CSS Grid Layout Cheatsheet

A comprehensive cheat sheet for CSS Grid Layout, covering fundamental concepts, properties for containers and items, and practical examples to help you master grid-based layouts.

Grid Container Properties

Display Property

Defines the element as a grid container, enabling grid layout for its direct children.

`display: grid;` - Creates a block-level grid.
`display: inline-grid;` - Creates an inline-level grid.

Example:

```
.container {
  display: grid;
}
```

Grid Template Columns

Specifies the number and width of columns in the grid.

`grid-template-columns: <track-size> ...;` - Defines column sizes explicitly.
`grid-template-columns: repeat(<n>, <track-size>);` - Repeats column definitions.
`grid-template-columns: auto ...;` - Columns size to content.
`grid-template-columns: minmax(<min>, <max>) ...;` - Sets min and max column sizes.

Examples:

```
.container {
  grid-template-columns: 1fr 2fr 1fr; /* Three columns with relative widths */
  grid-template-columns: repeat(3, 100px); /* Three 100px columns */
  grid-template-columns: auto 1fr auto; /* Columns adapt to content, with flexible middle column */
}
```

Grid Template Rows

Specifies the number and height of rows in the grid.

`grid-template-rows: <track-size> ...;` - Defines row sizes explicitly.
`grid-template-rows: repeat(<n>, <track-size>);` - Repeats row definitions.
`grid-template-rows: auto ...;` - Rows size to content.
`grid-template-rows: minmax(<min>, <max>) ...;` - Sets min and max row sizes.

Examples:

```
.container {
  grid-template-rows: 100px auto 100px; /* Fixed and flexible row heights */
  grid-template-rows: repeat(2, 1fr); /* Two equal height rows */
}
```

Grid Template Areas

Defines named grid areas, allowing you to place items using names rather than line numbers.

`grid-template-areas: '<area-name> ...' ...;` - Assigns names to grid areas.
`.` - Represents an empty grid cell.

Example:

```
.container {
  grid-template-areas:
    'header header'
    'main   aside'
    'footer footer';
}
```

Gap Properties

Specifies the size of the gap between grid rows and columns.

`grid-gap: <row-gap> <column-gap>;` - Sets both row and column gaps.
`grid-row-gap: <length>;` - Sets only the row gap.
`grid-column-gap: <length>;` - Sets only the column gap.
`gap: <row-gap> <column-gap>;` - Shorthand for `grid-gap`.

Examples:

```
.container {
  gap: 10px; /* 10px gap between rows and columns */
  row-gap: 20px; /* 20px gap between rows */
  column-gap: 15px; /* 15px gap between columns */
}
```

Justify Content

Aligns the grid along the inline (row) axis when the grid size is smaller than the container.

`justify-content: start;` - Aligns to the start of the container.
`justify-content: end;` - Aligns to the end of the container.
`justify-content: center;` - Centers the grid within the container.
`justify-content: stretch;` - Stretches grid items to fill the container.
`justify-content: space-around;` - Distributes space evenly around items.
`justify-content: space-between;` - Distributes space evenly between items.
`justify-content: space-evenly;` - Distributes space evenly around and between items.

Grid Item Properties

Grid Column & Row

Specifies an item's start and end lines/tracks within the grid.

`grid-column-start: <line-number> | <area-name> | span <number> | auto;`
`grid-column-end: <line-number> | <area-name> | span <number> | auto;`
`grid-row-start: <line-number> | <area-name> | span <number> | auto;`
`grid-row-end: <line-number> | <area-name> | span <number> | auto;`
`grid-column: <start-line> / <end-line> | <start-line> / span <value>;` - Shorthand for column.
`grid-row: <start-line> / <end-line> | <start-line> / span <value>;` - Shorthand for row.

Examples:

```
.item {
  grid-column: 1 / 3; /* Item spans from column line 1 to 3 */
  grid-row: 2 / span 2; /* Item spans two rows starting from row line 2 */
}
```

Grid Area

Assigns an item to a named grid area or defines its position based on line numbers.

