



CSS



Fundamentos de CSS

MSc. Arcanjo Miguel Mota Lopes

CSS Layouts

- É uma das coisas mais básicas que os designers desejam fazer com CSS.
- Não existe uma maneira ‘certa ou direta’ para fazer layouts
- Em meados dos anos 90, os layouts eram construídos usando `<table>`



CSS Layouts

```
<table width="799" border="0" cellspacing="1" cellpadding="1">
  <tr>
    <td bgcolor="#000000">
      <table width="800" height="485" border="0">
        <tr>
          <td height="81" colspan="2" bgcolor="#CCCCCC">
            <table width="100%" border="0">
              <tr>
                <td bgcolor="#FF9966">&nbsp;</td>
                <td bgcolor="#FF9966">&nbsp;</td>
                <td bgcolor="#FF9966">&nbsp;</td>
              </tr>
            </table>
          </td>
        </tr>
        <tr>
          <td width="191" bgcolor="#FFFFFF">&nbsp;</td>
          <td width="599" bgcolor="#FFFFFF">&nbsp;</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

S



CSS Layouts

```
<table width="799" border="0" cellspacing="1" cellpadding="1">
  <tr>
    <td bgcolor="#000000">
      <table width="800" height="485" border="0">

        #CCCCCC">
          <td bgcolor="#FF9966">&nbsp;</td>
          <td bgcolor="#FF9966">&nbsp;</td>
          <td bgcolor="#FF9966">&nbsp;</td>
        </tr>
      </table>
    </td>
  </tr>
  <tr>
    <td width="191" bgcolor="#FFFFFF">&nbsp;</td>
    <td width="599" bgcolor="#FFFFFF">&nbsp;</td>
  </tr>
</table>
```

S

CSS Layouts

--	--

|

```
<td bgcolor="#000000">
```

--

--	--

```
#CCCCC">
```


d>
d>
d>

</table>

</td>

|
 | |

</table>

</td>

</table>

S

CSS Layouts

[illegible]|

```
<td bgcolor="#000000">
```

--

[illegible]

```
#CCCCC">
```

		d>
		d>
		r>

</tr>

|
 | |

</table>

</td>

</table>

CSS Layouts

```
<table width="799" border="0" cellspacing="1" cellpadding="1">
```

```
<tr>
```

```
<td bgcolor="#000000">
```

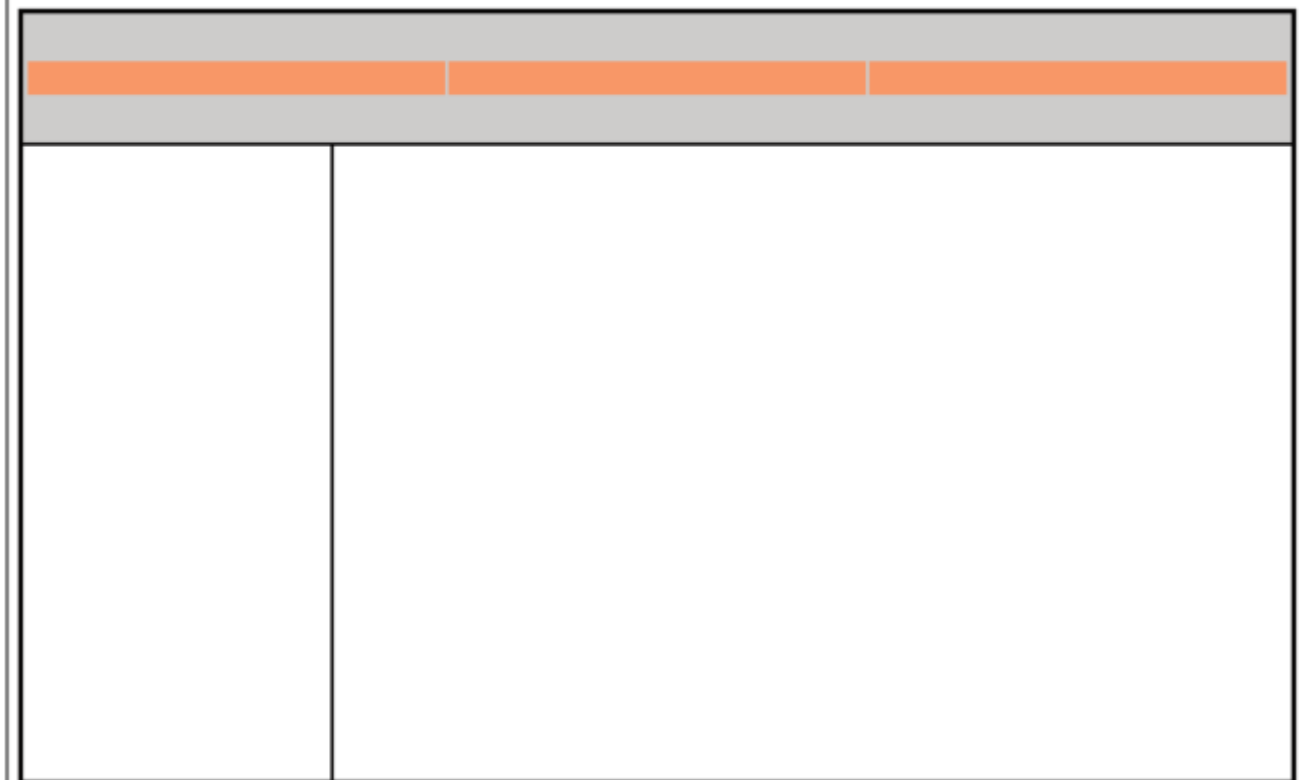
```
<table width="800" height="485" border="0">
```

