https://css-tricks.com/snippets/css/complete-guide-grid/

https://gridbyexample.com/examples/

**CSS Grid layout**

> a 2-dimensional system, can handle both columns and rows

> grid layout, parent element = grid container : children = grid items

**Grid Container**

The element on which display: grid is applied. It's the direct parent of all the grid items. In this example container is the grid container.

<div class="container">

<div class="item item-1"></div>

<div class="item item-2"></div>

<div class="item item-3"></div>

</div>

**Grid Item**

The children (e.g. direct descendants) of the grid container. Here the item elements are grid items, but sub-item isn't.

<div class="container">

<div class="item"></div>

<div class="item">

<p class="sub-item"></p>

</div>

<div class="item"></div>

</div>

Others: Grid Line, Grid Track, Grid Cell, Grid Area

Properties for the Grid Container

**display : grid** | inline-grid | subgrid /\*define grid container\*/

grid-template-columns : track-size | line-name /\*define columns and rows

grid-template-rows :

/\*--examples--grid-container--\*/

.container {

display: grid;

grid-template-columns: 40px 50px auto 50px 40px;

grid-template-rows: 25% 100px auto;

}

.container {

grid-template-columns: 1fr 1fr 1fr;

}

**grid-template-areas** : grid-area-name | . | none

\*define grid template by referencing grid-area property. repeating the name of a grid area causes the content to span those cells. A period signifies an empty cell. The syntax itself provides a visualization of the structure of the grid.

/\*--example--grid-template-areas--\*/

.item-a {

grid-area: header;

}

.item-b {

grid-area: main;

}

.item-c {

grid-area: sidebar;

}

.item-d {

grid-area: footer;

}

.container {

grid-template-columns: 50px 50px 50px 50px;

grid-template-rows: auto;

grid-template-areas:

"header header header header"

"main main . sidebar"

"footer footer footer footer";

}

**grid-template** : none | subgrid | <grid-template-rows> / <grid-template-columns

\*A shorthand for setting grid-template-rows, grid-template-columns, and grid-template-areas in a single declaration\*/

grid-column-gap : line-size – a length value

grid-row-gap

\*specifies the size of the grid lines.

.container {

grid-template-columns: 100px 50px 100px;

grid-template-rows: 80px auto 80px;

grid-column-gap: 10px;

grid-row-gap: 15px;

}

**grid-gap :** <grid-row-gap> <grid-column-gap>

\*a shorthand for grid-row-gap and grid-column-gap

**justify-items :** start | end | center | stretch (horizontal justification)

\*aligns the content inside a grid item along the row axis (as opposed to align-items which aligns the column axis).

\*can be set on individual grid items via the justify-self property (horizontal justification inside a grid item.

.container {

justify-items: center | stretch;

}

**align-items :** start | end | center | stretch (vertical justification)

\*aligns the content inside a grid item along the column axis

\*can be set on individual grid items via the align-self property (vertical justification inside a grid item.

.container {

justify-items: center | stretch;

}

**justify-content**

\*sometimes the total size of your grid might be less than the size of its grid container. This could happen if all of your grid items are sized with non-flexible units like px. In this case you can set the alignment of the grid within the grid container. This property aligns the grid along the row axis (horizontal justification).

Values:

start – aligns the grid to the left end of the grid container.

end – aligns the grid to the right end of the grid container.

center – aligns the grid to the center of the grid container.

stretch – resizes the grid items to allow the grid to fill the full width of the grid container

space-around – places an even amount of space between each grid item, with half-sized spaces on the far ends

space-between – places an even amount of space between each grid item, with no space at the far ends

space-evenly – places an even amount of space between each grid item, including the far ends

**align-content** : start | end | center | stretch | space-around | space-between | space-evenly

\*this property aligns the grid along the column axis (vertical justification).

**grid-auto-columns :** <track-size>

**grid-auto-rows**

\*specifies the size of any auto-generated grid tracks (aka implicit grid tracks). Implicit grid tracks get created when you explicitly position rows or columns (via grid-template-rows / grid-template-columns) that are out of range of the defined grid.

This creates a 2x2 grid:

.container {

grid-template-columns : 60px 60px;

grid-template-rows: 90px 90px;

}

But now imagine you use grid-column and grid-row to position your grid items like this:

.item-a {

grid-column: 1/ 2;

grid-row: 2/3;

}

.item-b {

grid-column: 5/ 6;

grid-row: 2/3;

}

We told .item-b to start on column line 5 and end at column line 6, but we never defined a column line 5 or 6.

Because we referenced lines that don’t exist, implicit tracks with widths of 0 are created to fill in the gaps. We can use **grid-auto-columns** and **grid-auto-rows** to specify the widths of these implicit tracks:

.container {

grid-auto-columns: 60px;

}

**grid-auto-flow :** row | column | row dense | column dense

\*if you have grid items that you don’t explicitly place on the grid, the auto-placement algorithm kicks in to automatically place the items. This property controls how the auto-placement algorithm works.

