

}

|  |
| --- |
| Flex (shorthand) |

**flex-basis, flex-shrink, flex-grow in the same flex.**

**flex: flex-grow flex-shrink flex-basis**

**flex: 1 (is valid, where flex-grow: 1 and other setted to to)**

**tip: if you flex-grow is 1 so you can use flex: 1**

**To have a flex-basis = 33.33% is similar to have a flex: 1 (and flex-basis: 0)**

**You can create a layout following**

.d-flex-13 {

    display: flex;

    justify-content: space-between;

}

.d-flex-13 .box {

    flex: 0 0 calc( 33.33% - 1rem );

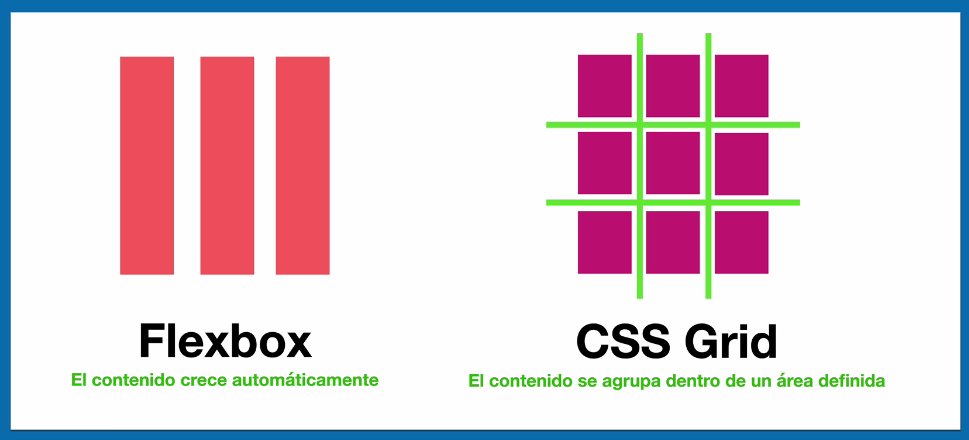
}

|  |
| --- |
| Order (in flex) |

**Search!**

|  |
| --- |
| CSS GRID |

* **CSS Grid or CSS Grid Layout**
* **Flexbox is unidirectional (row or column) but CSS Grid allow you distribute element in both directions.**
* **display: grid 🡪 only affect the first level of child**

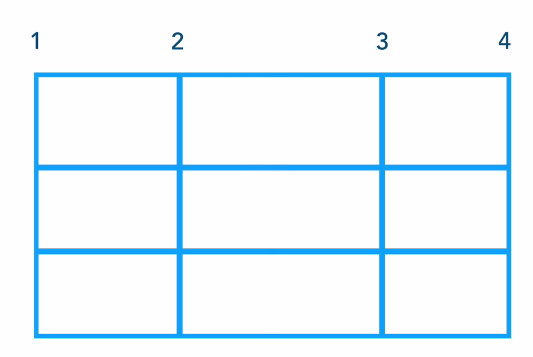


|  |
| --- |
| Terminology |

* **For create column we use grid-template-columns**
* **For create rows we use grid-template-rows**
* **We also create a grid with grid-template-areas**

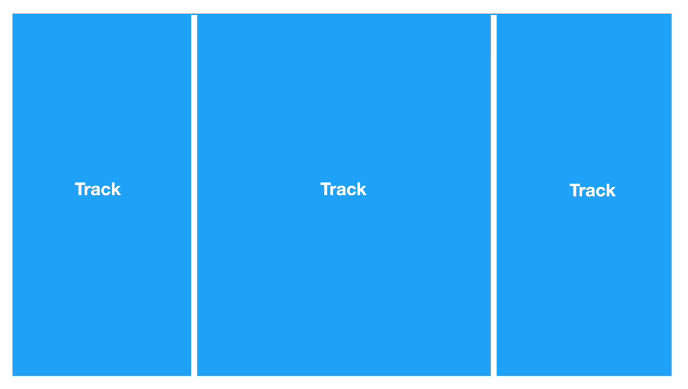
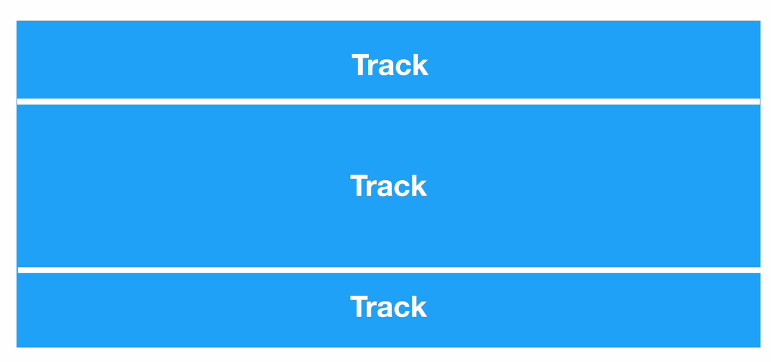
**Grid Lines**

Grid lines are created when you define tracks in the explicit grid using CSS Grid Layout.



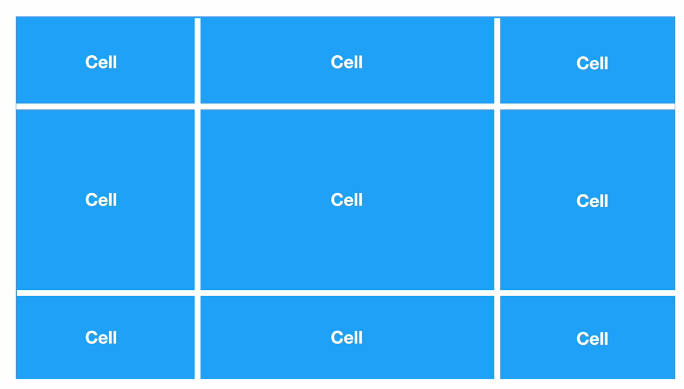
**Grid Track**

A grid track is the space between **two adjacent grid lines**.



