Pseudo clases, pseudo elementos y grid

Sí, bueno, es demasiado específico como para encontrar un meme que se ajuste al tema...

Pseudo clases

Según mozilla, “Una **pseudoclase CSS** es una palabra clave que se añade a los selectores y que especifica un estado especial del elemento seleccionado.”

Ya sabemos usar algunas!

```
div:hover {
  background-color: #F89B4D;
}
```

Las pseudoclases son:

- :active
- :checked
- :default
- :dir()
- :disabled
- :empty
- :enabled
- :first
- :first-child
- :first-of-type
- :fullscreen
- :focus
- :hover
- :indeterminate
- :in-range
- :invalid
- :lang()
- :last-child
- :last-of-type
- :left
- :link
- :not()
- :nth-child()
- :nth-last-child()
- :nth-last-of-type()
- :nth-of-type()
- :only-child
- :only-of-type
- :optional
- :out-of-range
- :read-only
- :read-write
- :required
- :right
- :root
- :scope (en-US)
- :target
- :valid
- :visited

De ellas, las que más se usan suelen ser:

- :hover
- :active
- :visited
- :focus
- :disabled
- :first-child
- :last-child
- nth-child() => entre paréntesis ponemos el número de elemento hijo
- :not() => está buenísima si queremos que afecte a todos menos a, por ej, los que tienen cierta class

Pseudo elementos

Según mozilla, “Al igual que las `pseudo-classes`, los pseudo-elementos se añaden a los selectores, pero en cambio, no describen un estado especial sino que, permiten añadir estilos a una parte concreta del documento. Por ejemplo, el pseudoelemento `::first-line` selecciona solo la primera línea del elemento especificado por el selector.”

```
selector::pseudo-elemento { propiedad: valor; }
```

Hay bastantes menos que pseudo clases, aunque son un poquito más tricky de usar:

- `::after`
- `::before`
- `::first-letter`
- `::first-line`
- `::selection`
- `::backdrop`
- `::placeholder` 🧪
- `::marker` 🧪
- `::spelling-error` 🧪
- `::grammar-error` (en-US) 🧪

Notas

De vez en cuando se utilizan dos puntos dobles (::) en vez de solo uno (:). Esto forma parte de CSS3 y de un intento para distinguir pseudo-elementos de pseudo-clases.

> **Nota:** `::selection` siempre se escribe con dos puntos dobles (::).

Solo se puede usar un pseudo-elemento por selector. Debe aparecer después del selector simple.

Bueno, y ahora sí a los grid!

Siempre leyendo la documentación oficial, nos enteramos que “**CSS Grid layout** contiene funciones de diseño dirigidas a los desarrolladores de aplicaciones web. El CSS grid se puede utilizar para lograr muchos diseños diferentes. También se destaca por permitir dividir una página en áreas o regiones principales, por definir la relación en términos de tamaño, posición y capas entre partes de un control construido a partir de primitivas HTML.

Al igual que las tablas, el grid layout permite a un autor alinear elementos en columnas y filas. Sin embargo, con CSS grid son posibles muchos más diseños y de forma más sencilla que con las tablas. Por ejemplo, los elementos secundarios de un contenedor de cuadrícula podrían posicionarse para que se solapen y se superpongan, de forma similar a los elementos posicionados en CSS.”

“

Básicamente es similar a flex en cierto punto, solo que podemos controlar rows, col, o ambos.

No van a encontrar algo mejor que esto: https://css-tricks.com/snippets/css/complete-guide-grid/ , pero yo sé que les gustan mis capturas así que se las dejo.

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

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

Igual que ponerlo como flex, no?

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

```css
.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
  */
}
```

fr resulta que es fracción:

```css
.grid {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-column-gap: 10px;
}
```

Esto generaría 4 columnas (4), cada una de 1 unidad (1fr), y todas tendrían en cuenta que tienen que tener un gap de 10px entre ellas.

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

grid-template-areas

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.

