CSS GRID

MUHAMMAD JUNAID

0304-1659294 | junaidfazal08@gmail.com

AL-JUNAID INSTITUTE OF GROUP

Properties for the Parent (Grid Container)

Display

Defines the element as a grid container and establishes a new grid formatting context for its contents.

Values:

- **grid** – generates a block-level grid
- **inline-grid** – generates an inline-level grid

```
.container {
  display: grid | inline-grid;
}
```

grid-template-columns
grid-template-rows

Defines the columns and rows of the grid with a space-separated list of values. The values represent the track size, and the space between them represents the grid line.

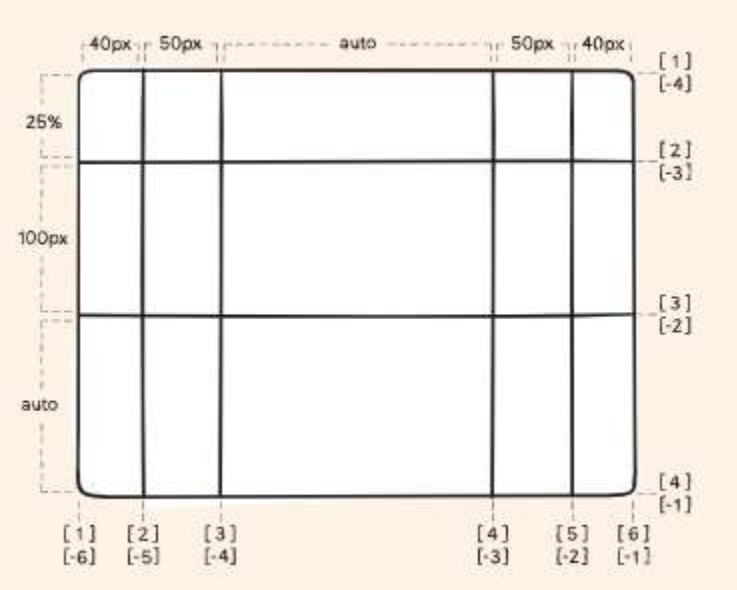
Values:

- **<track-size>** – can be a length, a percentage, or a fraction of the free space in the grid (using the fr unit)
- **<line-name>** – an arbitrary name of your choosing

```
.container {
  grid-template-columns: ...  ...;
  /* e.g.
```

```
    1fr 1fr
    minmax(10px, 1fr) 3fr
    repeat(5, 1fr)
    50px auto 100px 1fr
  */
  grid-template-rows: ... ...;
  /* e.g.
    min-content 1fr min-content
    100px 1fr max-content
  */
}
```

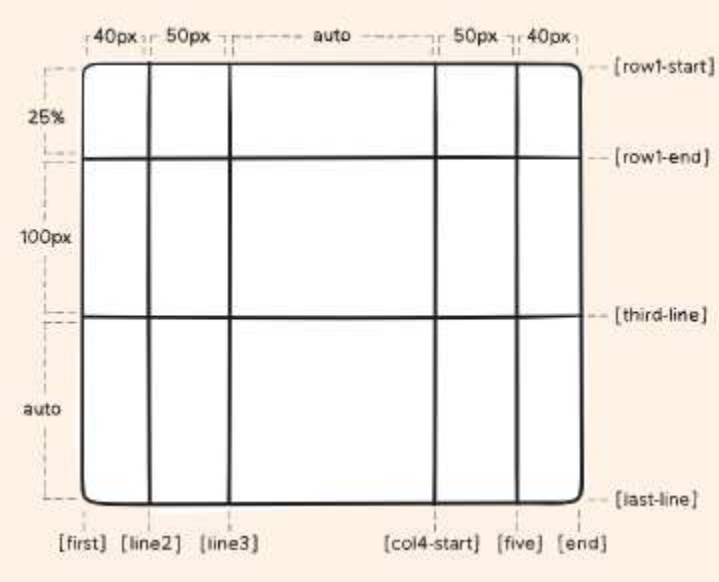
Grid lines are automatically assigned positive numbers from these assignments (-1 being an alternate for the very last row).



But you can choose to explicitly name the lines. Note the bracket syntax for the line names:

```
.container {
```

```
  grid-template-columns: [first] 40px [line2] 50px [line3]
auto [col4-start] 50px [five] 40px [end];
  grid-template-rows: [row1-start] 25% [row1-end] 100px
[third-line] auto [last-line];
}
```



Note that a line can have more than one name. For example, here the second line will have two names: row1-end and row2-start:

```
.container {
  grid-template-rows: [row1-start] 25% [row1-end
row2-start] 25% [row2-end];
}
```

If your definition contains repeating parts, you can use the repeat() notation to streamline things:

```
.container {
  grid-template-columns: repeat(3, 20px [col-start]);
```

```
}
```

Which is equivalent to this:

```
.container {
  grid-template-columns: 20px [col-start] 20px [col-start] 20px
[col-start];
}
```

If multiple lines share the same name, they can be referenced by their line name and count.

```
.item {
  grid-column-start: col-start 2;
}
```

The fr unit allows you to set the size of a track as a fraction of the free space of the grid container. For example, this will set each item to one third the width of the grid container:

```
.container {
  grid-template-columns: 1fr 1fr 1fr;
}
```

The free space is calculated after any non-flexible items. In this example the total amount of free space available to the fr units doesn't include the 50px:

```
.container {
  grid-template-columns: 1fr 50px 1fr 1fr;
}
```

grid-template-areas

AL-JUNAID INSTITUTE OF GROUP

Defines a grid template by referencing the names of the grid areas which are specified with the 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.

Values:

- **<grid-area-name>** – the name of a grid area specified with grid-area
- **.** – a period signifies an empty grid cell
- **none** – no grid areas are defined

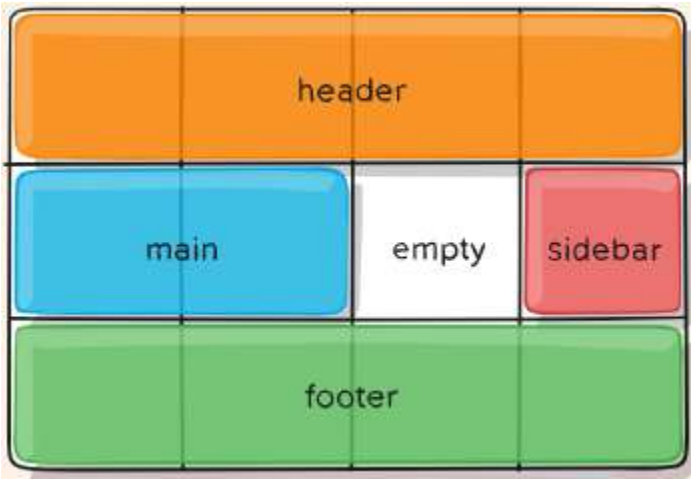
```
.container {
  grid-template-areas:
    "<grid-area-name> | . | none | ..."
    "...";
}
```

Example:

```
.item-a {
  grid-area: header;
}
.item-b {
  grid-area: main;
}
.item-c {
  grid-area: sidebar;
}
.item-d {
  grid-area: footer;
}
.container {
```

```
  display: grid;
  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";
}
```

That'll create a grid that's four columns wide by three rows tall. The entire top row will be composed of the **header** area. The middle row will be composed of two **main** areas, one empty cell, and one **sidebar** area. The last row is all **footer**.



Each row in your declaration needs to have the same number of cells.

You can use any number of adjacent periods to declare a single empty cell. As long as the periods have no spaces between them they represent a single cell.

Notice that you're not naming lines with this syntax, just areas. When you use this syntax the lines on either end of the areas are actually getting named automatically. If the name of your grid area is **foo**, the name of the area's starting row line and starting column line will be **foo-start**, and the name of

its last row line and last column line will be **foo-end**. This means that some lines might have multiple names, such as the far left line in the above example, which will have three names: header-start, main-start, and footer-start.

grid-template

Values:

- **none** – sets all three properties to their initial values
- **<grid-template-rows> / <grid-template-columns>** – sets grid-template-columns and grid-template-rows to the specified values, respectively, and sets grid-template-areas to none

```
.container {
  grid-template: none | <grid-template-rows> / <grid-template-columns>;
}
```

It also accepts a more complex but quite handy syntax for specifying all three. Here's an example:

```
.container {
  grid-template:
    [row1-start] "header header header" 25px [row1-end]
    [row2-start] "footer footer footer" 25px [row2-end]
    / auto 50px auto;
}
```

That's equivalent to this:

AL-JUNAID INSTITUTE OF GROUP

```
.container {
  grid-template-rows: [row1-start] 25px [row1-end row2-start] 25px [row2-end];
  grid-template-columns: auto 50px auto;
  grid-template-areas:
    "header header header"
    "footer footer footer";
}
```

Since grid-template doesn't reset the implicit grid properties (grid-auto-columns, grid-auto-rows, and grid-auto-flow), which is probably what you want to do in most cases, it's recommended to use the grid property instead of grid-template.

column-gap
row-gap
grid-column-gap
grid-row-gap

Specifies the size of the grid lines. You can think of it like setting the width of the gutters between the columns/rows.