`grid-area: <area-name> | <row-start> / <column-start> / <row-end> / <column-end>;`

Examples:

```
.item {
  grid-area: header; /* Places item in the 'header' area */
  grid-area: 1 / 1 / 3 / 4; /* Equivalent to setting row and column start/end lines */
}
```

Justify Self

Aligns a grid item along the inline (column) axis within its grid area.

`justify-self: start;` - Aligns item to the start of its area.
`justify-self: end;` - Aligns item to the end of its area.
`justify-self: center;` - Centers item within its area.
`justify-self: stretch;` - Stretches item to fill its area.

Example:

```
.item {
  justify-self: center; /* Centers the item horizontally */
}
```

Align Self

Aligns a grid item along the block (row) axis within its grid area.

`align-self: start;` - Aligns item to the top of its area.
`align-self: end;` - Aligns item to the bottom of its area.
`align-self: center;` - Centers item vertically within its area.
`align-self: stretch;` - Stretches item to fill its area.

Example:

```
.item {
  align-self: end; /* Aligns the item to the bottom */
}
```

Place Self

Shorthand property for setting both `align-self` and `justify-self`.

`place-self: <align-self> <justify-self>;`

Example:

```
.item {
  place-self: center start; /* Centers vertically, aligns to the start horizontally */
}
```

Advanced Grid Concepts

Implicit vs. Explicit Grid

The **explicit grid** is defined using `grid-template-columns` and `grid-template-rows`. The **implicit grid** is created when content is placed outside of the explicit grid.

`grid-auto-rows: <track-size>;` - Defines the size of implicitly created rows.
`grid-auto-columns: <track-size>;` - Defines the size of implicitly created columns.
`grid-auto-flow: row | column | dense;` - Controls how auto-placement algorithm works.

Examples:

```
.container {
  grid-auto-rows: 1fr; /* All implicit rows are the same height */
  grid-auto-flow: dense; /* Tries to fill in gaps in the grid */
}
```

Minmax Function

Defines a size range greater than or equal to min and less than or equal to max. Can be used in `grid-template-columns` and `grid-template-rows`.

`minmax(<min>, <max>)`

Examples:

```
.container {
  grid-template-columns: minmax(100px, 200px) 1fr; /* First column is between 100px and 200px */
}
```

Repeat Function

Repeats a track list multiple times, useful for creating repetitive grid structures.

`repeat(<number | auto-fill | auto-fit>, <track-list>)`

`auto-fill`: Fills the row with as many columns as it can fit.
`auto-fit`: Behaves like auto-fill, but collapses empty tracks.

Examples:

```
.container {
  grid-template-columns: repeat(auto-fit, minmax(150px, 1fr)); /* Creates as many 150px columns as can fit */
}
```

Nesting Grids

A grid item can also be a grid container, creating nested grids. This allows for complex layouts where individual sections have their own grid structure.

Example:

```
<div class="container">
  <div class="item">
    <div class="nested-grid"> ... </div>
  </div>
</div>


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

Practical Grid Examples

Basic Layout Structure

Creating a simple header, content, and footer layout.

HTML:

```
<div class="grid-container">
  <header>Header</header>
  <main>Main Content</main>
  <footer>Footer</footer>
</div>
```

CSS:

```
.grid-container {
  display: grid;
  grid-template-areas:
    'header'
    'main'
    'footer';
  grid-template-rows: auto 1fr auto;
  height: 100vh;
}
header { grid-area: header; }
main   { grid-area: main;   }
footer { grid-area: footer; }
```

Sidebar Layout

Implementing a layout with a main content area and a sidebar.

HTML:

```
<div class="grid-container">
  <main>Main Content</main>
  <aside>Sidebar</aside>
</div>
```

CSS:

```
.grid-container {
  display: grid;
  grid-template-columns: 3fr 1fr;
  gap: 1rem;
}
main { /* styles for main */ }
aside { /* styles for aside */ }
```

Responsive Image Gallery

Creating an image gallery that adapts to different screen sizes.

HTML:

```
<div class="gallery">
  <img src="image1.jpg" alt="">
  <img src="image2.jpg" alt="">
  ...
</div>
```

CSS:

```
.gallery {
  display: grid;
  grid-template-columns: repeat(auto-fit,
minmax(200px, 1fr));
  gap: 1rem;
}
.gallery img {
  width: 100%;
  height: auto;
}
```