Folhas de Estilo em Cascata: Layouts Básicos

UFJF - DCC202 - Desenvolvimento Web

Prof. Igor Knop igor.knop@ufjf.br

https://bit.ly/3SMc1Kx

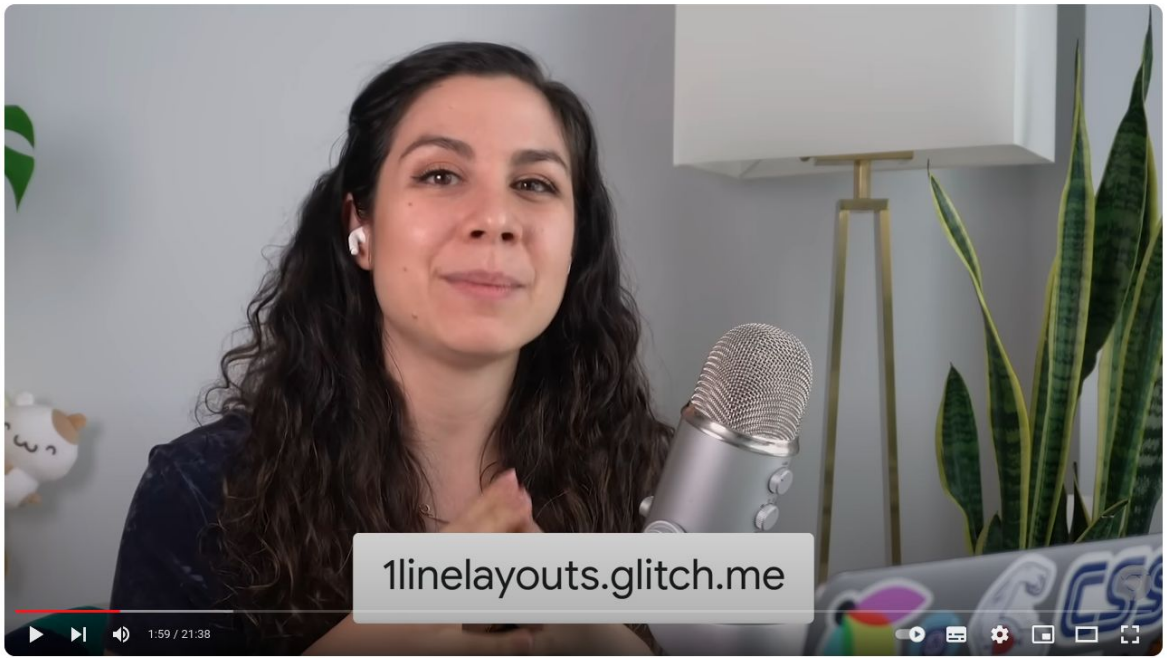
Layouts Básicos

A quantidade de seletores, propriedades valores e comportamentos do CSS podem ser intimidadoras.

Mas várias soluções podem ser aplicadas repetidamente em layouts simples, para problemas comuns.

Aprender essas soluções e associar um nome a elas ajuda a criar um repertório e se acostumar com os funcionamentos do CSS.

O conteúdo aqui apresentado é baseado no vídeo da Una Kravets para o web.dev e canal Chrome for Developers: https://www.youtube.com/watch?v=qm0IfG1GyZU



Documento com uma folha de estilo

Vamos criar uma base para os exemplos, começando com um **layout00.html**.

Esse documento terá apenas um link para o uma folha css na colocaremos os estilos comuns entre as várias páginas.

Para cada um dos demais documentos, faremos uma cópia deste.

```
<!DOCTYPE html>
<html lang="pt-BR">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport"
content="width=device-width,
initial-scale=1.0" />
   <title>DCC202 - Layouts 00</title>
   <link rel="stylesheet" href="layouts.css"
/>
 </head>
 <body>
   <main>
     <h1>DCC202 - Layouts 00</h1>
   </main>
 </body>
</html>
```

Super Centered

Aqui o objetivo é centralizar um único elemento dentro de um contêiner.

Primeiro vamos fazer o nosso main ocupar toda a área da tela, com o `min-height: 100vh`.

Ao mudá-lo para grid, a propriedade `place-items: center;` garante que ele centraliza no eixo principal e cruzado.