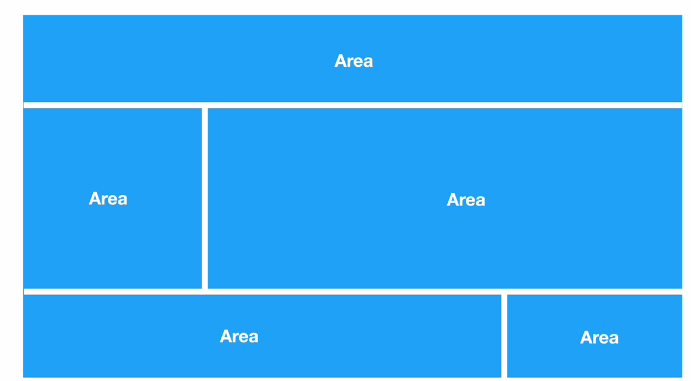
**Grid Cell**

In a CSS Grid Layout, a grid cell is the smallest unit you can have on your CSS grid. It is the space between four intersecting grid lines and conceptually much like a table cell.



**Grid area**

A grid area is one or more grid cells that make up a rectangular area on the grid. Grid areas are created when you place an item using line-based placement or when defining areas using named grid areas.



Examples

.listado-categorias {

    display: grid;

    grid-template-columns: 200px 200px 200px 200px;

    grid-template-rows: 100px 100px;

}

Equivalent to:

.listado-categorias {

    display: grid;

    /\* grid-template-columns: 200px 200px 200px 200px; \*/

    /\* grid-template-columns: repeat(4, 200px); \*/

    grid-template-columns: 1fr 1fr;

    grid-template-rows: 100px 100px;

}

Other example

.listado-categorias {

    display: grid;

    grid-template-columns: repeat(3, 1fr);

    gap: 20px;

}

|  |
| --- |
| CSS GRID RESUME |

**display: grid; 🡪 The same with flex only apply to the first level of child.**

**A row if not setted a height will take the space that required based on its content (padding + margin + border + content).**

**grid-template-columns & positioning (col-start and col-end)**

.grid-2 {

    display: grid;

    grid-template-columns: 300px 300px 300px;

}

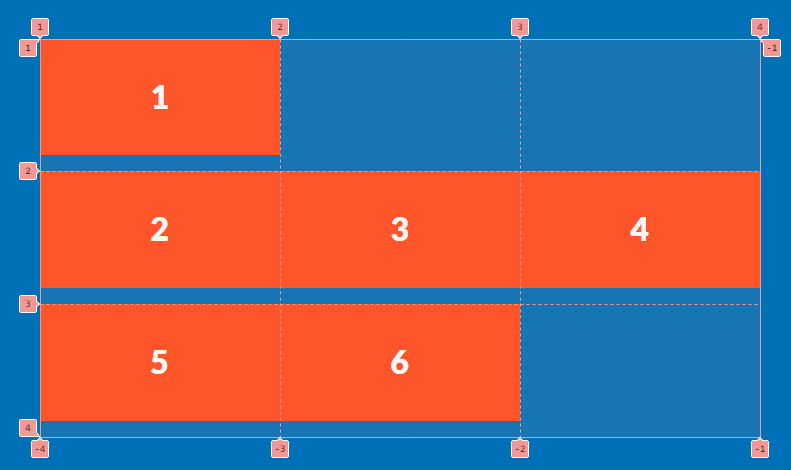
.grid-2 .box:nth-child(2) {

/\*     grid-column-start: 1;

    grid-column-end: 2; \*/

    grid-column: 1 / 2;

}



**grid-template-rows & positioning (row-start and row-end)**

.grid-3 {

    display: grid;

    grid-template-rows: repeat(2, 300px);

    grid-template-columns: repeat(3, 300px);

}

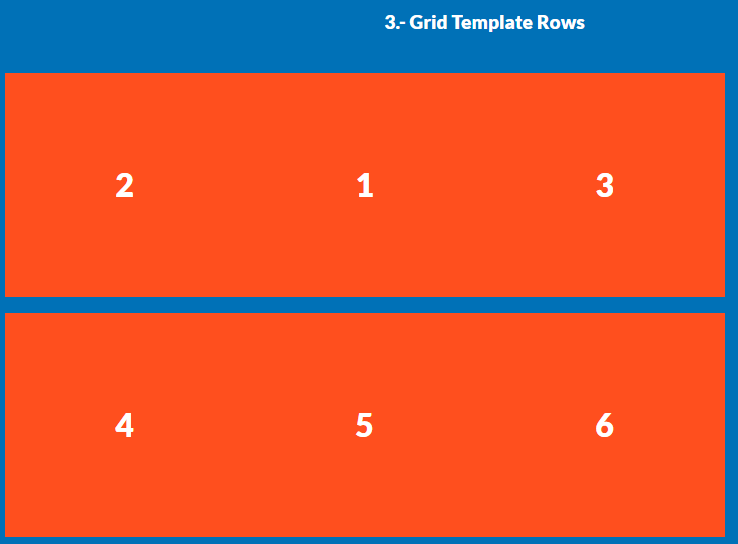
.grid-3 .box:nth-child(2) {

/\*     grid-row-start: 1;

    grid-row-end: 2; \*/

    grid-row: 1 / 2;

}



|  |
| --- |
| CSS GRID: Column span and row span |

.grid-4 {

    display: grid;

    grid-template-rows: repeat(3, 300px);

    grid-template-columns: repeat(3, 300px);

}

.grid-4 .box:nth-child(2) {

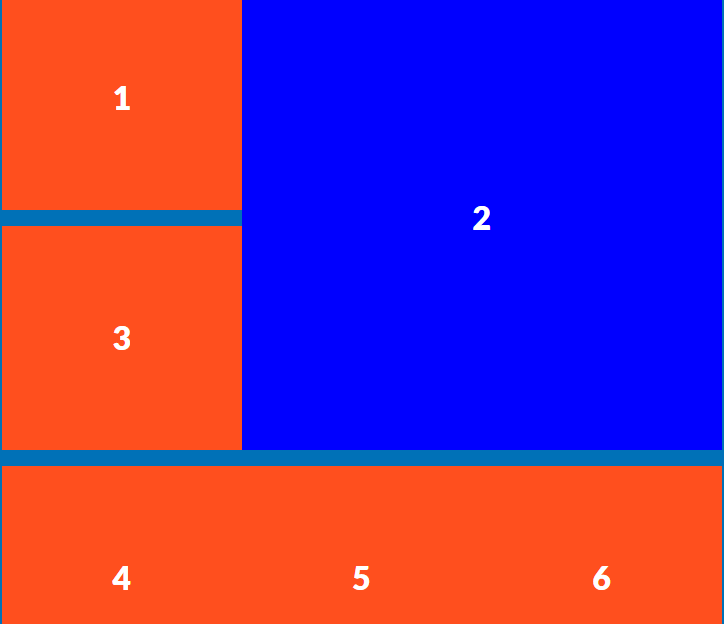
    background-color: blue;