Values:

- **<line-size>** – a length value

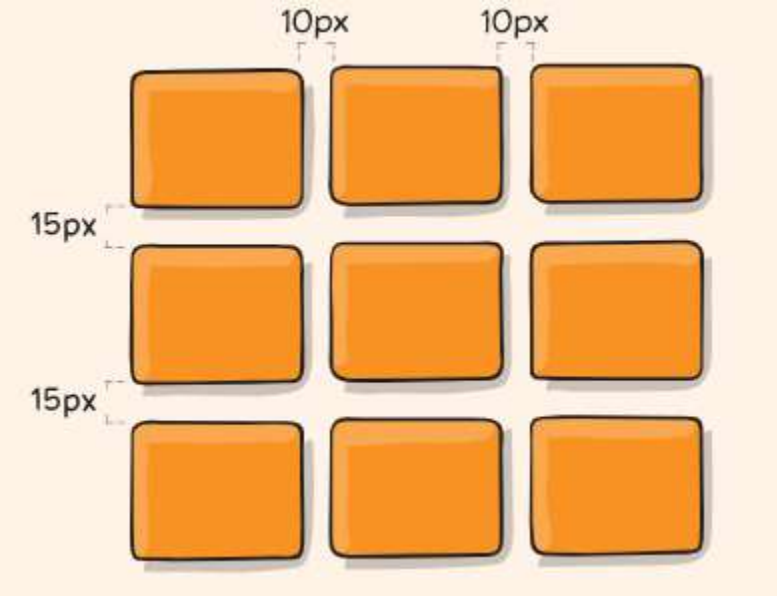
```
.container {
  /* standard */
  column-gap: <line-size>;
  row-gap: <line-size>;

  /* old */
```

```
  grid-column-gap: <line-size>;
  grid-row-gap: <line-size>;
}
```

Example:

```
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  column-gap: 10px;
  row-gap: 15px;
}
```



The gutters are only created between the columns/rows, not on the outer edges.

Note: The grid- prefix will be removed and grid-column-gap and grid-row-gap renamed to column-gap and row-gap.

gap
grid-gap

AL-JUNAID INSTITUTE OF GROUP

A shorthand for row-gap and column-gap

Values:

- **<grid-row-gap> <grid-column-gap>** – length values

```
.container {
  /* standard */
  gap: <grid-row-gap> <grid-column-gap>;

  /* old */
  grid-gap: <grid-row-gap> <grid-column-gap>;
}
```

Example:

```
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  gap: 15px 10px;
}
```

If no row-gap is specified, it's set to the same value as column-gap

Note: The grid- prefix is deprecated (but who knows, may never actually be removed from browsers). Essentially grid-gap renamed to gap.

Justify-items

AL-JUNAID INSTITUTE OF GROUP

Aligns grid items along the inline (row) axis (as opposed to align-items which aligns along the block (column) axis). This value applies to all grid items inside the container.

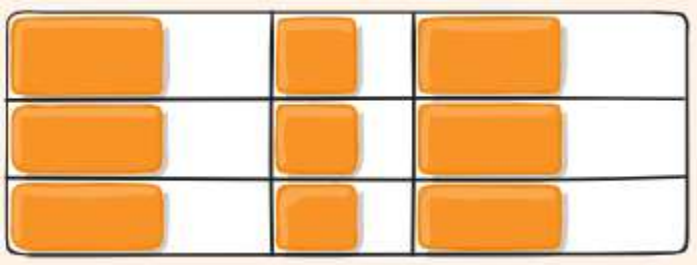
Values:

- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell
- **stretch** – fills the whole width of the cell (this is the default)

```
.container {
  justify-items: start | end | center | stretch;
}
```

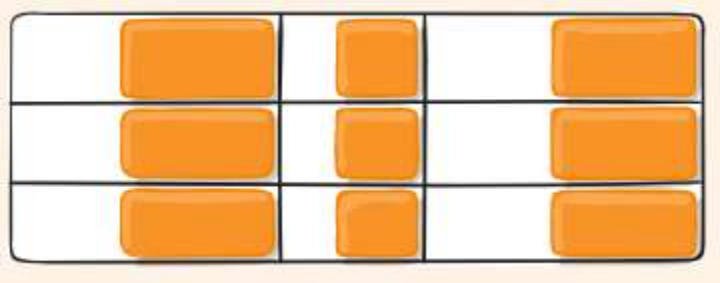
Examples:

```
.container {
  justify-items: start;
}
```

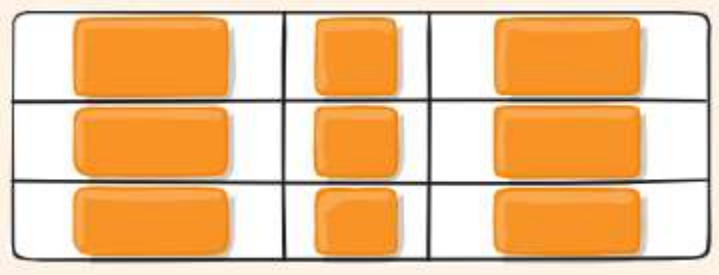


```
.container {
  justify-items: end;
}
```

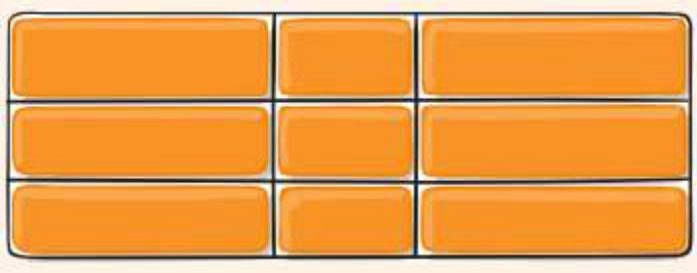
AL-JUNAID INSTITUTE OF GROUP



```
.container {
  justify-items: center;
}
```



```
.container {
  justify-items: stretch;
}
```



This behavior can also be set on individual grid items via the justify-self property.

align-items

Aligns grid items along the block (column) axis (as opposed to justify-items which aligns along the inline (row) axis). This value applies to all grid items inside the container.

AL-JUNAID INSTITUTE OF GROUP

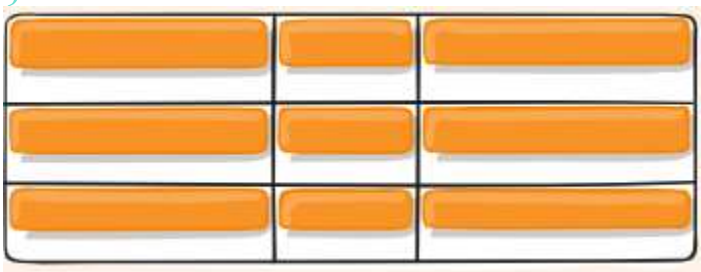
Values:

- **stretch** – fills the whole height of the cell (this is the default)
- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell
- **baseline** – align items along text baseline. There are modifiers to baseline — first baseline and last baseline which will use the baseline from the first or last line in the case of multi-line text.

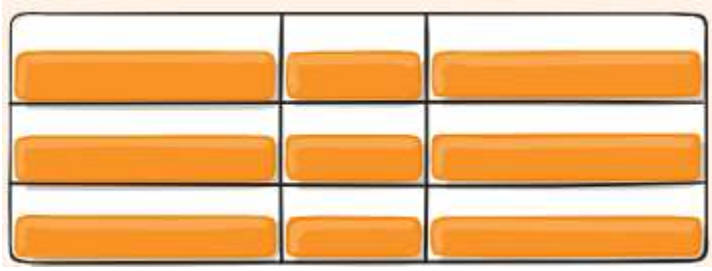
```
.container {
  align-items: start | end | center | stretch;
}
```

Examples:

```
.container {
  align-items: start;
}
```

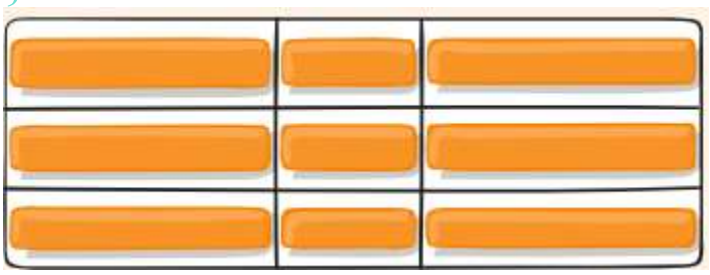


```
.container {
  align-items: end;
}
```

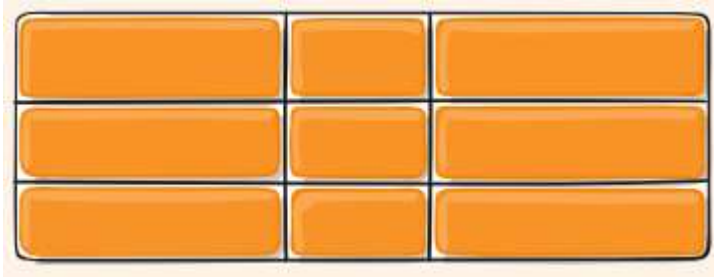


AL-JUNAID INSTITUTE OF GROUP

```
.container {
  align-items: center;
}
```



```
.container {
  align-items: stretch;
}
```



This behavior can also be set on individual grid items via the align-self property.

There are also modifier keywords safe and unsafe (usage is like align-items: safe end). The safe keyword means “try to align like this, but not if it means aligning an item such that it moves into inaccessible overflow area”, while unsafe will allow moving content into inaccessible areas (“data loss”).

AL-JUNAID INSTITUTE OF GROUP

place-items

place-items sets both the align-items and justify-items properties in a single declaration.

Values:

- **<align-items> / <justify-items>** – The first value sets align-items, the second value justify-items. If the second value is omitted, the first value is assigned to both properties.

For more details, see align-items and justify-items.

This can be very useful for super quick multi-directional centering:

```
.center {
  display: grid;
  place-items: center;
}
```

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 inline (row) axis (as opposed to align-content which aligns the grid along the block (column) axis).