```
#CCCCCC">
```

```
</td>
```

```
</tr>
```

```
</table>
```



CSS Layouts

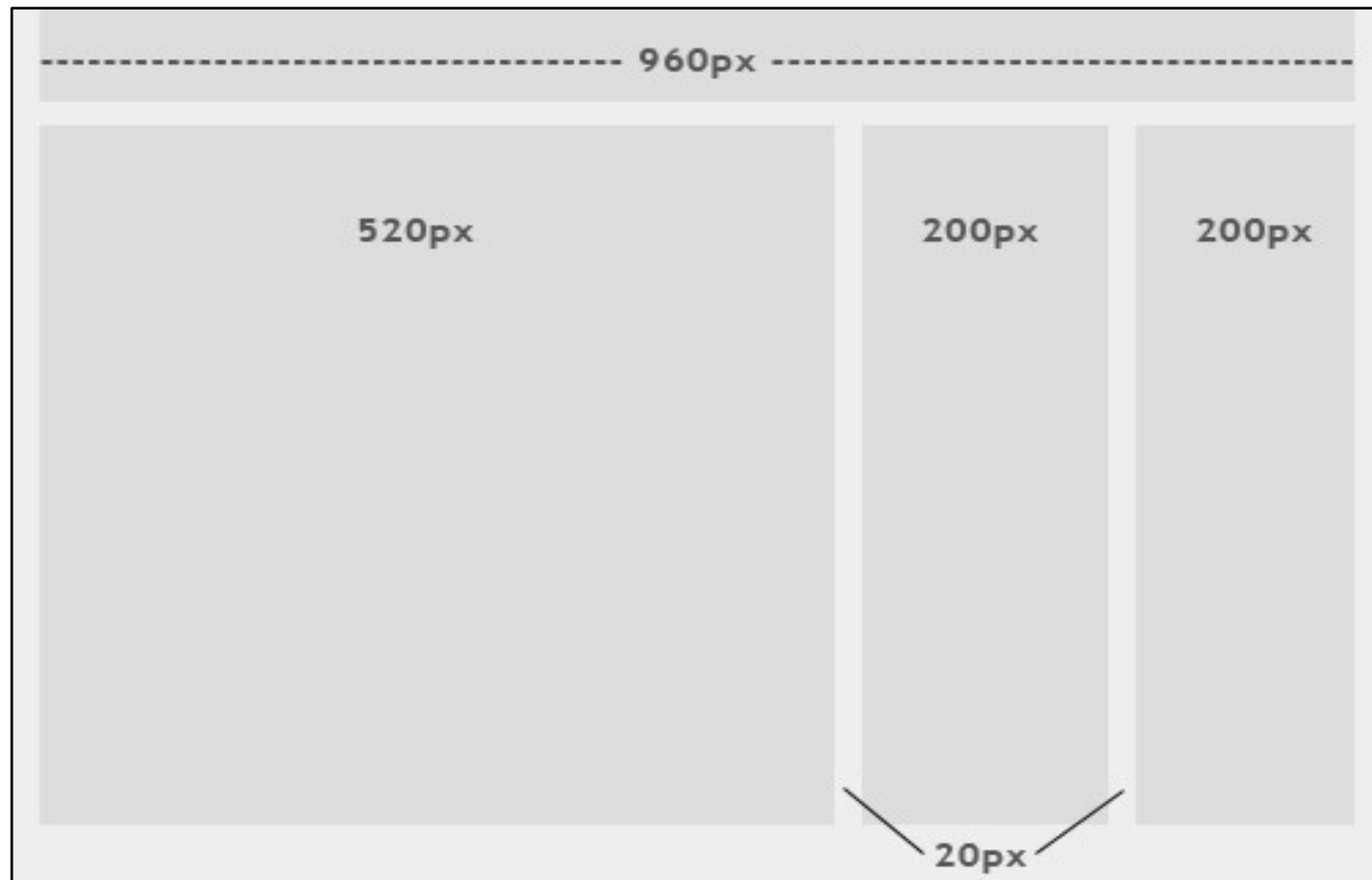
- Layout de largura fixa

- ✓ Todos os elementos da página são aninhados em um contêiner que possui uma largura explícita
- ✓ É útil para o designer porque oferece uma maneira de posicionar de forma confiável os vários elementos do layout (como cabeçalhos, barras laterais e rodapés).
- ✓ Não há necessidade de min-width ou max-width, que não é suportado por todos os navegadores.
- ✓ Mesmo que um site seja projetado para ser compatível com a menor resolução de tela, 800 x 600, o conteúdo ainda será amplo o suficiente em uma resolução maior para ser facilmente legível



CSS Layouts

- Layout de largura fixa



CSS Layouts

[Home](#) [Training Classes](#) [About](#) [Locations](#)





Enhance your creative skills

Web Design Training

- Flash classes
- Dreamweaver classes
- HTML Email classes
- CSS & XHTML classes
- Microsoft Expression Web classes
- Microsoft Expression Blend classes