Já no article, apenas para testes, colocamos uma borda e a possibilidade de redimensionar direto no navegador.

```
body {
 margin: 0;
 padding: 0;
}

main {
 display: grid;
 place-items: center;
 min-height: 100vh;
}

main > article {
 resize: both;
 overflow: auto;
 border: 1px solid red;
}
```

```
<!DOCTYPE html>
<html lang="pt-BR">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport"
content="width=device-width, initial-scale=1.0"
/>
   <title>DCC202 - Layouts 01: Super
Centered</title>
   <link rel="stylesheet" href="layouts.css" />
 </head>
 <body>
   <main>
     <article>
       <h1>DCC202 - Layouts 01: Super
Centered</h1>
     </article>
   </main>
 </body>
</html>
```

Desconstructed Pancake

O objetivo é deixar elementos alinhados com um tamanho base. Se o espaço disponível fica menor, o layout passa elementos para a linha de baixo, se ajustando.

No elemento container o `display: flex` e o `flex-wrap: wrap` garantem que a linha será quebrada.

Nos itens, a propriedade `flex: 1 1 150px;` tem a ordem *flex-grow flex-shrink flex-base*. Os elementos vão tentar ter 150px, mas podem se esticar ou encolher com o espaço disponível.

```html
<ul class="desc-pancake">
    <li>Lorem, ipsum.</li>
    <li>Accusamus, odio.</li>
    <li>Ea, fugiat?</li>
</ul>
```

```css
.desc-pancake {
   display: flex;
   flex-wrap: wrap;
   justify-content: center;
   list-style: none;
   margin: 0;
   padding: 0;
}

.desc-pancake > * {
   border: 1px solid green;
   background-color: lightgreen;
   flex: 1 1 150px;
   min-height: 80px;
}
```

Sidebars Says

O objetivo é garantir que uma barra lateral tenha uma largura mínima quando encolhida e uma porcentagem máxima quando esticada. O resto do espaço fica para o conteúdo.

O CSS traz algumas funções úteis como a **`minmax()`**. Ela recebe um valor de mínimo e congela se nele caso o valor vá abaixo dele. O segundo argumento é um valor de máximo, que se congela nele se o valor subir.

No exemplo temos um tamanho mínimo absoluto e um valor máximo relativo.

```
<main class="sidebar-says">
 <nav>Menu / Info</nav>
 <article>
   <h1>DCC202 - Layouts 03: Sidebar
Says</h1>
 </article>
</main>

.sidebar-says {
   display: grid;
   grid-template-columns: minmax(100px, 20%)
1fr;
}

.sidebar-says > nav {
   background-color: lightblue;
}

.sidebar-says > article {
   background-color: lightgoldenrodyellow;
}
```

Pancake Stack

O objetivo é dividir o espaço vertical em linhas.

As linhas com o **auto** garantem que vão ter exatamente a altura para seu conteúdo.

Já o restante é dividido entre as frações, que no caso, só tem uma.

```
<main class="pancake-stack">
 <header>
   <h1>DCC202 - Layouts 04: Pancake Stack</h1>
 </header>
 <article>
   <h2>Article</h2>
 </article>
 <footer>
   Footer
 </footer>
</main>

.pancake-stack {
   display: grid;
   grid-template-rows: auto 1fr auto;
}
```

Classic Holy Grail

O objetivo é ter a divisão clássica de layout para um site desktop, com cabeçalho e rodapé e duas colunas laterais.

Isso pode ser alcançado nos itens de grid especificando exatamente em quais colunas eles estarão através das guias.

A guia número 1 é a primeira do grid.

```
.classic-holy-grail {
   display: grid;
   grid-template-columns: auto 1fr auto;
   grid-template-rows: auto 1fr auto;
}
.classic-holy-grail > header {
   grid-column: 1 / 4;
}
.classic-holy-grail > nav {
   grid-column: 1 / 2;
}
.classic-holy-grail > article {
   grid-column: 2 / 3;
}
.classic-holy-grail > footer {
   grid-column: 1 / 4;
}
<main class="classic-holy-grail">
 <header>
   <h1>DCC202 - Layouts 05: Classic Holy
Grail</h1>
 </header>
 <nav><h2>Nav</h2></nav>
 <article>
   <h2>Article</h2>
 </article>
 <aside>  Aside </aside>
 <footer>   Footer </footer>
</main>
```

12-Span Grid

O objetivo é criar um arcabouço comum para layouts mais complexos, que fujam de uma grade com células simétricas.