Values:

- **start** – aligns the grid to be flush with the start edge of the grid container
- **end** – aligns the grid to be flush with the end edge of the grid container

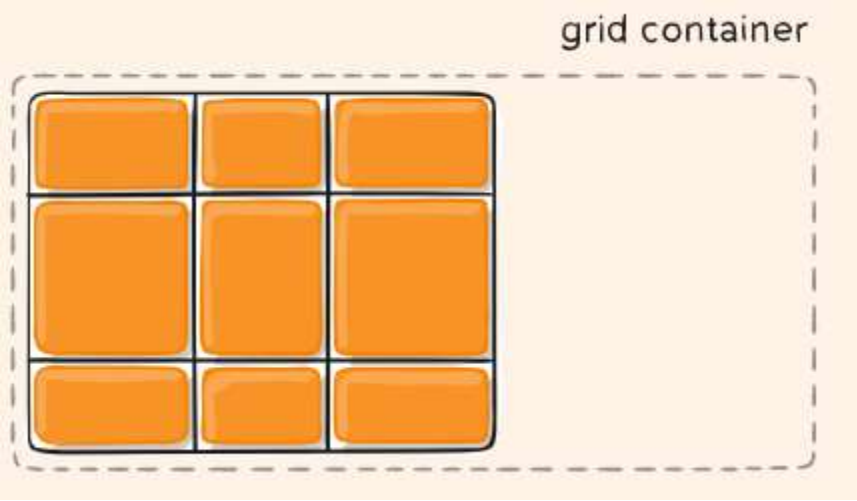
AL-JUNAID INSTITUTE OF GROUP

- **center** – aligns the grid in 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

```
.container {
  justify-content: start | end | center | stretch |
space-around | space-between | space-evenly;
}
```

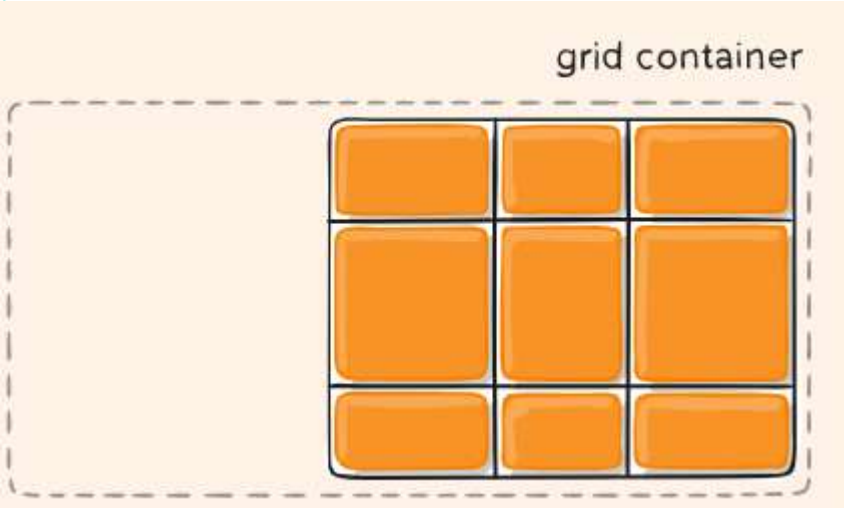
Examples:

```
.container {
  justify-content: start;
}
```

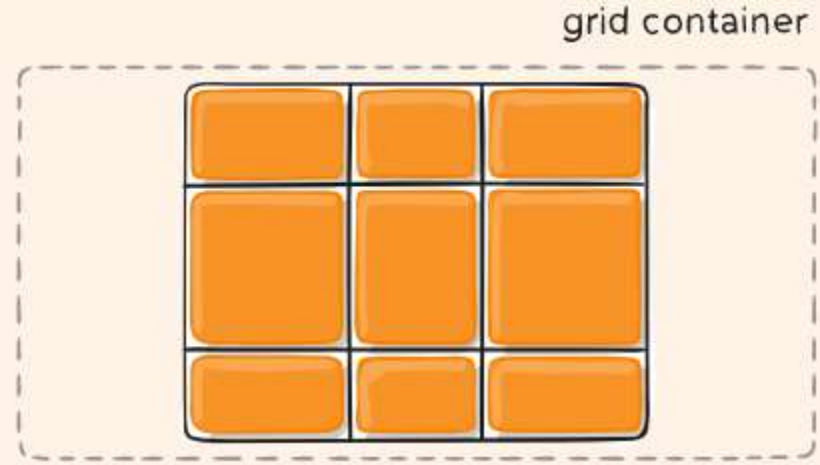


AL-JUNAID INSTITUTE OF GROUP

```
.container {
  justify-content: end;
}
```

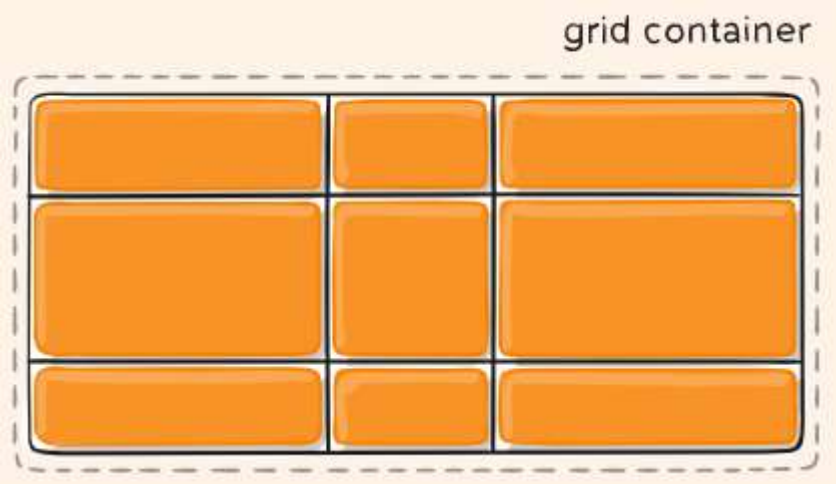



```
.container {
  justify-content: center;
}
```

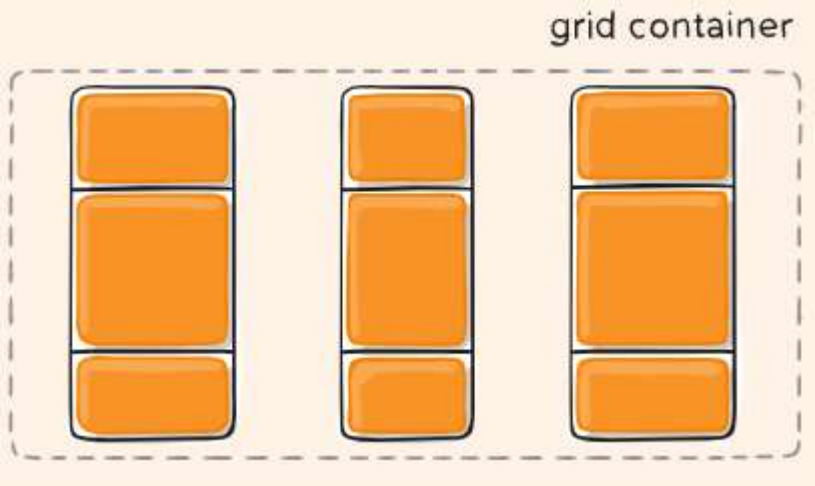



```
.container {
  justify-content: stretch;
}
```

AL-JUNAID INSTITUTE OF GROUP

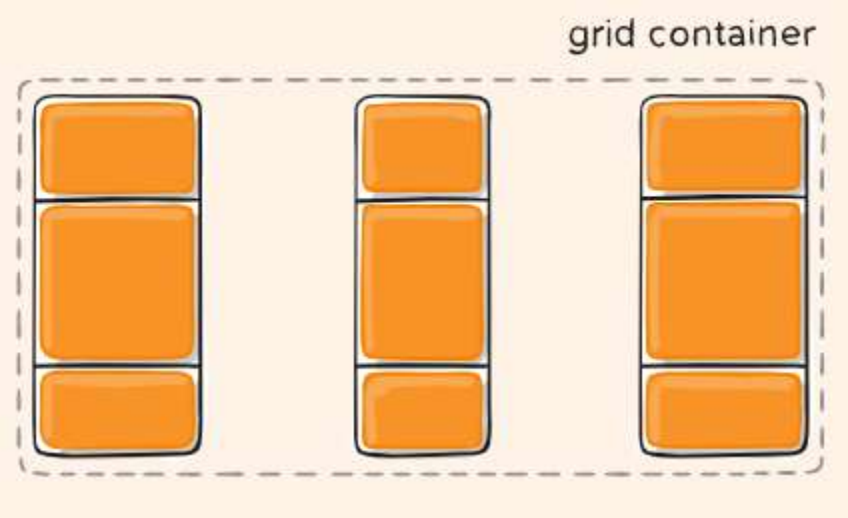



```
.container {
  justify-content: space-around;
}
```

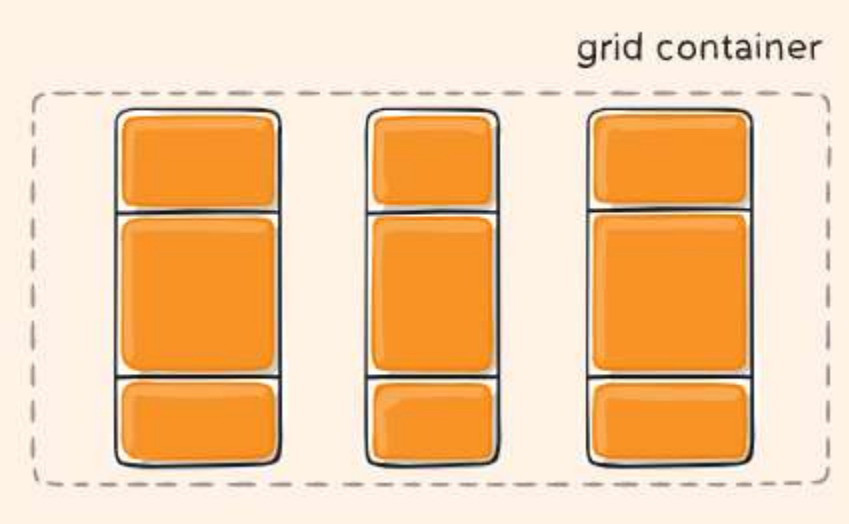



```
.container {
  justify-content: space-between;
}
```

AL-JUNAID INSTITUTE OF GROUP




```
.container {
  justify-content: space-evenly;
}
```



Align-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 block (column) axis (as opposed to justify-content which aligns the grid along the inline (row) axis).