    /\* grid-column: 2 / 4; \*/

    grid-column: 2 / span 2;

    grid-row: 1 / span 2;

}



|  |
| --- |
| Grid shorthand |

grid: grid-template-rows / grid-template-columns

/\*\* 5 \*\*/

.grid-5 {

    display: grid;

/\*     grid-template-rows: repeat(2, 300px);

    grid-template-columns: repeat(3, 300px); \*/

    grid: repeat(2, 300px) / repeat(3, 300px);

}

|  |
| --- |
| Grid Autoflow |

The grid-auto-flow CSS property controls how the auto-placement algorithm works, specifying exactly how auto-placed items get flowed into the grid.

/\*\* 6 \*\*/

.grid-6 {

    display: grid;

    grid: repeat(2, 300px) / repeat(3, 300px);

    grid-auto-flow: dense;

}

.grid-6 .box:nth-child(2) {

    grid-column: 1 / 2;

}

grid-auto-flow: column; /\* or 'row', 'row dense', 'column dense' \*/

we can mix row | column and dense in the same instruction.

|  |
| --- |
| Grid Fr & Grid Repeat |

Fr is a fractional unit and 1fr is for 1 part of the available space

.grid-7 {

    display: grid;

    /\* grid-template-columns: 50% 25% 25%; \*/

    grid-template-columns: 2fr repeat(2, 1fr)

}

An interesting example is:

.grid-7 {

    display: grid;

    grid: repeat(2, 50rem) / repeat(3, 1fr)

}

|  |
| --- |
| Grid Fr & Grid Repeat |

You don’t need to worry about margin and paddings! Just the gap between boxes.

.grid-8 {

    display: grid;

    grid: repeat(2, 50rem) / repeat(3, 1fr);

/\*     column-gap: 4rem;

    row-gap: 4rem; \*/

    gap: 4rem;

}

.grid-8 .box {

    margin: 0;

}

|  |
| --- |
| Grid Areas |

<https://developer.mozilla.org/en-US/docs/Glossary/Grid_Areas>

<https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-areas>

Using “areas” we avoid to use grid line numbers for positioning.

It is a good option to use areas but can be complicated when we use media queries (or maybe it is more easy to use grid line numbers)

.grid-9 {

    display: grid;

/\*     height: 120rem; \*/

    grid-template-columns: repeat(3, 1fr);

    grid-template-areas: "header header header"

                         "nav nav nav"

                         "contenido contenido sidebar"

                         "footer footer footer";

    grid-template-rows: 25rem 10rem 60rem 25rem;

    /\* grid-template-rows: 2.5fr 1fr 6fr 2.5fr; \*/

    gap: 4rem;

}

.grid-9 .box {

    margin: 0px;

}

.grid-9 .box:nth-child(1) {

    grid-area: header;

}

.grid-9 .box:nth-child(2) {

    grid-area: nav;

}

.grid-9 .box:nth-child(3) {

    grid-area: contenido;

}

.grid-9 .box:nth-child(4) {

    grid-area: sidebar;

}

.grid-9 .box:nth-child(5) {

    grid-area: footer;

}

|  |
| --- |
| Grid Templates |

Grid template for grid-template-areas, grid-template-columns, grid-template-rows.

/\*\* 10 \*\*/

.grid-10 {

    display: grid;

    height: 120rem;

    grid-template: "header header header" 2.5fr

                   "nav nav nav" 1fr

                   "contenido contenido sidebar" 6fr

                   "footer footer footer" 2.5fr / 1fr 1fr 1fr;

    gap: 4rem;

}

|  |
| --- |
| Alineación Grid |

We only talk about align VERTICALLY, because horizontally will be defined by the grid that we have.

By default, the box will try to take all the width and height in the cell. **We can to use place-content or align items or align-content.**

We can use align-items

/\*\* 11 \*\*/

.grid-11 {

    display: grid;

    height: 300px;

    grid-template-columns: repeat(6, 1fr);

    align-items: flex-end;

}

|  |
| --- |
| Autofill & Autofit |

<https://developer.mozilla.org/en-US/docs/Web/CSS/repeat>

These properties are used with the “repeat” function. “**auto-fill**” will take all the available space for create columns. **“auto-fit”** will detect how much columns are need whitout you need to specify it. If you need to use fractions, then use minmax

Example:

grid-template-columns: repeat( auto-fill,  200px );

¿How to use with fr?

.grid-12 {

    display: grid;

    grid-template-columns: repeat( auto-fill,  200px );

    /\* When the value (box width) be less 200px will take the max value (1fr) \*/

    grid-template-columns: repeat( auto-fit,  minmax(200px, 1fr) );

}

|  |
| --- |
| Flexbox or CSS GRID |





**Grid for main containers and flex for positioning inside these containers.**

|  |
| --- |
| Object fit |

**The object-fit CSS property sets how the content of a replaced element, such as an <img> or <video>, should be resized to fit its container.**

**Useful with CSS GRID**

[**https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit**](https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit)

.producto:nth-child(2) img,

.producto:nth-child(3) img {

    height: 30rem;

    width: 100%;

    object-fit: cover;

}

|  |
| --- |
| CSS Grid with repeat and minmax |

[**https://www.kuworking.com/css-minmax**](https://www.kuworking.com/css-minmax)