Values:

row – tells the auto-placement algorithm to fill in each row in turn, adding new rows as necessary

column – tells the auto-placement algorithm to fill in each column in turn, adding new rows as necessary

dense – tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later

/\*example\*/

Consider this HTML:

<section class="container">

<div class="item-a">item-a</div>

<div class="item-b">item-b</div>

<div class="item-c">item-c</div>

<div class="item-d">item-d</div>

<div class="item-e">item-e</div>

</section>

You define a grid with five columns and two rows, and set grid-auto-flow to row (which is also the default):

.container {

display: grid;

grid-template-columns: 60px 60px 60px 60px 60px;

grid-template-rows: 30px 30px;

grid-auto-flow: row;

}

When placing the items on the grid, you only specify spots for two of them:

.item-a {

grid-column: 1;

grid-row: 1 / 3;

}

.item-e {

grid-column: 5;

grid-row: 1 / 3;

}

Because we set grid-auto-flow to row, our grid will look like this. Notice how the three items we didn't place (item-b, item-c and item-d) flow across the available rows:

If we instead set grid-auto-flow to column, item-b, item-c and item-d flow down the columns:

.container {

display: grid;

grid-template-columns: 60px 60px 60px 60px 60px;

grid-template-rows: 30px 30px;

grid-auto-flow: column;

}

**grid**

\*A shorthand for setting all of the following properties in a single declaration: grid-template-rows, grid-template-columns, grid-template-areas, grid-auto-rows, grid-auto-columns, and grid-auto-flow. It also sets grid-column-gap and grid-row-gap to their initial values, even though they can't be explicitly set by this property.

Values:

none - sets all sub-properties to their initial values

<grid-template-rows> / <grid-template-columns> - sets grid-template-rows and grid-template-columns to the specified values, respectively, and all other sub-properties to their initial values

<grid-auto-flow> [<grid-auto-rows> [ / <grid-auto-columns>] ] - accepts all the same values as grid-auto-flow, grid-auto-rows and grid-auto-columns, respectively. If grid-auto-columns is omitted, it is set to the value specified for grid-auto-rows. If both are omitted, they are set to their initial values

**Properties for the Grid Items**

grid-column-start

grid-column-end

grid-row-start

grid-row-end

Determines a grid item's location within the grid by referring to specific grid lines. grid-column-start/grid-row-start is the line where the item begins, and grid-column-end/grid-row-end is the line where the item ends.

Values:

<line> - can be a number to refer to a numbered grid line, or a name to refer to a named grid line span <number> - the item will span across the provided number of grid tracks span

<name> - the item will span across until it hits the next line with the provided name

auto - indicates auto-placement, an automatic span, or a default span of one

Examples:

.item-a {

grid-column-start: 2;

grid-column-end: five;

grid-row-start: row1-start

grid-row-end: 3

}

.item-b {

grid-column-start: 1;

grid-column-end: span col4-start;

grid-row-start: 2

grid-row-end: span 2

}

If no grid-column-end/grid-row-end is declared, the item will span 1 track by default.

Items can overlap each other. You can use z-index to control their stacking order.

**grid-column**

**grid-row**

**\***Shorthand for grid-column-start + grid-column-end, and grid-row-start + grid-row-end, respectively.

Values:

<start-line> / <end-line> - each one accepts all the same values as the longhand version, including span

.item-c {

grid-column: 3 / span 2;

grid-row: third-line / 4;

}

If no end line value is declared, the item will span 1 track by default.

**grid-area**

**\***Gives an item a name so that it can be referenced by a template created with the grid-template-areas property. Alternatively, this property can be used as an even shorter shorthand for grid-row-start + grid-column-start + grid-row-end + grid-column-end.

Values:

<name> - a name of your choosing

<row-start> / <column-start> / <row-end> / <column-end> - can be numbers or named lines

**justify-self**

\*aligns the content inside a grid item along the row axis (as opposed to align-self which aligns along the column axis). This value applies to the content inside a single grid item.

To set alignment for all the items in a grid, this behavior can also be set on the grid container via the justify-items property.

Values:

start - aligns the content to the left end of the grid area

end - aligns the content to the right end of the grid area

center - aligns the content in the center of the grid area

stretch - fills the whole width of the grid area (this is the default)

**align-self**

\*Aligns the content inside a grid item along the column axis (as opposed to justify-self which aligns along the row axis). This value applies to the content inside a single grid item.

To align all the items in a grid, this behavior can also be set on the grid container via the align-items property.

Values:

start - aligns the content to the top of the grid area

end - aligns the content to the bottom of the grid area

center - aligns the content in the center of the grid area

stretch - fills the whole height of the grid area (this is the default)