AL-JUNAID INSTITUTE OF GROUP

Values:

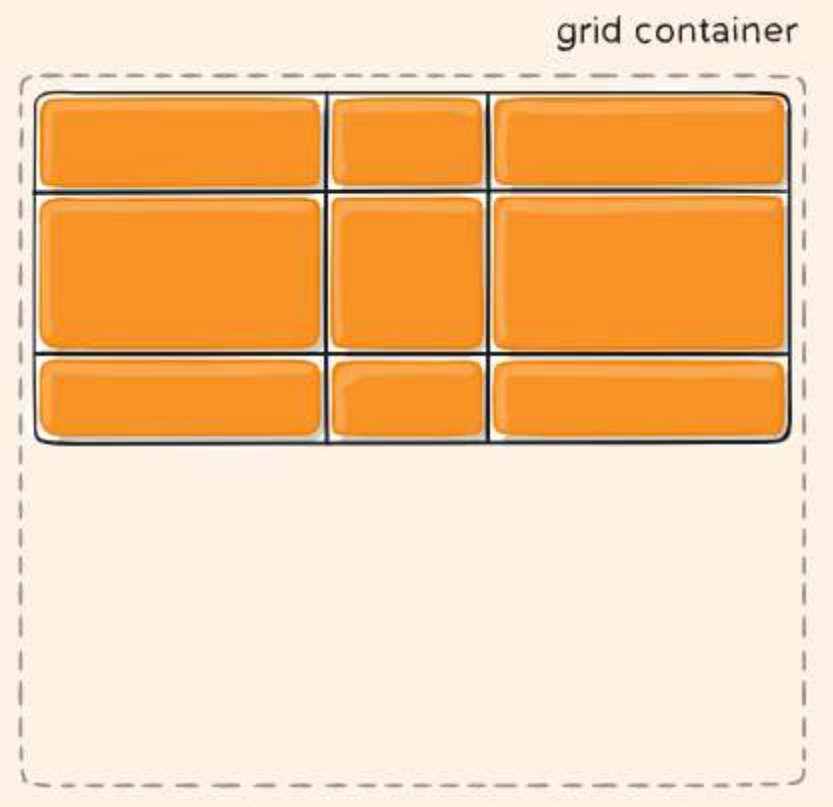
- **start** – aligns the grid to be flush with the start edge of the grid container
- **end** – aligns the grid to be flush with the end edge of the grid container
- **center** – aligns the grid in the center of the grid container
- **stretch** – resizes the grid items to allow the grid to fill the full height 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

```
.container {
  align-content: start | end | center | stretch |
space-around | space-between | space-
evenly;
}
```

Examples:

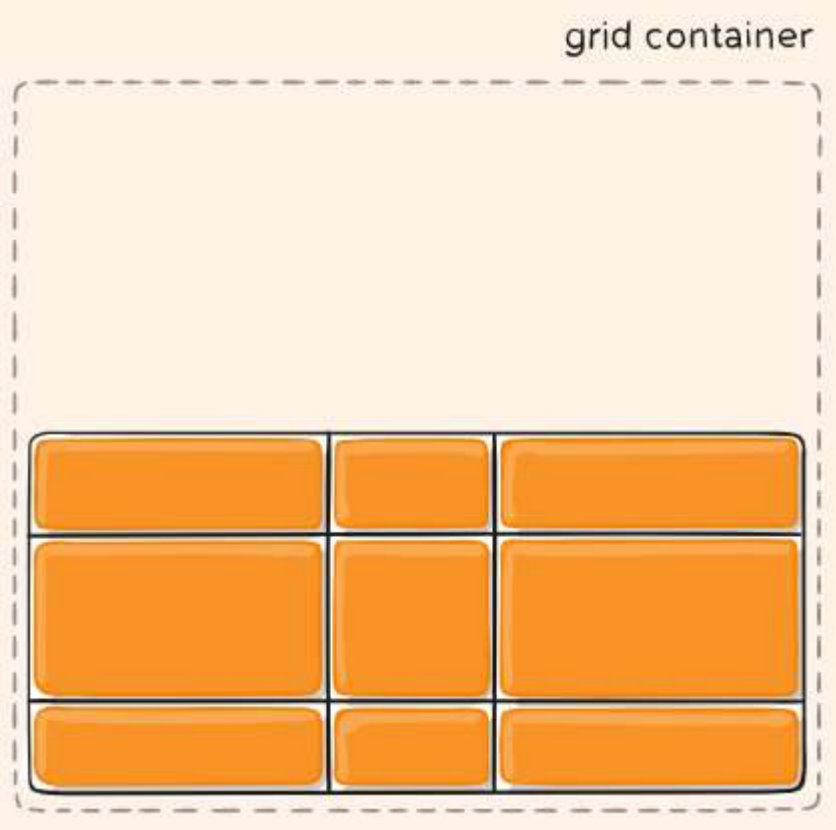
```
.container {
  align-content: start;
}
```

AL-JUNAID INSTITUTE OF GROUP



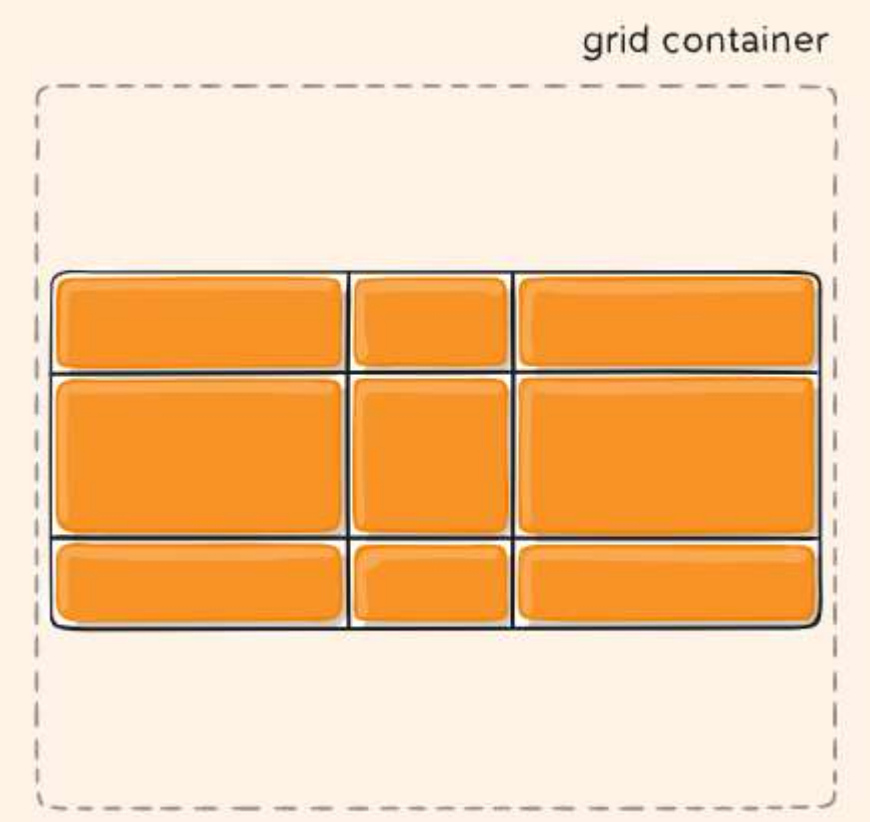

```
.container {
  align-content: end;
}
```

AL-JUNAID INSTITUTE OF GROUP



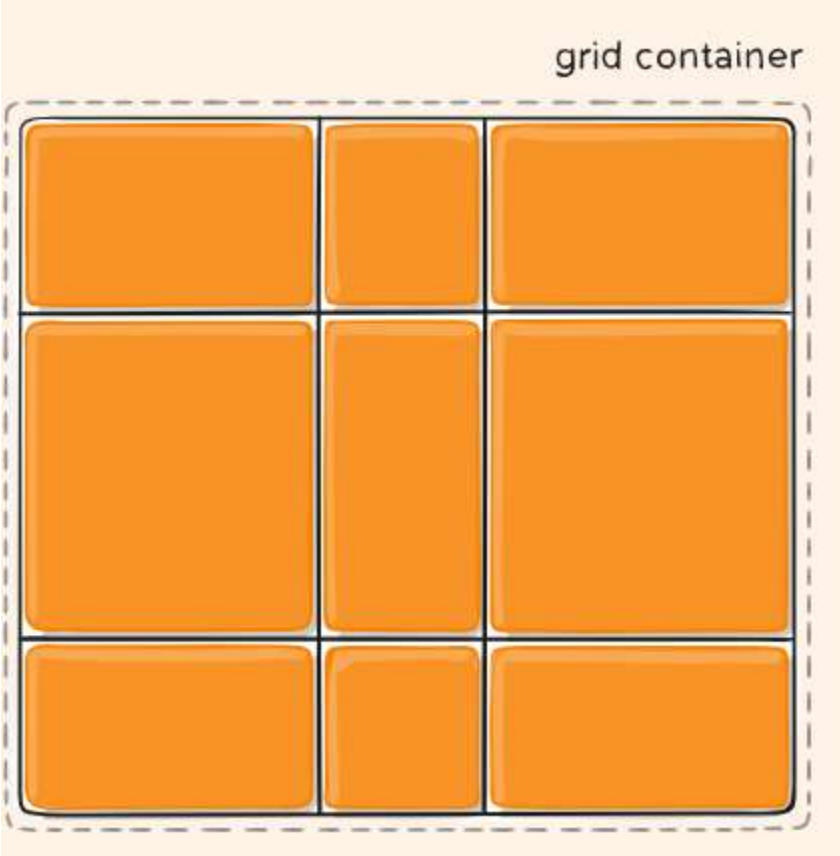

```
.container {
  align-content: center;
}
```