**We can to use repeat and mixmax for do responsive without media queries.**

.contenido-nosotros {

    display: grid;

    grid-template-columns: repeat(auto-fit, minmax( 25rem, 1fr) );

    column-gap: 4rem;

}

**SELECTORES**

|  |
| --- |
| Usual selectors |

**Max 3 levels in our selectors**

**Tag selector**

**class selector**

**id selector (not very reusable). Can select more than one html element, but it is not a valid html! 🡪 more specificity than class.**

**Attribute selector**

/\* That start with... \*/

a[href^="http"] {

    color: red;

}

/\* That finish with... \*/

a[href$=".com"] {

    color: red;

}

**First level of child (>): it is not the first child, it is the first level of child**

.admin > div {

    border: 2px solid red;

}

**Adjacent (next) sibling selector (+): first element “after” of the selector before “+”**

[**https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent\_sibling\_combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent_sibling_combinator)

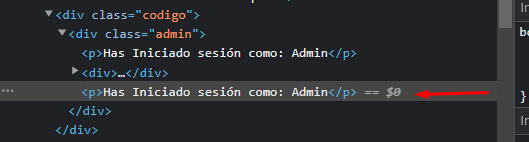
**The adjacent sibling combinator (+) separates two selectors and matches the second element only if it immediately follows the first element, and both are children of the same parent element.**

.admin div + p {

    color: red;

}

**Only select the element marked with red:**



**First & last child of a list**

ul li:first-child {

    border: 2px solid red;

}

ul li:first-of-type {

    border: 2px solid red;

}

ul li:last-child {

    border: 2px solid red;

}

ul li:last-of-type {

    border: 2px solid red;

}

**Difference between :first-child and :first-of-type:** [**https://www.geeksforgeeks.org/difference-between-first-child-and-first-of-type-selector-in-css/**](https://www.geeksforgeeks.org/difference-between-first-child-and-first-of-type-selector-in-css/)

**nth-child**

ul li:nth-child(2) {

    border: 1px solid red;

}

ul li:nth-child(2n + 1) {

    border: 1px solid red;

}

ul li:nth-child(odd) {

    border: 1px solid red;

}

ul li:nth-child(even) {

    border: 1px solid red;

}

ul li:nth-child(4n + 4) {

    border: 1px solid red;

}

**All except… (:not())**

p:not(.texto) {

    color:red;

}

**For multiple selectors combine it in the following way (:)**

p:not(.texto):not(.oferta) {

    color:red;

}

**First character and first line**

.primer-letra::first-letter {

    font-size: 20rem;

}

.primer-linea::first-line {

    font-size: 4rem;

}

**!!!**

        input:not([type="submit"]) {

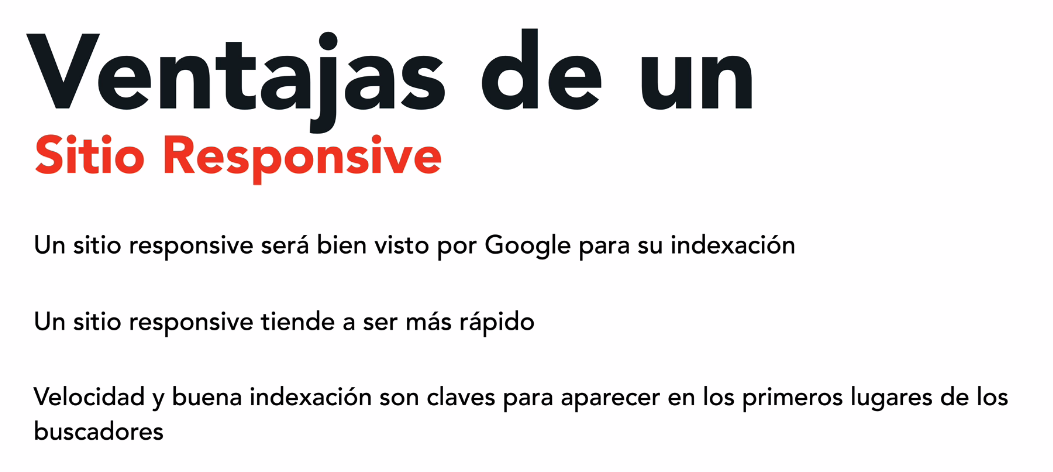
        }

**Responsive Web Design**

|  |
| --- |
| Intro |

[**https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\_layout/Responsive\_Design**](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design)





<https://responsively.app/> 🡪 Good tool for design & test responsive sites

|  |
| --- |
| Media Queries |

**@media**

**The @media CSS at-rule can be used to apply part of a style sheet based on the result of one or more media queries. With it, you specify a media query and a block of CSS to apply to the document if and only if the media query matches the device on which the content is being used.**

[**https://developer.mozilla.org/en-US/docs/Web/CSS/@media**](https://developer.mozilla.org/en-US/docs/Web/CSS/@media)

**We can create media queries based on px or in the pixel density.**

**Why we can to use “rem” in media queries: https://stackoverflow.com/questions/47409585/using-rem-units-in-media-queries-and-as-width**

@media ( max-width: 600px ) {

    .oferta {

        max-width: 30rem;

    }

    .precio {

        font-size: 2rem;

    }

}

**The order in the MQ is important in Desktop First & Mobile First.**

**Some articles, books, recommend put the media queries at the bottom of the css, but we should avoid this mainly by two reasons:**

* **If we are using MobileFirst, when someone visit the site the first time, she or he will be seeing the mobile version while the CSS is downloaded. It is also known as “flash of unstyled content” https://en.wikipedia.org/wiki/Flash\_of\_unstyled\_content**
* **The code for style different parts aren’t near in the code (which is slow for change something)**

**Obviously we will have more media queries, but it is not a performance issue. So the good practice is create the MQ inmediatly after the element that we are styling.**

|  |
| --- |
| Media Queries: Mobile First |

[**https://developer.mozilla.org/en-US/docs/Glossary/Mobile\_First**](https://developer.mozilla.org/en-US/docs/Glossary/Mobile_First)

**It is the recommended.**

**It is the approach which utilize the major part of the CSS Frameworks.**

**Mobile first, a form of progressive enhancement, is a web-development and web-design approach that focuses on prioritizing design and development for mobile screen sizes over design and development for desktop screen sizes.**