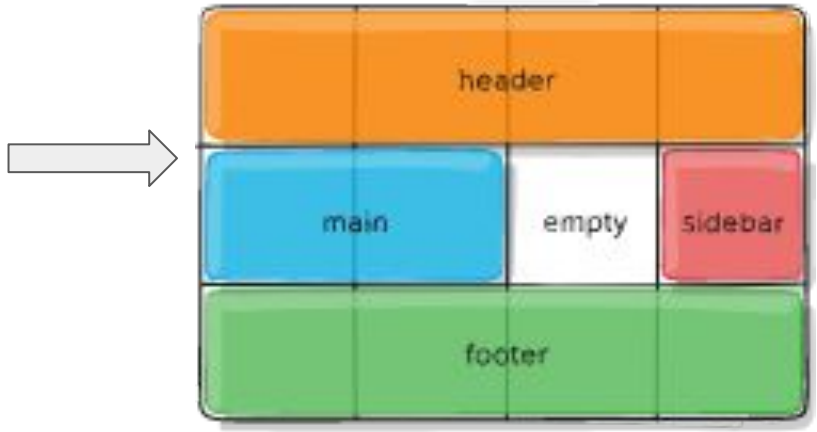
Values:

- **<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

Example:

```css
.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";
}
```



```css
.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:

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

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:

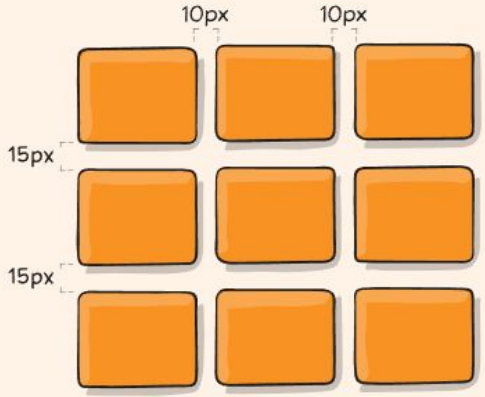
- **<line-size>** – a length value

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

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

Example:

```css
.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.

justify-items

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.

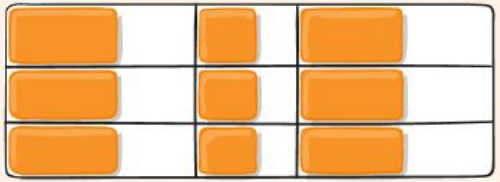
Values:

- **`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)

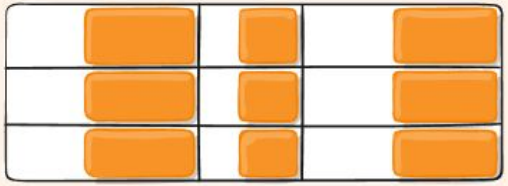
```css
.container {
  justify-items: start | end | center | stretch;
}
```

Examples:

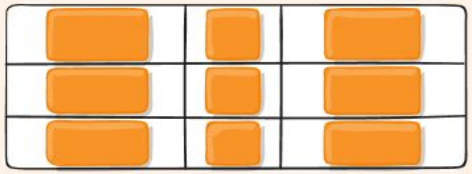
```css
.container {
  justify-items: start;
}
```