AL-JUNAID INSTITUTE OF GROUP



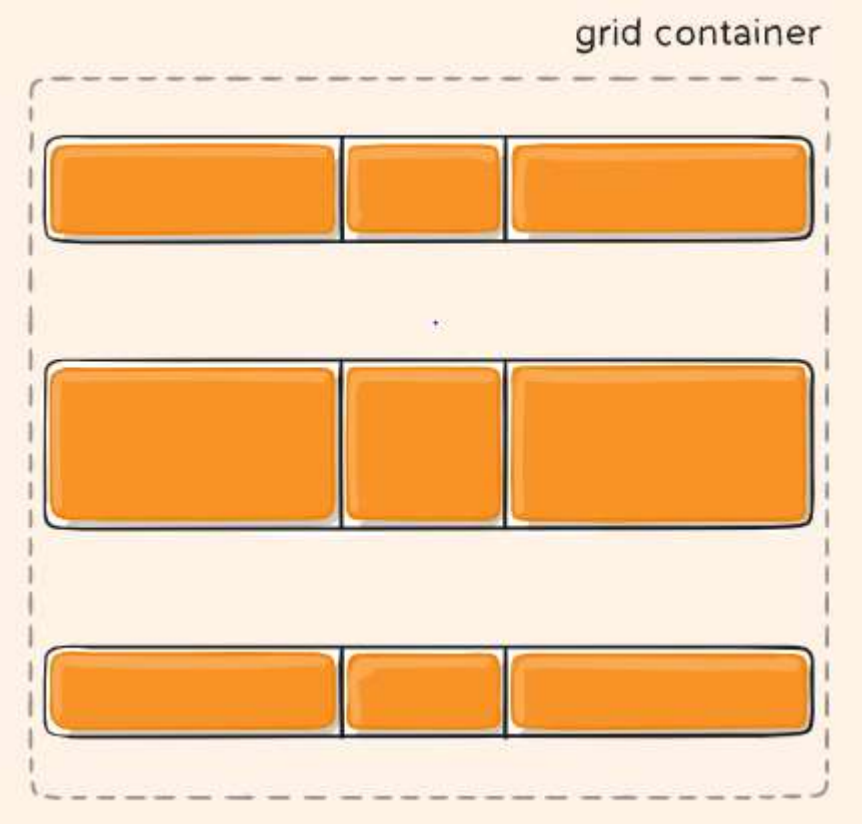

```
.container {
  align-content: stretch;
}
```

AL-JUNAID INSTITUTE OF GROUP



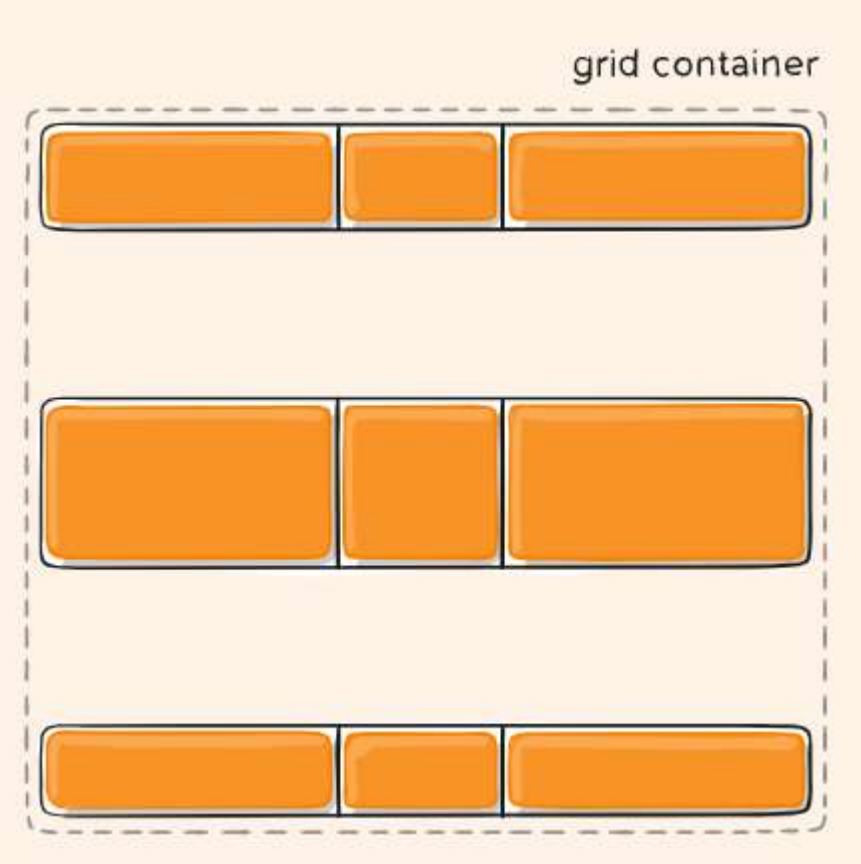

```
.container {
  align-content: space-around;
}
```

AL-JUNAID INSTITUTE OF GROUP



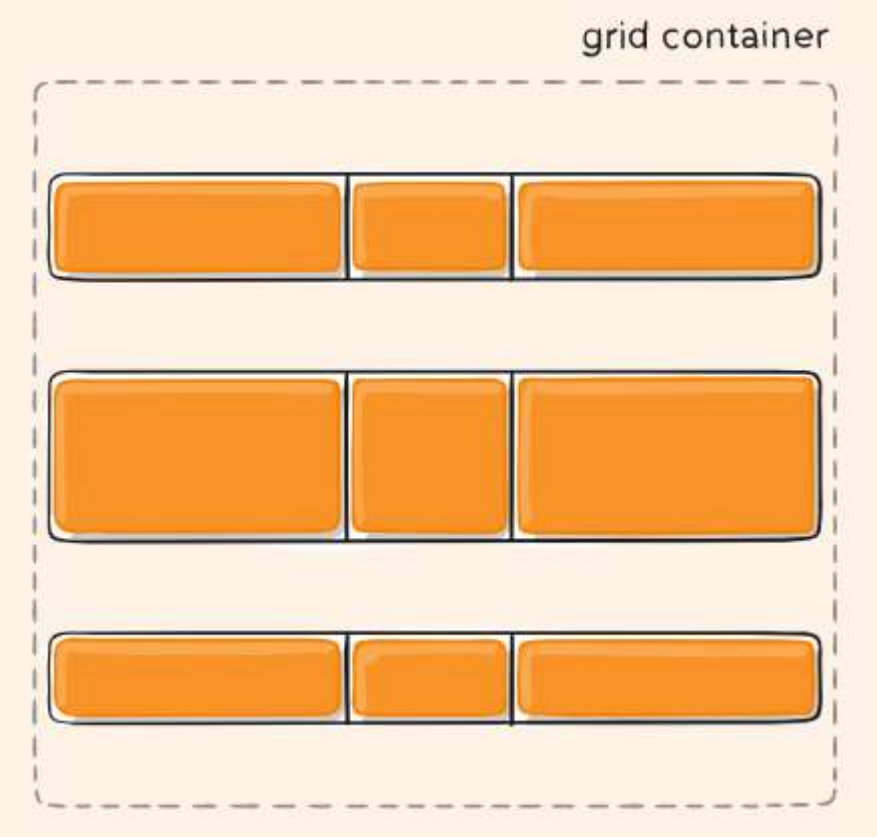

```
.container {
  align-content: space-between;
}
```

AL-JUNAID INSTITUTE OF GROUP




```
.container {
  align-content: space-evenly;
}
```

AL-JUNAID INSTITUTE OF GROUP



place-content

place-content sets both the align-content and justify-content properties in a single declaration.

Values:

- **<align-content> / <justify-content>** – The first value sets align-content, the second value justify-content. If the second value is omitted, the first value is assigned to both properties.

All major browsers except Edge support the place-content shorthand property.

AL-JUNAID INSTITUTE OF GROUP

grid-auto-columns
grid-auto-rows

Specifies the size of any auto-generated grid tracks (aka implicit grid tracks). Implicit tracks get created when there are more grid items than cells in the grid or when a grid item is placed outside of the explicit grid.