**The rationale behind the mobile-first approach is to provide users with good user experiences at all screen sizes—by starting with creating a user experience that works well on small screens, and then building on top of that to further enrich the user experience as the screen size increases. The mobile-first approach contrasts with the older approach of designing for desktop screen sizes first, and then only later adding some support for small screen sizes.**

**When we work with Mobile First always we use “min-width”**

**Note: In this approach in the MQ we go from the lower size (px) to bigger size (px), in min-width**

|  |
| --- |
| Media Queries: Desktop First |

**From the desktop to mobile.**

**In the media queries is the CSS code for devices that aren’t desktop. Outside of the MQ we have all the code for desktop.**

**We use “max-width”**

@media ( max-width: 600px ) {

}

**We shouldn’t mix between max-width and min-width approaches.**

**The order is important!**

@media ( max-width: 600px ) {

    .oferta {

        max-width: 100%;

    }

}

@media ( max-width: 800px ) {

    .oferta {

        max-width: 40rem;

    }

}

**The first MQ *never will be applied* (because cascading)**

**Note: In this approach in the MQ we go from the bigger size (px) to lower size (px), in max-width**

|  |
| --- |
| Media Queries: between two sizes |

@media ( min-width: 600px ) and ( max-width: 800px ) {

    .oferta {

        background-color: yellow;

    }

}

**The following code is valid**

@media ( min-width: 600px ) and ( max-width: 800px ), ( min-width: 1200px ) {

    .oferta {

        background-color: yellow;

    }

}

|  |
| --- |
| Width – device width – density pixel |

[**https://www.sitepoint.com/media-queries-width-vs-device-width/**](https://www.sitepoint.com/media-queries-width-vs-device-width/)

|  |
| --- |
| Snippet for MD for Mobile First |

    "media query": {

        "prefix": "mq",

        "body": [

            "@media (min-width: $1) {\n\t$0\n}"

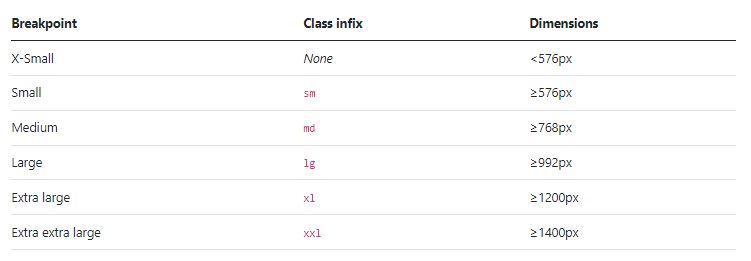
        ]

    }

|  |
| --- |
| Responsive Breakpoints |

[**https://getbootstrap.com/docs/5.0/layout/breakpoints/**](https://getbootstrap.com/docs/5.0/layout/breakpoints/)

[**https://css-tricks.com/snippets/css/media-queries-for-standard-devices/**](https://css-tricks.com/snippets/css/media-queries-for-standard-devices/)



/\* Mobile (landscape)\*/

@media (min-width: 550px) {

}

/\* Tablet (portrait) \*/

@media (min-width: 768px) {

}

/\* Tablet (landscape) \*/

@media (min-width: 1024px) {

}

/\* Laptop \*/

@media (min-width: 1200px) {

}

/\* Big Monitor / TV \*/

@media (min-width: 1600px) {

}

**Normally we use 768, 992, 1200**

|  |
| --- |
| Responsive Containers |

**“min” has a good support.**

.contenedor-responsive {

    background-color: #fff;

/\*     width: 90%;

    max-width: 1000px; \*/

    width: min(90%, 1000px);

    height: 400px;

    margin: 0 auto;

}

|  |
| --- |
| 3 Columns responsive (flexbox & css grid) |

**FLEXBOX**

@media (min-width: 768px) {

    .tres-columnas-flex {

        display: flex;

        gap: 2rem;

    }

    .columna {

        flex-grow: 1; /\* or flex: 1 \*/

    }

}

**CSS GRID**

@media (min-width: 768px) {

    .tres-columnas-grid {

        display: grid;

        grid-template-columns: repeat(3, 1fr);

        column-gap: 2rem;

    }

}

|  |
| --- |
| Responsive Images |

**Normally the images are always in a container. Instead of give a width to an image, we give a width to a container.**

img {

    max-width: 100%;

    display: block;

}

**If I use width instead max-width if the image is very small then it will show pixelated. With max-width the image never will grow more than its original size.**

**Display: block is for remove the margin in the bottom of the image.**

[**https://stackoverflow.com/questions/13111136/why-is-there-a-pesky-little-space-between-img-and-other-elements**](https://stackoverflow.com/questions/13111136/why-is-there-a-pesky-little-space-between-img-and-other-elements)

[**https://stackoverflow.com/questions/5804256/image-inside-div-has-extra-space-below-the-image**](https://stackoverflow.com/questions/5804256/image-inside-div-has-extra-space-below-the-image)

|  |
| --- |
| avif & webp for better performance |

**A good practice is always to put the width and height in the image (for not be waiting to the css load). This practice is recommended by people of google, etc.**

**.avif is really good but still poor browser supported.**

**.webp has better support.**

**.avif is supported by chrome (2022)**

**Now, we can use the “picture” tag and I can achieve full browser support:**

        <picture>

            <source srcset = "img/imagen.avif" type="image/avif" />

            <source srcset = "img/imagen.webp" type="image/webp" />

            <img loading="lazy" src="img/image.jpg" alt="Imagen">

        </picture>

**First, load “avif”, if it is not supported then load “webp”, if it is not supported then load the “jpg”. The image always is placed in the “img tag” (I select this with img in css)**

**loading=”lazy” is optional and it is used for lazy loading which is supported by modern browsers. There is no problem when have it when it is not supported by the browser. Native lazy loading:** [**https://web.dev/native-lazy-loading**](https://web.dev/native-lazy-loading)