O espaço é dividido em 12 colunas e um número necessário de linhas ou um divisor de 12. Cada elemento é organizado para se estender nessa grade entre as linhas e colunas.

```html
<ul class="l12-span">
  <li>Lorem.</li>
  <li>Assumenda!</li>
  <li>Veniam.</li>
  <li>Aliquid.</li>
  <li>Laboriosam.</li>
  <li>Dicta.</li>
</ul>
```

```css
.l12-span {
   display: grid;
   gap: 0.2em;
   grid-template-columns: repeat(12, 1fr);
   grid-template-rows: repeat(12, 1fr);
}
.l12-span>li:nth-child(1) {
   grid-column: 1/4;
   grid-row: 1/4;
}
.l12-span>li:nth-child(2) {
   grid-column: 4/10;
   grid-row: 1/8;
}
.l12-span>li:nth-child(3) {
   grid-column: 10/12;
   grid-row: 1/12;
}
.l12-span>li:nth-child(4) {
   grid-column: 1/4;
   grid-row: 1/12;
}
```

RAM (Repeat, Auto, Minmax)

O objetivo é ter uma lista de elementos dispostos na tela, mas com a apenas com a quantidade necessária para ocupar o espaço disponível. Se não for possível manter a largura mínima, uma nova linha é criada.

A opção de **`auto-fit/auto-fill`** para o **`repeat()`** vai garantir que o número seja o mínimo para completar.

```
.ram {
    display: grid;
    gap: 0.1em;
    grid-template-columns: repeat(auto-fit,
minmax(100px, 1fr));
    list-style: none;
    margin: 0;
    padding: 0;
}

.ram>li {
    background-color: lightcyan;
    min-height: 50px;
    display: grid;
    place-items: center;
}
```

Line up cards

O objetivo é ter cartões, que são divididos horizontalmente em linhas. Essas linhas devem ter o mesmo tamanho quando os cartões estão organizados lado a lado.

Um layout como o RAM mantém os cartões lado a lado, mas o cartões em si são exibidos como flexbox em colunas, com espaço entre as caixas.

O `display: flex` com `flex-direction: column;` e `justify-content: space-between;` vão garantir que o os cards usem o espaço vertical todo e alinhe os conteúdos.

```
<dl class="line-up-cards">
 <div>
   <dt>Card 1</dt>
   <dd>Lorem ipsum dolor sit amet.</dd>
   <dd></dd>
 </div>
 <div>
   <dt>Card 2</dt>
   <dd>
     Lorem ipsum dolor sit amet
consectetur, adipisicing elit. Asperiores
     deserunt minima dicta.
   </dd>
   <dd></dd>
 </div>
 <div>
   <dt>Card 3</dt>
   <dd>Lorem, ipsum dolor.</dd>
   <dd></dd>
 </div>
</dl>
```

Line up cards

O objetivo é ter cartões, que são divididos horizontalmente em linhas. Essas linhas devem ter o mesmo tamanho quando os cartões estão organizados lado a lado.

Um layout como o RAM mantém os cartões lado a lado, mas o cartões em si são exibidos como flexbox em colunas, com espaço entre as caixas.

O `display: flex` com `flex-direction: column;` e `justify-content: space-between;` vão garantir que o os cards usem o espaço vertical todo e alinhe os conteúdos.

```
.line-up-cards {
 display: grid;
 grid-template-columns: repeat(auto-fit,
minmax(100px, 1fr));
 gap: 0.2em;
}
.line-up-cards dd:first-of-type {
 font-size: 0.7em;
}
.line-up-cards dd:last-child {
 background-color: lightgoldenrodyellow;
 height: 3em;
}
.line-up-cards > div {
 display: flex;
 flex-direction: column;
 justify-content: space-between;
 padding: 0.5em;
 gap: 0.2em;
}
```

Clamp my style

Respect for Aspect

Para saber mais...

- Yoksel. **FlexBox Cheat Sheet**. 2024. Available on Internet: <https://yoksel.github.io/flex-cheatsheet/>
- Coyier, Chris. **A Complete Guide to Flexbox**. 2024. Available on Internet: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- MDN. **Basic Concepts of Grid Layout**. 2024. Available on internet: <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Basic_concepts_of_grid_layout>