Values:

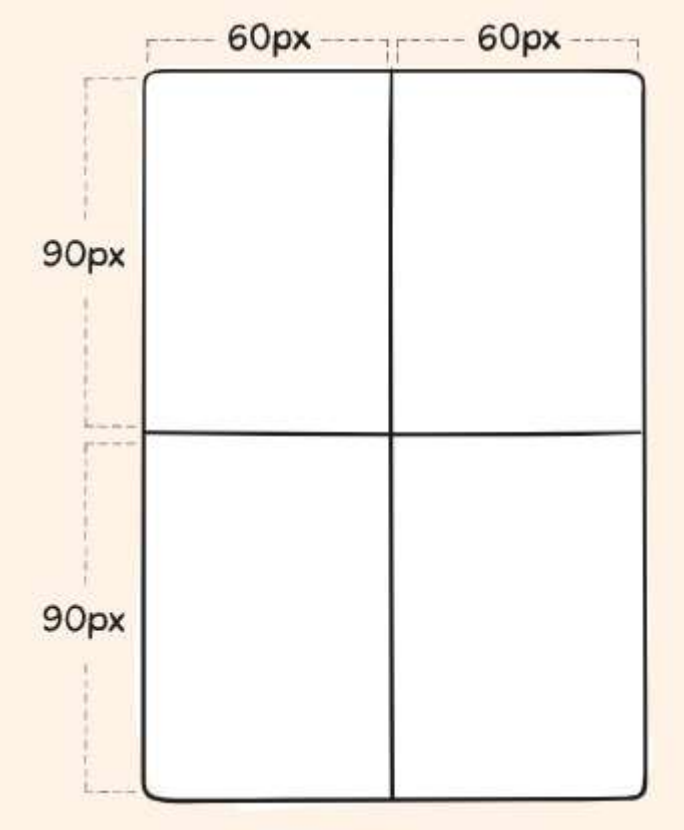
- **<track-size>** – can be a length, a percentage, or a fraction of the free space in the grid (using the fr unit)

```
.container {
  grid-auto-columns: <track-size> ...;
  grid-auto-rows: <track-size> ...;
}
```

To illustrate how implicit grid tracks get created, think about this:

```
.container {
  grid-template-columns: 60px 60px;
  grid-template-rows: 90px 90px;
}
```

AL-JUNAID INSTITUTE OF GROUP

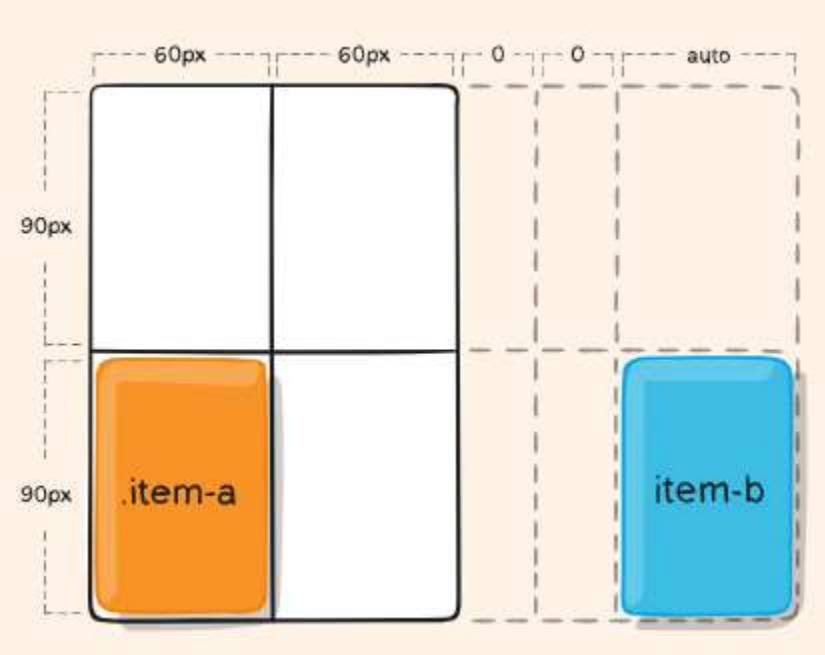


This creates a 2 x 2 grid.

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;
```

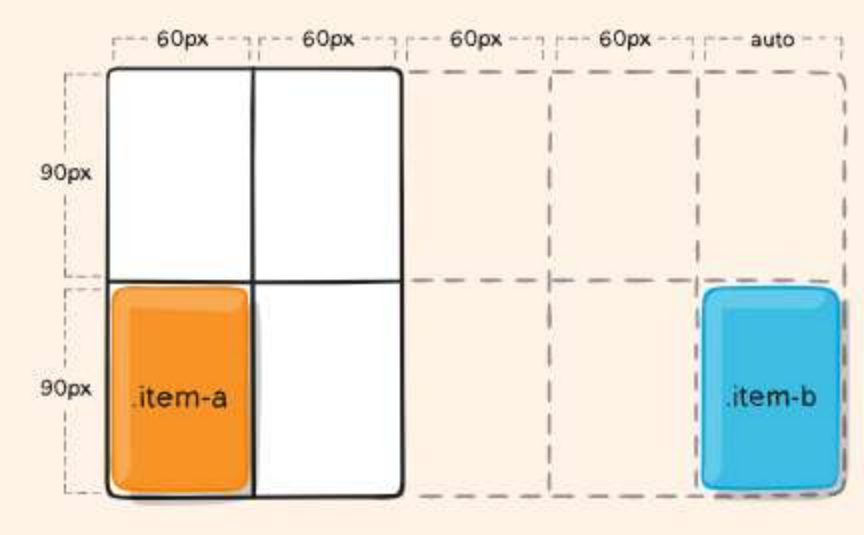
```
}
```



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;
}
```

AL-JUNAID INSTITUTE OF GROUP



grid-auto-flow

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 (default)
- **column** – tells the auto-placement algorithm to fill in each column in turn, adding new columns as necessary
- **dense** – tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later

```
.container {
  grid-auto-flow: row | column | row dense | column dense;
}
```

Note that **dense** only changes the visual order of your items and might cause them to appear out of order, which is bad for accessibility.

Examples:

Consider this HTML:

```
<section class="container">
  <div class="item-a">        </div>
  <div class="item-b">        </div>
  <div class="item-c">        </div>
  <div class="item-d">        </div>
  <div class="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):

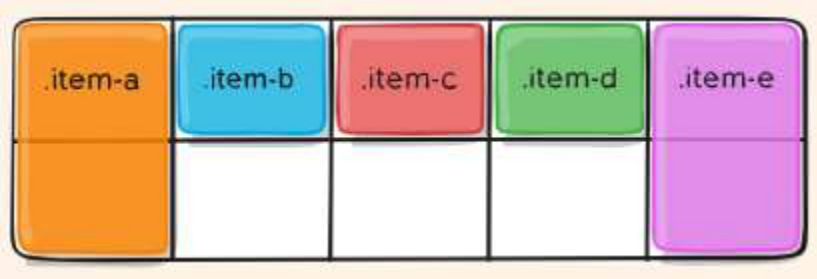
```
.container {
  display: grid;
  grid-template-columns: 60px 60px 60px 60px 60px;
  grid-template-rows: 30px 30px;
  grid-auto-flow: row;
}
```

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 {
```

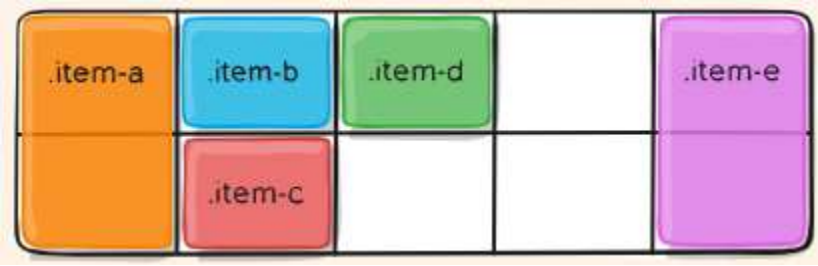
```
  grid-column: 5;
  grid-row: 1 / 3;
}
```

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;
}
```



AL-JUNAID INSTITUTE OF GROUP

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 (Note: You can only specify the explicit or the implicit grid properties in a single grid declaration).

Values:

- **none** – sets all sub-properties to their initial values.
- **<grid-template>** – works the same as the grid-template shorthand.
- **<grid-template-rows> / [auto-flow && dense?] <grid-auto-columns>?** – sets grid-template-rows to the specified value. If the auto-flow keyword is to the right of the slash, it sets grid-auto-flow to column. If the dense keyword is specified additionally, the auto-placement algorithm uses a "dense" packing algorithm. If grid-auto-columns is omitted, it is set to auto.
- **[auto-flow && dense?] <grid-auto-rows>? / <grid-template-columns>** – sets grid-template-columns to the specified value. If the auto-flow keyword is to the left of the slash, it sets grid-auto-flow to row. If the dense keyword is specified additionally, the auto-placement algorithm uses a "dense" packing algorithm. If grid-auto-rows is omitted, it is set to auto.

Examples:

The following two code blocks are equivalent:

```
.container {
  grid: 100px 300px / 3fr 1fr;
}
```

```
.container {
```

```
  grid-template-rows: 100px 300px;
  grid-template-columns: 3fr 1fr;
}
```

The following two code blocks are equivalent:

```
.container {
  grid: auto-flow / 200px 1fr;
}
.container {
  grid-auto-flow: row;
  grid-template-columns: 200px 1fr;
}
```

The following two code blocks are equivalent:

```
.container {
  grid: auto-flow dense 100px / 1fr 2fr;
}


.container {
  grid-auto-flow: row dense;
  grid-auto-rows: 100px;
  grid-template-columns: 1fr 2fr;
}
```

And the following two code blocks are equivalent:

```
.container {
  grid: 100px 300px / auto-flow 200px;
```

```
}

.container {
  grid-template-rows: 100px 300px;
  grid-auto-flow: column;
  grid-auto-columns: 200px;
}
```

It also accepts a more complex but quite handy syntax for setting everything at once. You specify grid-template-areas, grid-template-rows and grid-template-columns, and all the other sub-properties are set to their initial values. What you're doing is specifying the line names and track sizes inline with their respective grid areas. This is easiest to describe with an example:

```
.container {
  grid: [row1-start] "header header header" 1fr
[row1-end]
        [row2-start] "footer footer footer" 25px [row2-
end]
        / auto 50px auto;
}
```

That's equivalent to this:

```
.container {
  grid-template-areas:
    "header header header"
    "footer footer footer";
```