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



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



```css
.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.

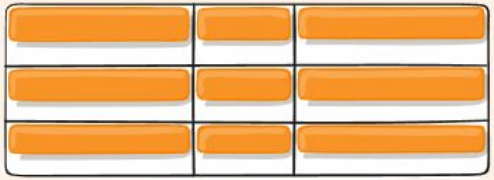
Values:

- **`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 height of the cell (this is the default)

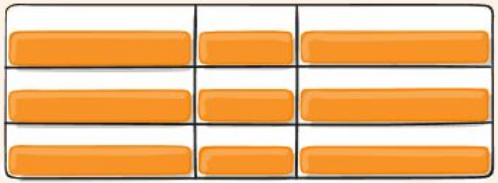
```css
.container {
  align-items: start | end | center | stretch;
}
```

Examples:

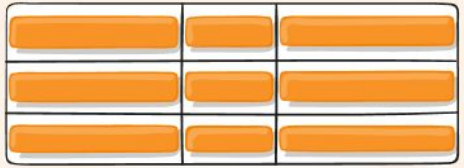
```css
.container {
  align-items: start;
}
```



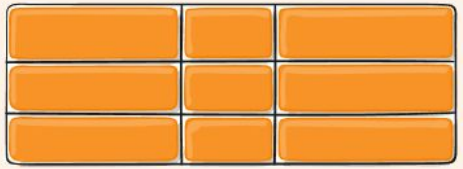
```css
.container {
  align-items: end;
}
```



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



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



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

`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:

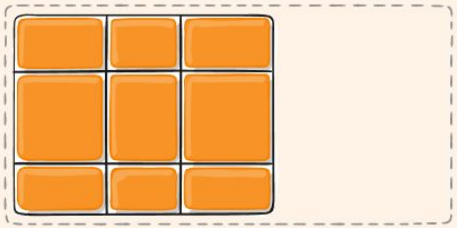
- **`start`** – aligns the grid to be flush with the start edge of the grid container
- **`end`** – aligns the grid to be flush with the end edge of the grid container
- **`center`** – aligns the grid in the center of the grid container
- **`stretch`** – resizes the grid items to allow the grid to fill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

```css
.container {
  justify-content: start | end | center | stretch | space-arou
}
```

Examples:

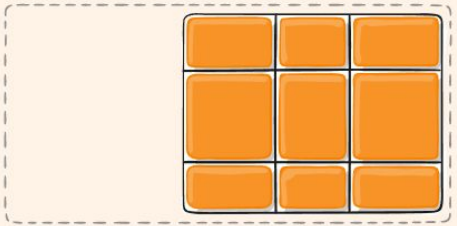
```css
.container {
  justify-content: start;
}
```

grid container