[**https://stackoverflow.com/questions/57753240/native-lazy-loading-loading-lazy-not-working-even-with-flags-enabled**](https://stackoverflow.com/questions/57753240/native-lazy-loading-loading-lazy-not-working-even-with-flags-enabled)

|  |
| --- |
| Shorthand for avif & webp |

    "imagenes": {

        "prefix": "im",

        "body": [

            "<picture>",

                "\t<source srcset=\"$1.avif\" type=\"image/avif\">",

                "\t<source srcset=\"$2.webp\" type=\"image/webp\">",

                "\t<img loading=\"lazy\" src=\"$3.jpg\" width=\"500\" height=\"300\" alt=\"$4\">",

            "</picture>"

        ]

    }

|  |
| --- |
| Using SRCSET |

[**https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia\_and\_embedding/Responsive\_images**](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images)

**RWD is not just that the site looks well else it has performance.**

**Responsive images — images that work well on devices with widely differing screen sizes, resolutions, and other such features. This helps to improve performance across different devices.**

**There is multiple syntaxes for achieve the same objective, one of them is:**

        <picture>

            <source sizes="1920w, 1280w, 640w" srcset="img/imagen.avif 1920w,

                    img/imagen-1280.avif 1280w,

                    img/imagen-640.avif 640w" type="image/avif">

            <source sizes="1920w, 1280w, 640w" srcset="img/imagen.webp 1920w,

                    img/imagen-1280.webp 1280w,

                    img/imagen-640.webp 640w" type="image/webp">

            <source sizes="1920w, 1280w, 640w" srcset="img/imagen.jpg 1920w,

                    img/imagen-1280.jpg 1280w,

                    img/imagen-640.jpg 640w" type="image/jpeg">

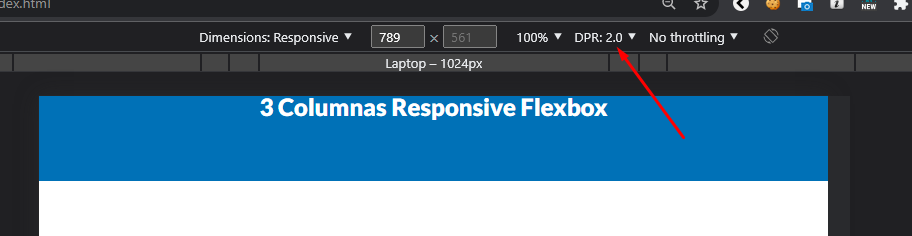
            <img loading="lazy" decoding="async" src="img/imagen.jpg" lazyalt="imagen" width="500" height="300">

        </picture>

decoding="async" 🡪 view more

CSS arguably has better tools for responsive design than HTML, and we'll talk about those in a future CSS module (<https://cloudfour.com/thinks/responsive-images-101-part-8-css-images/>)

**When you are testing take in mind the DPR (device pixel ratio)**



|  |
| --- |
| background & avif & webp |

[**https://caniuse.com/?search=avif**](https://caniuse.com/?search=avif)

[**https://caniuse.com/?search=webp**](https://caniuse.com/?search=webp)

[**https://modernizr.com/**](https://modernizr.com/) **🡪 Respond to your user’s browser features.**

**Today ‘modernizr’ doesn’t support the detection of “avif files”.**

.notavif.notwebp .sobre-tech {

    background-image:

    linear-gradient( to bottom, transparent 50%, var(--primario) 0% ),

    url('../img/imagen-mujer.jpg');

}

.webp .sobre-tech {

    background-image:

    linear-gradient( to bottom, transparent 50%, var(--primario) 0% ),

    url('../img/imagen-mujer.webp');

}

.avif .sobre-tech {

    background-image:

    linear-gradient( to bottom, transparent 50%, var(--primario) 0% ),

    url('../img/imagen-mujer.avif');

}

**CSS STYLE GUIDE**

|  |
| --- |
| Intro |

**A CSS style guide is a set of standards and rules on how to use and write CSS code. It often contains global branding definitions like colors and typography and a set of reusable components for building a more consistent and maintainable project.**

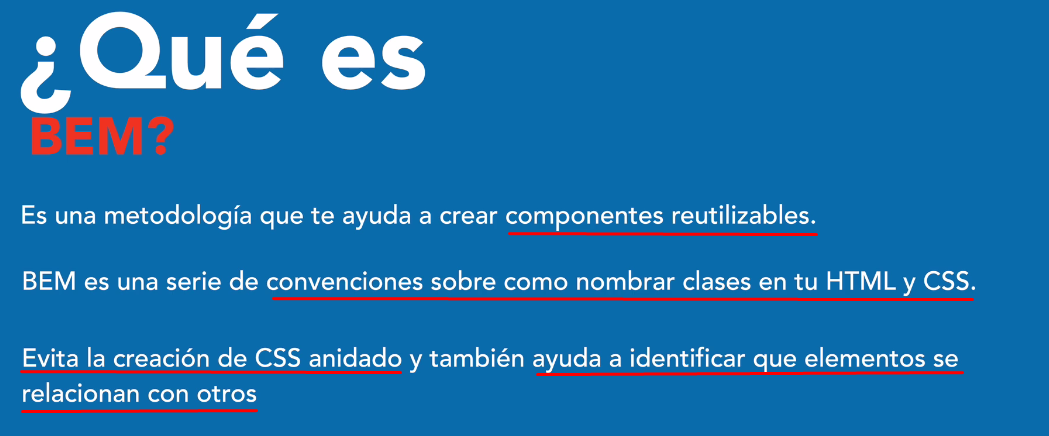
<https://css-tricks.com/css-style-guides/>

<https://blog.logrocket.com/5-things-to-consider-when-creating-your-css-style-guide-7b85fa70039d>

**BEM (block; element; modifier)**

|  |
| --- |
| Intro |

<https://victorroblesweb.es/2022/08/03/que-es-bem-y-como-utilizarlo-en-css/>



With we need to write more HTML.

With BEM the code is more organized that with “modules”

With BEM we avoid the nesting (then avoid the selectors with 3 or 4 classes), because each element has a class that describes this and it is start with the name of the block that is its parent, so with that we avoid nesting.

In BEM the appearance of one block doesn’t depend on other block.