```
  grid-template-rows: [row1-start] 1fr [row1-end row2-start] 25px [row2-end];
  grid-template-columns: auto 50px auto;
}
```

Properties for the Children (Grid Items)

float, display: inline-block, display: table-cell, vertical-align and column-* properties have no effect on a grid item.

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:

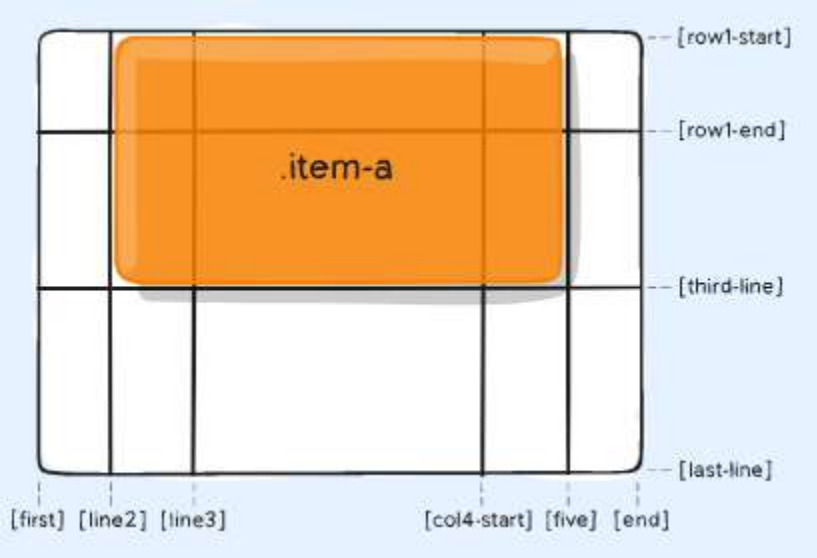
- **<line>** – can be a number to refer to a numbered grid line, or a name to refer to a named grid line
- **span <number>** – the item will span across the provided number of grid tracks
- **span <name>** – the item will span across until it hits the next line with the provided name
- **auto** – indicates auto-placement, an automatic span, or a default span of one

```
.item {
  grid-column-start: <number> | <name> | span <number> | span <name> | auto;
```

```
  grid-column-end: <number> | <name> | span <number> |
span <name> | auto;
  grid-row-start: <number> | <name> | span <number> | span
<name> | auto;
  grid-row-end: <number> | <name> | span <number> | span
<name> | auto;
}
```

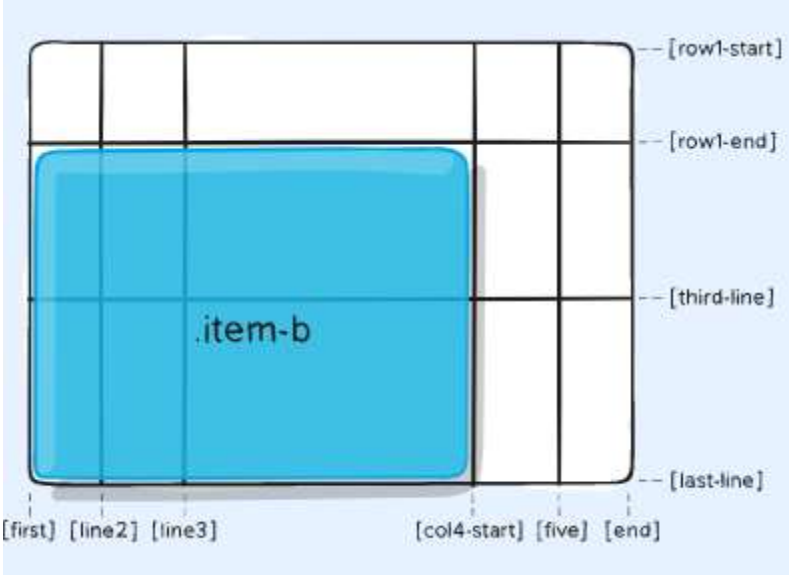
Examples:

```
.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

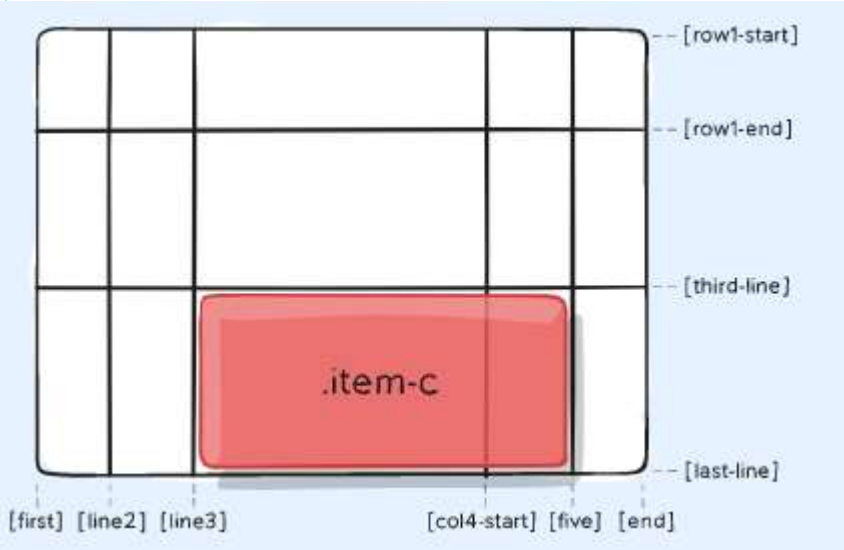
```
.item {
  grid-column: <start-line> / <end-line> | <start-line> /
span <value>;
```

AL-JUNAID INSTITUTE OF GROUP

```
  grid-row: <start-line> / <end-line> | <start-line> / span
<value>;
}
```

Example:

```
.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

```
.item {
  grid-area: <name> | <row-start> / <column-start> / <row-end> / <column-end>;
}
```

Examples:

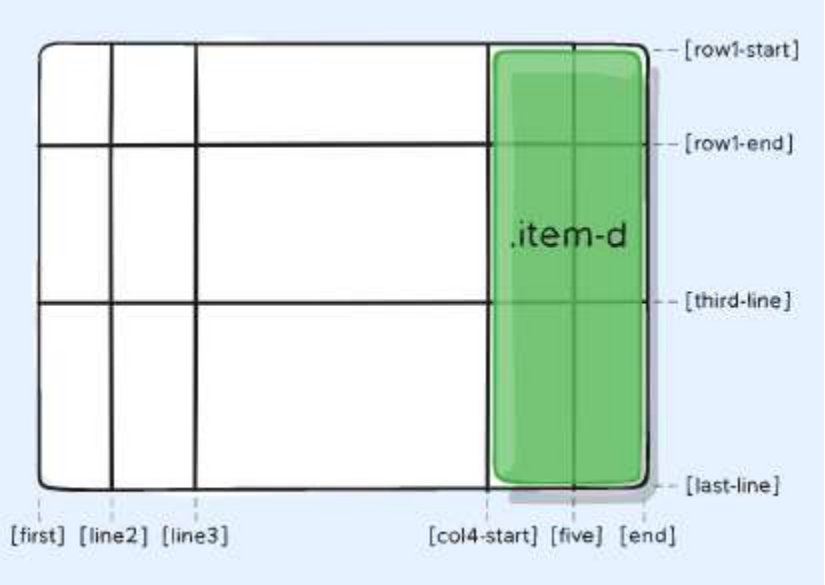
As a way to assign a name to the item:

```
.item-d {
  grid-area: header;
}
```

As the short-shorthand for grid-row-start + grid-column-start + grid-row-end + grid-column-end:

```
.item-d {
  grid-area: 1 / col4-start / last-line / 6;
}
```

AL-JUNAID INSTITUTE OF GROUP



Justify-self

Aligns a grid item inside a cell along the inline (row) axis (as opposed to align-self which aligns along the block (column) axis). This value applies to a grid item inside a single cell.

Values:

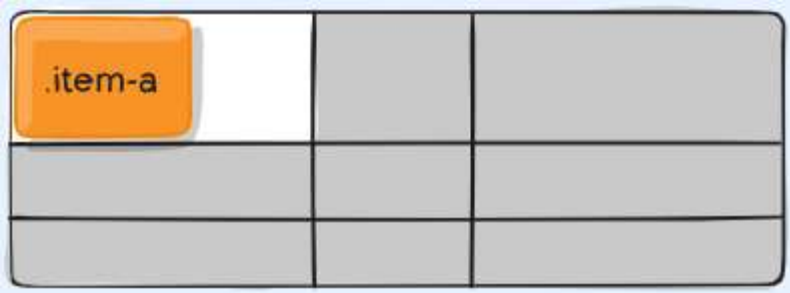
- **start** – aligns the grid item to be flush with the start edge of the cell
- **end** – aligns the grid item to be flush with the end edge of the cell
- **center** – aligns the grid item in the center of the cell
- **stretch** – fills the whole width of the cell (this is the default)

```
.item {
  justify-self: start | end | center | stretch;
}
```

AL-JUNAID INSTITUTE OF GROUP

Examples:

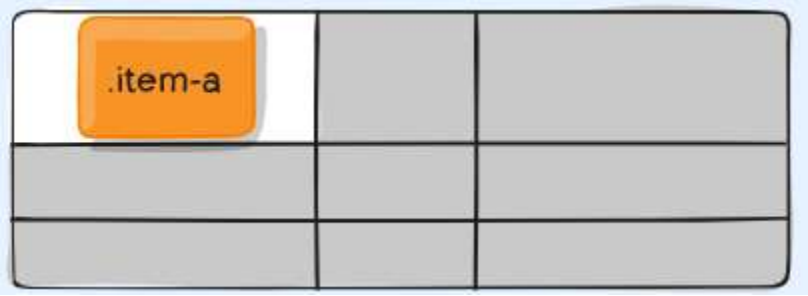
```
.item-a {
  justify-self: start;
}
```