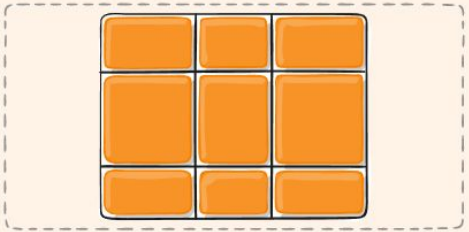
```css
.container {
  justify-content: end;
}
```

grid container



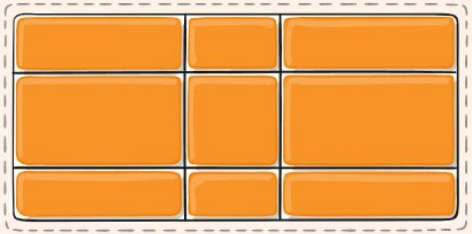
```css
.container {
  justify-content: center;
}
```

grid container



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

grid container



`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).

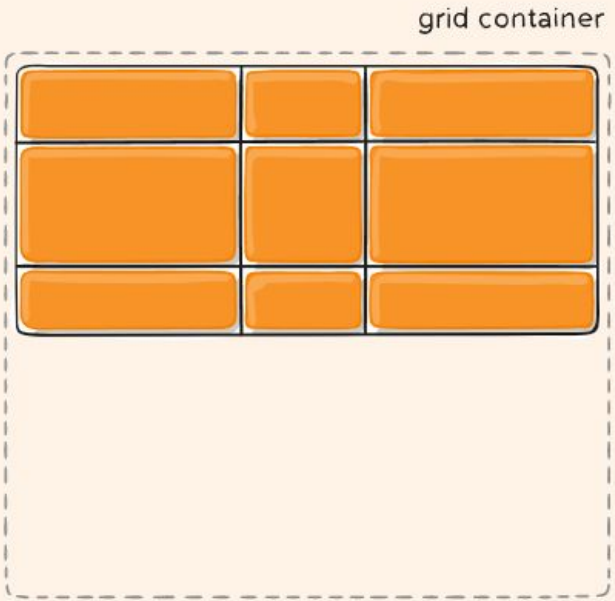
Values:

- **`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

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

Examples:

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



Todas esas fueron propiedades para EL PADRE. Es igual que con flex, los hijos pueden tener sus propias propiedades también:

`grid-column-start`

`grid-column-end`

`grid-row-start`

`grid-row-end`

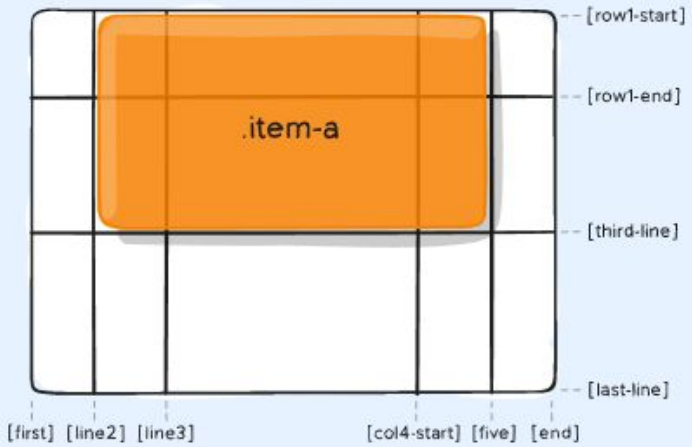
Determines a grid item's location within the grid by referring to specific grid lines. `grid-column-start`/`grid-row-start` is the line where the item begins, and `grid-column-end`/`grid-row-end` is the line where the item ends.

Values:

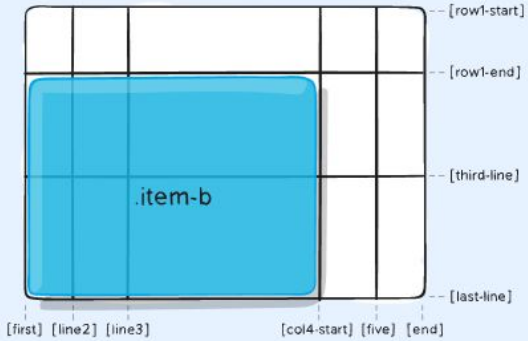
- **`<line>`** – can be a number to refer to a numbered grid line, or a name to refer to a named grid line
- **`span <number>`** – the item will span across the provided number of grid tracks
- **`span <name>`** – the item will span across until it hits the next line with the provided name
- **`auto`**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

Examples:

```css
.item-a {
  grid-column-start: 2;
  grid-column-end: five;
  grid-row-start: row1-start;
  grid-row-end: 3;
}
```



```css
.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.

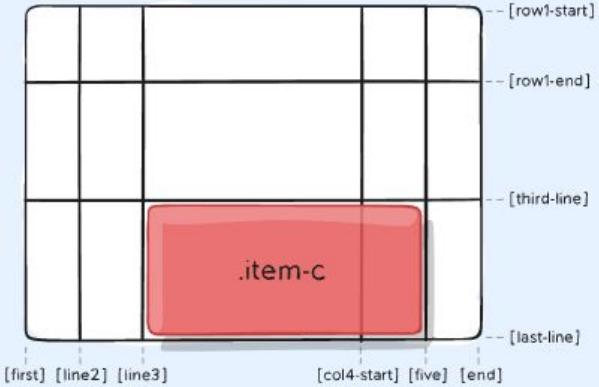
Values:

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

```css
.item {
  grid-column: <start-line> / <end-line> | <start-line> / span
  grid-row: <start-line> / <end-line> | <start-line> / span <v
}
```

Example:

```css
.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

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

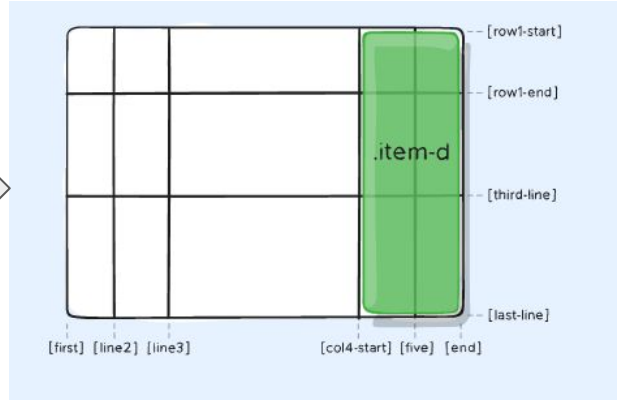
Examples:

As a way to assign a name to the item:

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

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

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



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.

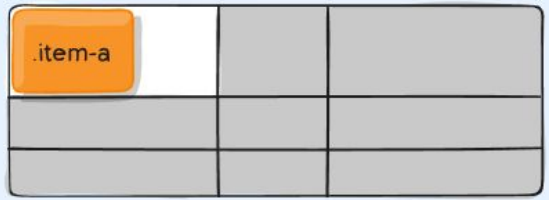
Values:

- **`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)

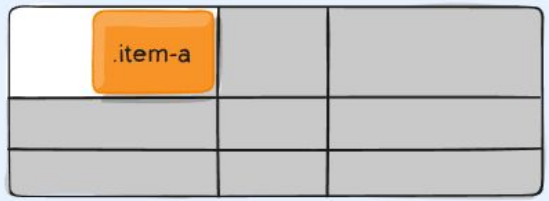
```css
.item {
  justify-self: start | end | center | stretch;
}
```

Examples:

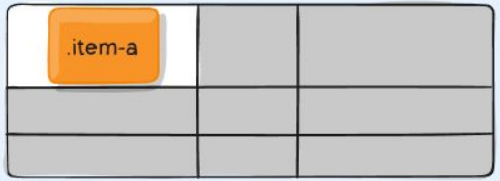
```css
.item-a {
  justify-self: start;
}
```



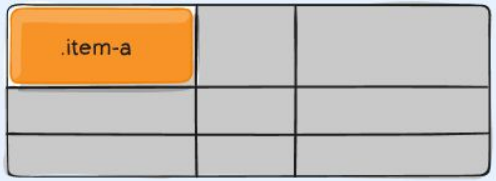

```css
.item-a {
  justify-self: end;
}
```




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




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



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.

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.

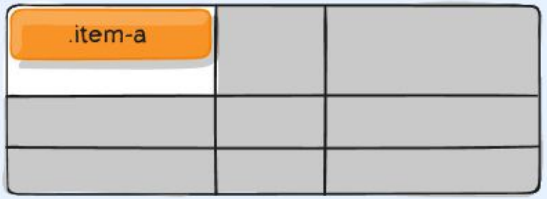
Values:

- **`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)

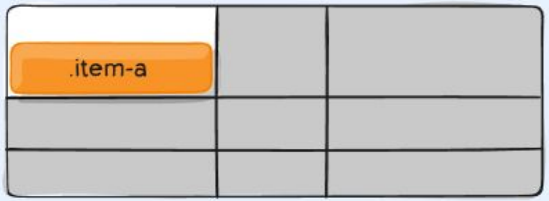
```css
.item {
  align-self: start | end | center | stretch;
}
```

Examples:

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

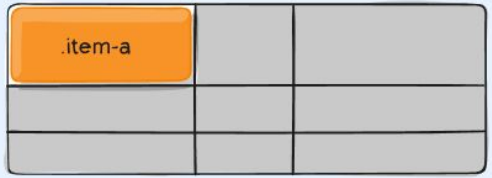



```css
.item-a {
  align-self: end;
}
```




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


```css
.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.

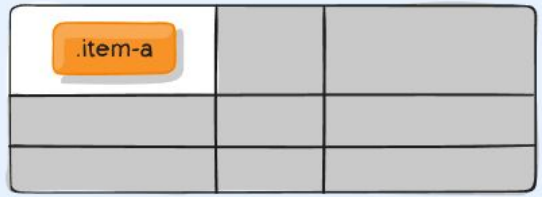
Values:

- **`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.

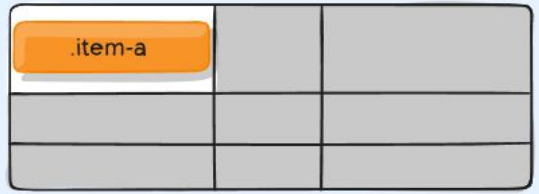
Examples:

Examples:

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




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




NEW THINGS?
I'M TERRIFIED OF THAT.
makeameme.org