CSS Grid Layout: Layout Bidimensional Poderoso

O CSS Grid Layout é um sistema de layout bidimensional, o que significa que ele pode gerenciar colunas e linhas simultaneamente, oferecendo controle incomparável sobre o posicionamento e o dimensionamento dos elementos.
1. Ativando o Grid
Para começar a usar o Grid, você define o elemento pai como um **Contêiner Grid (Grid Container)**.

Propriedade	Exemplo de Declaração	Descrição
`display`	`.container { display: grid; }`	Define o contêiner como um grid de nível de bloco.
`display`	`.container { display: inline-grid; }`	Define um contêiner grid que se comporta como um elemento inline.

---

2. Propriedades do Contêiner Grid

Estas propriedades definem a estrutura das linhas e colunas do grid.

2.1. Definindo a Estrutura do Grid

Controla explicitamente o tamanho e o número de linhas e colunas.

Propriedade	Exemplo de Declaração	Descrição dos Valores
`grid-template-columns`	`.c { grid-template-columns: 100px 1fr auto; }`	Define os tamanhos das colunas (neste caso, 3 colunas: 100px fixo, 1 fração do espaço restante, e tamanho automático).
`grid-template-rows`	`.c { grid-template-rows: 50px 100px; }`	Define os tamanhos das linhas (neste caso, 2 linhas fixas).
`grid-template-areas`	`.c { grid-template-areas: "header header" "nav main" "nav footer"; }`	Define uma estrutura de layout usando nomes de área (requer que os itens filhos tenham a propriedade `grid-area`).
`grid-template`	`.c { grid-template: "a a" 100px "b c" auto / 1fr 2fr; }`	*Shorthand* para rows, columns e areas.

2.2. Fluxo e Alinhamento Automático

Controla como os itens se comportam automaticamente e como o espaço é distribuído.

Propriedade	Exemplo de Declaração	Descrição dos Valores
`grid-auto-flow`	`.c { grid-auto-flow: row dense; }`	Controla como os itens automáticos são colocados. Valores: `row` (padrão), `column`, `dense`.
`grid-auto-columns`	`.c { grid-auto-columns: 100px; }`	Define o tamanho de colunas *implícitas* (criadas automaticamente).
`grid-auto-rows`	`.c { grid-auto-rows: 50px; }`	Define o tamanho de linhas *implícitas* (criadas automaticamente).

2.3. Espaçamento (Gutters)

Define o espaço entre as células do grid.

Propriedade	Exemplo de Declaração	Descrição dos Valores
`column-gap`	`.c { column-gap: 20px; }`	Espaçamento apenas entre colunas.
`row-gap`	`.c { row-gap: 1em; }`	Espaçamento apenas entre linhas.
`gap`	`.c { gap: 10px 20px; }`	*Shorthand* para `row-gap` e `column-gap`. `gap: 10px;` aplica o mesmo valor para ambos.

---

3. Propriedades dos Itens Grid

Estas propriedades são aplicadas diretamente aos elementos filhos para posicioná-los e dimensioná-los dentro do grid.

3.1. Posicionamento e Dimensionamento

Define onde e quanto um item ocupa no grid.

Propriedade	Exemplo de Declaração	Descrição
`grid-column-start`	`.item { grid-column-start: 2; }`	Linha de início da coluna.
`grid-column-end`	`.item { grid-column-end: 4; }`	Linha de término da coluna.
`grid-row-start`	`.item { grid-row-start: 1; }`	Linha de início da linha (row).
`grid-row-end`	`.item { grid-row-end: 3; }`	Linha de término da linha (row).
`grid-column`	`.item { grid-column: 2 / span 2; }`	*Shorthand* (Início / Fim ou Início / Span X células).
`grid-row`	`.item { grid-row: 1 / 3; }`	*Shorthand* (Início / Fim).
`grid-area`	`.item { grid-area: header; }`	Atribui o item a uma área nomeada definida no contêiner.
`grid-area`	`.item { grid-area: 1 / 2 / 3 / 4; }`	*Shorthand* para `row-start / column-start / row-end / column-end`.

3.2. Alinhamento Individual

Controla o alinhamento de um item específico dentro de sua própria célula ou células combinadas.

Propriedade	Exemplo de Declaração	Descrição
`justify-self`	`.item { justify-self: center; }`	Alinha o item no eixo horizontal (inline axis) dentro da célula.
`align-self`	`.item { align-self: end; }`	Alinha o item no eixo vertical (block axis) dentro da célula.
`place-self`	`.item { place-self: center end; }`	*Shorthand* para `align-self justify-self`.

Exemplos de Código Completos

Exemplo 1: **Layout Clássico com Áreas Nomeadas**

Usando `grid-template-areas` para um layout de cabeçalho, navegação, conteúdo e rodapé.

html

```
<div class="layout-areas">
    <header>Header</header>
    <nav>Nav</nav>
    <main>Main Content</main>
    <footer>Footer</footer>
</div>
```

css

```
.layout-areas {
  display: grid;
  grid-template-areas:
    "header header"
    "nav    main"
    "nav    footer";
  grid-template-rows: 80px 1fr 60px; /* Altura do header, conteúdo, footer */
  grid-template-columns: 200px 1fr;  /* Largura da nav, conteúdo principal */
  height: 100vh;
}

header { grid-area: header; background: #f1f1f1; }
nav    { grid-area: nav; background: #ddd; }
main   { grid-area: main; background: #fff; }
footer { grid-area: footer; background: #f1f1f1; }
```

Use o código com cuidado.

Exemplo 2: **Grid Responsivo com `repeat()` e `minmax()`**

Criando uma galeria de cartões que automaticamente se ajustam para ter no mínimo 200px de largura.

html

```
<div class="galeria-grid">
    <div class="card">1</div>
    <div class="card">2</div>
    <div class="card">3</div>
    <div class="card">4</div>
    <div class="card">5</div>
    <div class="card">6</div>
</div>
```

css

```
.galeria-grid {
  display: grid;
  gap: 15px; /* Espaçamento entre os cartões */

  /* magica do grid responsivo: */
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

  /* Explicação da linha acima: */
  /* repeat(): Repete o padrão de colunas automaticamente */
  /* auto-fit: Cria o máximo de colunas possível na largura disponível */
  /* minmax(200px, 1fr): Cada coluna terá no mínimo 200px, e no máximo 1 fração
do espaço restante */
}

.card {
  background-color: lightcoral;
  padding: 40px;
  text-align: center;
}
```