```
.item-a {
  justify-self: end;
}
```




```
.item-a {
  justify-self: center;
}
```



AL-JUNAID INSTITUTE OF GROUP

```
.item-a {
  justify-self: stretch;
}
```



Align-self

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

Values:

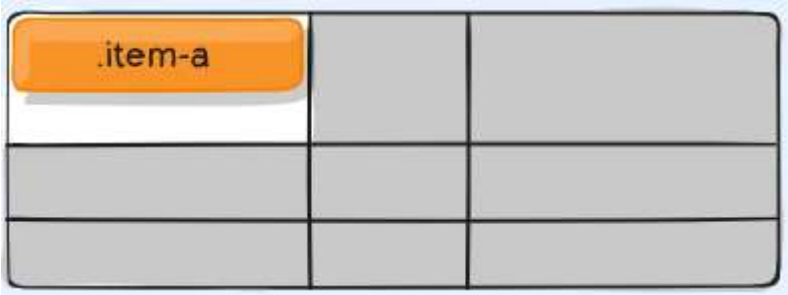
- **start** – aligns the grid item to be flush with the start edge of the cell
- **end** – aligns the grid item to be flush with the end edge of the cell
- **center** – aligns the grid item in the center of the cell
- **stretch** – fills the whole height of the cell (this is the default)

```
.item {
  align-self: start | end | center | stretch;
}
```

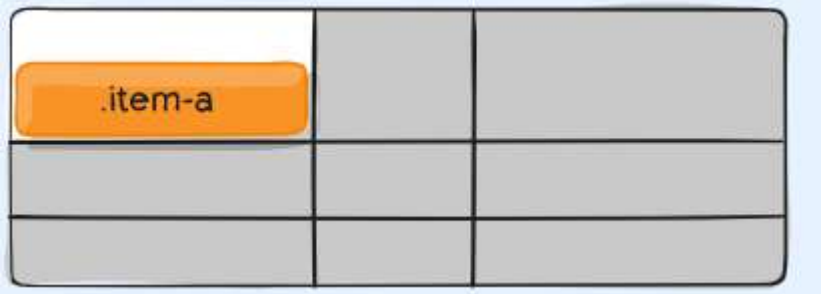
Examples:

```
.item-a {
  align-self: start;
```

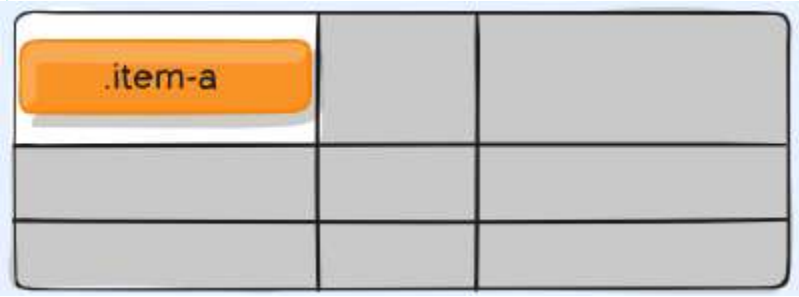
```
}
```



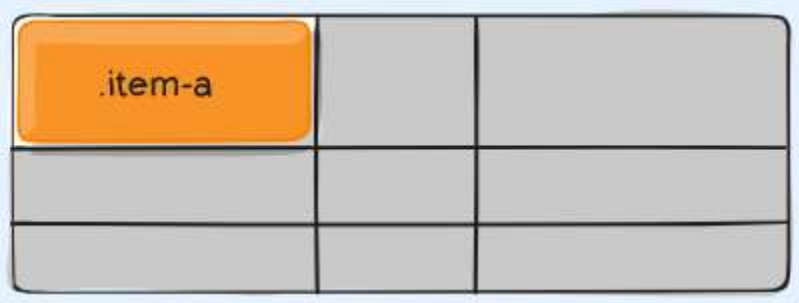

```
.item-a {
  align-self: end;
}
```




```
.item-a {
  align-self: center;
}
```



```
.item-a {
  align-self: stretch;
}
```



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

place-self

place-self sets both the align-self and justify-self properties in a single declaration.

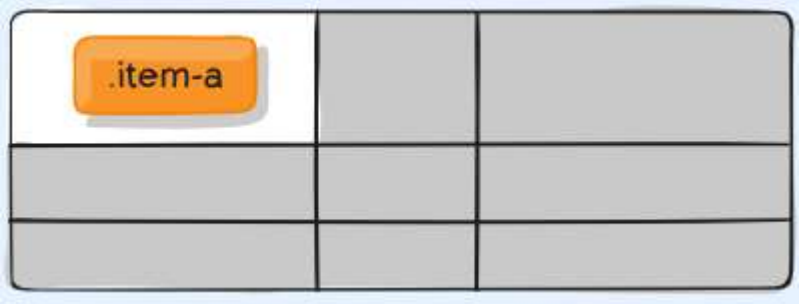
Values:

- **auto** – The "default" alignment for the layout mode.
- **<align-self> / <justify-self>** – The first value sets align-self, the second value justify-self. If the second value is omitted, the first value is assigned to both properties.

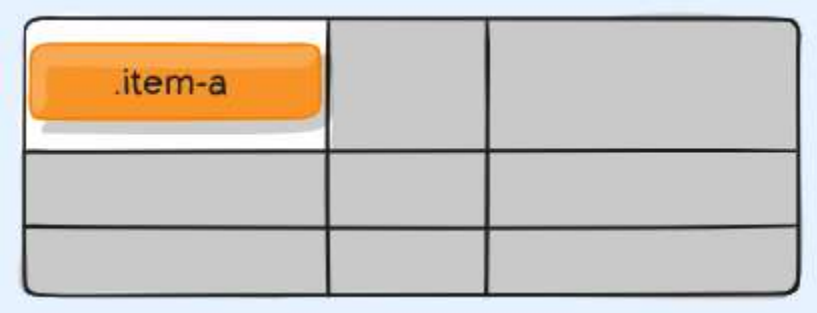
Examples:

```
.item-a {
  place-self: center;
}
```

AL-JUNAID INSTITUTE OF GROUP




```
.item-a {
  place-self: center stretch;
}
```



All major browsers except Edge support the place-self shorthand property.

Special Units & Functions

fr units

You'll likely end up using a lot of fractional units in CSS Grid, like 1fr. They essentially mean "portion of the remaining space". So a declaration like:

```
grid-template-columns: 1fr 3fr;
```

Means, loosely, 25% 75%. Except that those percentage values are much more firm than fractional units are. For example, if you added padding to those percentage-based columns, now you've broken 100% width (assuming a content-box box model). Fractional units

also much more friendly in combination with other units, as you can imagine:

```
grid-template-columns: 50px min-content 1fr;
```

Sizing Keywords

When sizing rows and columns, you can use all the lengths you are used to, like px, rem, %, etc, but you also have keywords:

- min-content: the minimum size of the content. Imagine a line of text like “The very long hotdog.”, the min-content is likely the width of the world “The”.
- max-content: the maximum size of the content. Imagine the sentence above, the max-content is the length of the whole sentence.
- auto: this keyword is a lot like fr units, except that they “lose” the fight in sizing against fr units when allocating the remaining space.
- fit-content: use the space available, but never less than min-content and never more than max-content.
- fractional units: see above

Sizing Functions

- The minmax() function does exactly what it seems like: it sets a minimum and maximum value for what the length is able to be. This is useful for in combination with relative units. Like you may want a column to be only able to shrink so far.

  ```
  grid-template-columns: minmax(100px, 1fr) 3fr;
  ```
- The min() function.
- The max() function.

AL-JUNAID INSTITUTE OF GROUP

The repeat() Function and Keywords

The repeat() function can save some typing:

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

/* easier: */
grid-template-columns:
  repeat(8, 1fr);

/* especially when: */
grid-template-columns:
  repeat(8, minmax(10px, 1fr));
```

But repeat() can get extra fancy when combined with keywords:

- auto-fill: Fit as many possible columns as possible on a row, even if they are empty.
- auto-fit: Fit whatever columns there are into the space. Prefer expanding columns to fill space rather than empty columns.

```
grid-template-columns:
repeat(auto-fit, minmax(250px, 1fr));
```

Masonry

An experimental feature of CSS grid is masonry layout. Note that there are lots of approaches to CSS masonry, but mostly of them are trickery and either have major downsides or aren't what you quite expect.

The spec has an official way now, and this is behind a flag in Firefox:

AL-JUNAID INSTITUTE OF GROUP

```
.container {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-rows: masonry;
}
```

Subgrid

Subgrid is an extremely useful feature of grids that allows grid items to have a grid of their own that inherits grid lines from the parent grid.

```
.parent-grid {
  display: grid;
  grid-template-columns: repeat(9, 1fr);
}
.grid-item {
  grid-column: 2 / 7;

  display: grid;
  grid-template-columns: subgrid;
}
.child-of-grid-item {
  /* gets to participate on parent grid! */
  grid-column: 3 / 6;
}
```

AL-JUNAID INSTITUTE OF GROUP

This is only supported in Firefox right now, but it really needs to get everywhere.

It's also useful to know about display: contents;. This is not the same as Subgrid, but it can be a useful tool sometimes in a similar fashion.

```
<div class="grid-parent">
  <div class="grid-item"></div>
  <div class="grid-item"></div>
  <ul style="display: contents;">
    <!-- These grid-items get to participate on
         the same grid!-->
    <li class="grid-item"></li>
    <li class="grid-item"></li>
  </ul>
</div>
```