The recommendation is to write class even in elements that we will not give style

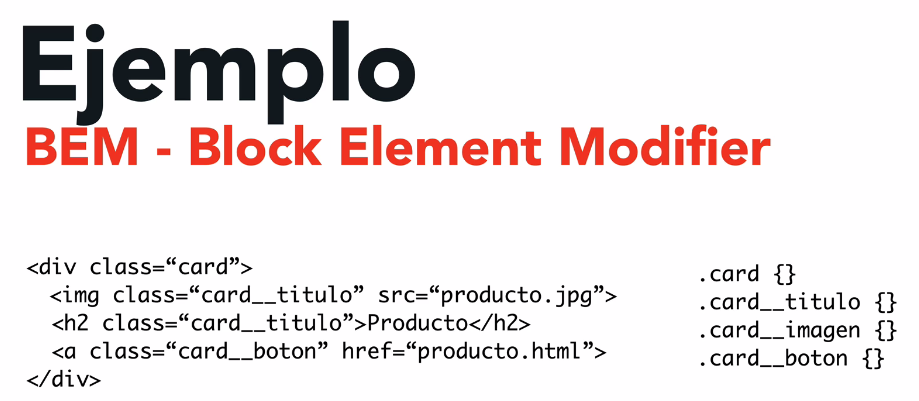
**The name of the block shouldn’t be used in other block.**

**The main idea is that we can reutilize the different blocks.**

**With BEM, We avoid with the next (the nestling), it doesn’t mean that is bad practice to write the code in this way (in many project we utilize this methodology):**



An example with BEM is the follow:





.card {} // It is a block (it is identified because doesn’t have \_\_ or --, and it is unique in the project)

.card\_\_titulo // it is an element

.card\_\_precio--oferta {} // it is a modifier (just change the color respect to the price)

We see that we identify “elements” by the double underscore “\_\_” and we identify “modifier” by double slash “--“.

**The most used are the “elements” and the less used are the “modifiers”**

|  |
| --- |
| Code Example |

                <div class="producto">

                    <img class="producto\_\_imagen" src="img/producto5.jpg" alt="imagen producto">

                    <div class="producto\_\_contenido">

                        <h3 class="producto\_\_nombre">Producto 5</h3>

                        <p class="producto\_\_descripcion">Lorem ipsum dolor sit, amet consectetur adipisicing elit. Commodi repellendus tempore delectus accusantium, minima ea amet ipsam, et voluptatibus mollitia error excepturi, similique dolorum placeat porro ducimus ullam quae libero.</p>

                        <div class="producto\_\_precios">

                            <p class="producto\_\_precio producto\_\_precio--oferta">$99</p>

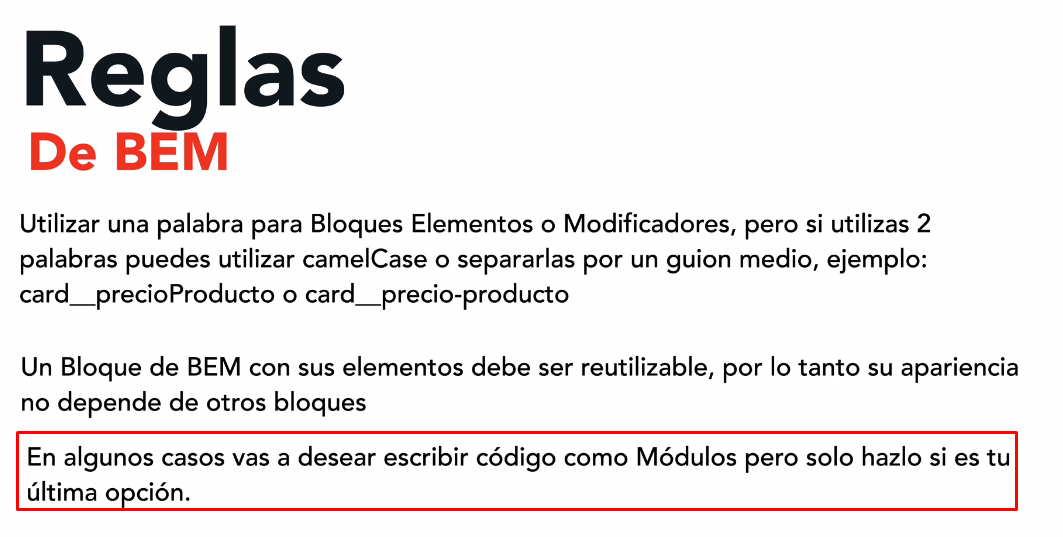
                            <p class="producto\_\_precio">$199</p>

                        </div>

                        <button class="producto\_\_boton" type="button">Añadir al carrito</button>

                    </div>

                </div><!--.producto-->



For share common styles it is good to use the next approach (for example for a container), attribute selector.

[class$="\_\_contenedor"] {

    max-width: 120rem;

    margin: 0 auto;

    width: 90%;

}

With this don’t take css code from other blocks and I am not using utilities in the blocks.

**GULP & SASS**

|  |
| --- |
| SASS |

**SASS 🡪 Syntactically awesome stylesheet.**

**It is considered a standard in the industry.**

**Compatible with many frameworks like Bootstrap.**

**Maybe considered a programming language because it has variables, functions and many more.**

**SASS is compiled, it is not supported natively by the browser. We can to use different options like webpack or gulp.**

Advantages

* Better order and structure in the code.
* Features that doesn’t exists in CSS
* Can be compiled to compatible CSS (for example with PostCSS), it means that I can write ultimate generation CSS code and still be compatible with old browsers.
* Support CSS code (CSS superset)
* Less code because use a syntax called “nesting”

Disadvantages

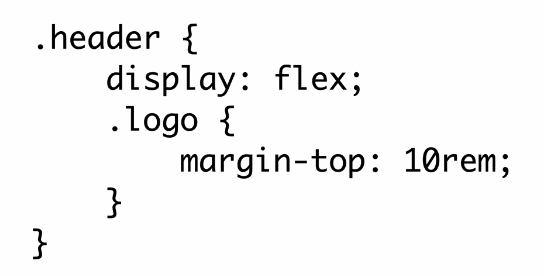
* New syntax to learn.
* It is necessary compile with a special tool because it is not native for the browser (the browser doesn’t understand sass).
* Nesting can cause problems

**SASS Files extensions**

The sass files can have two extensions: .scss and .sass (less used)

Can be imported and used both in the same project (including .css)

The syntax for .scss and .sass is different.



The files that start with “\_”         input:not([type="submit"]) {

        }

and not will be compiled by itself.

In old versions of sass we include a file with

@import 'header/header';

In new versions with use “use”

@use 'header/header';

@import @use @forward

<https://stackoverflow.com/questions/73353442/what-are-the-differences-between-use-and-forward-in-sass>

Both @use and @forward are the alternatives provided by Dart sass for importing other stylesheets. When @import exposes the members such as variables, mixins, and functions globally accessible, @use loads them with the namespace. Hence, sass encourages using @use over @import.

Coming to the other, the @forward is paired with @use. For example, when a group of stylesheets have to be reused across multiple stylesheets, we can merge all of them into a single entry point using @forward and make use of @use to consume in the other stylesheets.

& 🡪 refer to the same element

.header {

    padding: 5rem 0;

    &::before {

        content: '';

    }

    &:hover {

        background-color: red;

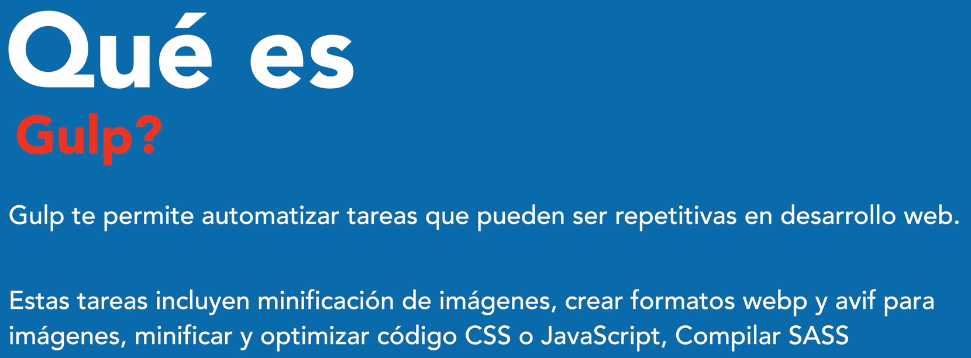
    }

    .contenido-header {

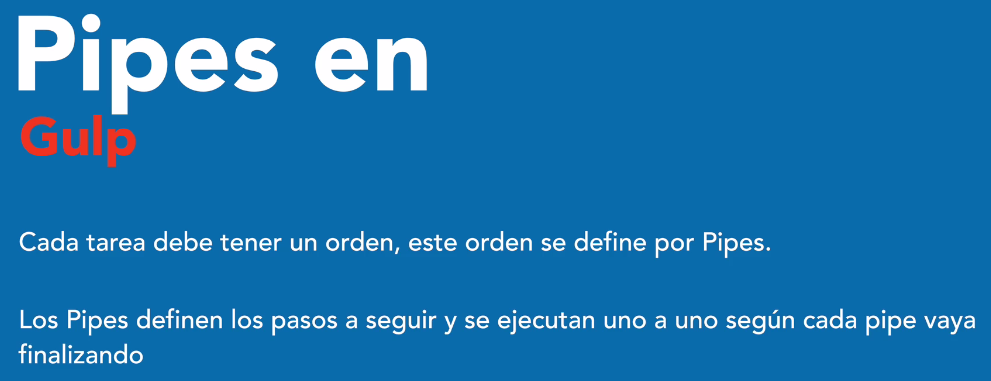
    }

}

|  |
| --- |
| Gulp |



[**https://gulpjs.com/docs/en/getting-started/quick-start/**](https://gulpjs.com/docs/en/getting-started/quick-start/)



**gulpfile.js**

function task ( done ) {

    console.log("Hi!")

    done();

}

exports.firstTask = task;

from the console:

**gulp firstTask**

Some util extensions: Live Sass Compiler

Dev Dependencies:

* gulp
* sass
* gulp-postcss
* gulp-sass
* autoprefixer
* postcss
* gulp-imagemin
* gulp-webp
* gulp-avif
* gulp-sourcemaps
* cssnano

We can set in browsersList in package.json the destionation browsers

  "browserslist": [

    "IE 10"

  ]

We can to use new css features even in older browsers

  "browserslist": [

    "last 1 version",

    ">1%"

  ]

|  |
| --- |
| Mixins |

**It is recommendable group all the mixins in one file.**

[**https://codigofacilito.com/articulos/mixins-en-sass**](https://codigofacilito.com/articulos/mixins-en-sass)

    @include m.telefono {

        padding: 2rem 0;

    }

    @include m.tablet {

        padding: 3rem;

    }

    @include m.desktop {

        padding: 2rem;

    }

|  |
| --- |
| Sourcemaps |

[**https://css-tricks.com/should-i-use-source-maps-in-production/**](https://css-tricks.com/should-i-use-source-maps-in-production/)

|  |
| --- |
| SASS & BEM |

https://medium.com/swlh/sass-bem-nesting-variables-970403f42dd6

<https://getwaves.io>

|  |
| --- |
| SCROLL (NEW CSS) |

scroll-snap-type: x mandatory;

scroll-snap-align: center;

|  |
| --- |
| aspect-ratio |

    &\_\_imagen {

        // Old solutions

        /\* padding-top: 50%;

        height: 40rem; \*/

        aspect-ratio: 1 / 1;

    }

|  |
| --- |
| Flex and position relative |

|  |
| --- |
| Transitions vs animations in CSS |

@media (min-width: 992px) {

    .header\_\_guitarra {

        display: block;

        position: absolute;

        right: 0;

        bottom: 0;

        animation-name: showImage;

        animation-duration: 1s;

        animation-delay: 0s;

        animation-timing-function: ease-in-out;

        animation-iteration-count: 1;

    }

}

@keyframes showImage {

    0% {

        opacity: 0;

        transform: translateX(-10rem)

    }

    50% {

        opacity: 0;

    }

    100% {

        opacity: 1;

        transform: translateX(0)

    }

}