- Microsoft Silverlight classes
- Private and Customized Training
- Adobe Classes
- Annual Training Pass

[See All Training Classes](#)

Graphics Software & Adobe Training

- Photoshop classes
- Illustrator classes
- InDesign classes
- Acrobat classes
- After Effects classes
- Premiere Pro classes
- QuarkXPress classes
- Adobe Training
- Private and Customized Training
- Training Pass Options

[See All Training Classes](#)

AGI instructors have authored the best-selling book series on creative software, published by Wiley Publishing. When you receive training from AGI, you learn directly from the expert instructors of these books:

- Photoshop CS5 Digital Classroom
- Dreamweaver CS5 Digital Classroom
- Flash CS5 Digital Classroom
- InDesign CS5 Digital Classroom
- Illustrator CS5 Digital Classroom
- Photoshop CS4 Digital Classroom
- Dreamweaver CS4 Digital Classroom
- Flash CS4 Digital Classroom
- InDesign CS4 Digital Classroom
- Illustrator CS4 Digital Classroom
- Photoshop Elements 7 Digital Classroom

m
le
ut



CSS Layouts

- Layout de largura fixa

- ✗ Um layout de largura fixa pode criar espaço em branco excessivo para usuários com resoluções de tela maiores.
- ✗ Resoluções de tela menores podem exigir uma barra de rolagem horizontal, dependendo da largura do layout fixo
- ✗ Texturas, padrões e continuação de imagem são necessários para acomodar aqueles com resoluções maiores.
- ✗ Os layouts de largura fixa geralmente têm uma pontuação geral mais baixa quando se trata de usabilidade



CSS Layouts

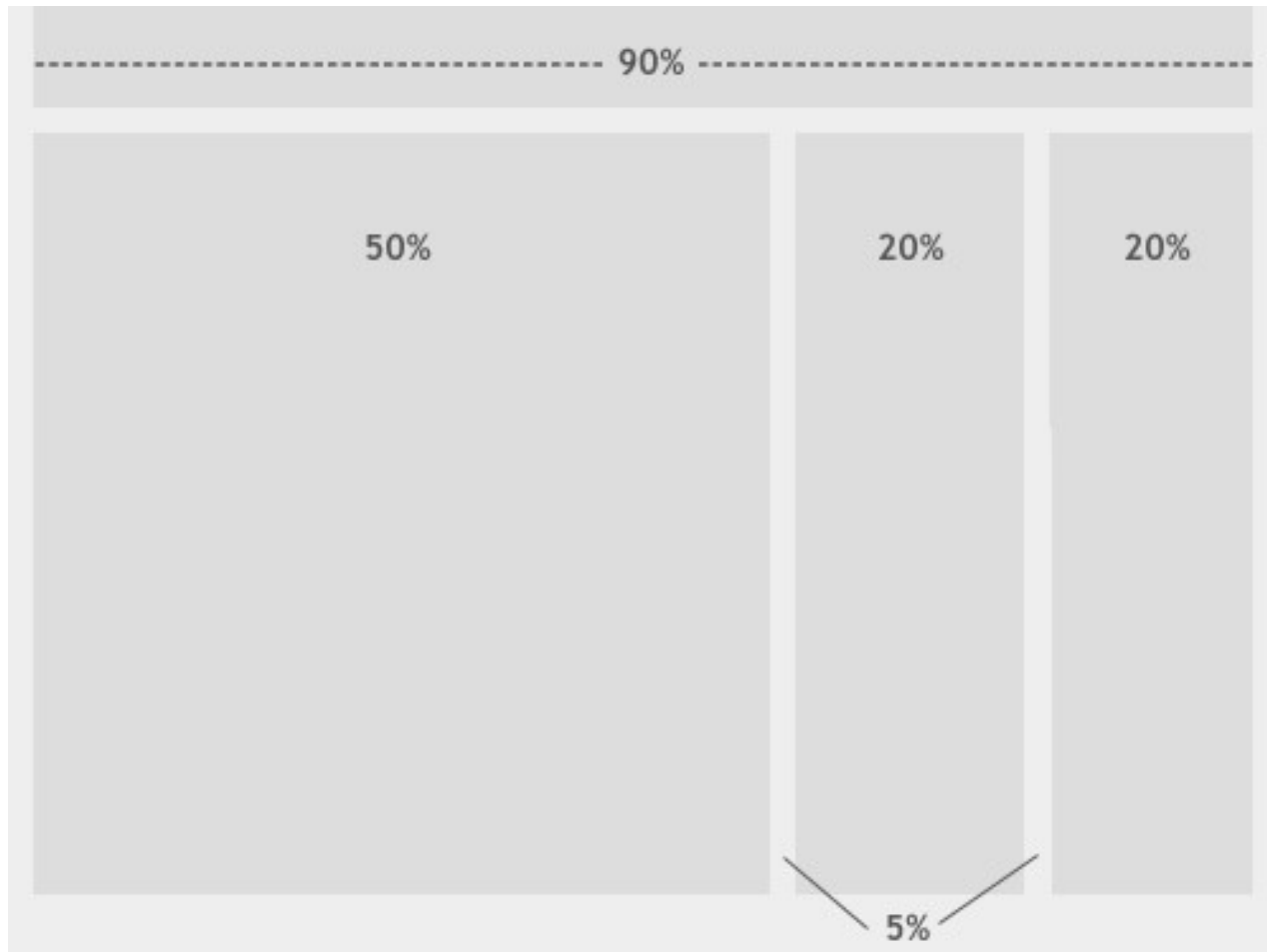
- Layout Flexível

- ✓ São assim chamados porque são projetados para se adaptar à largura da janela do navegador.
- ✓ É útil quando os usuários têm diferentes resoluções de monitor, impossibilitando a criação de um layout de largura fixa que tenha a mesma aparência em todas as telas.
- ✓ Um layout flexível projetado corretamente pode se ajustar automaticamente para caber na janela do navegador do usuário.
- ✓ Se bem projetado, pode eliminar as barras de rolagem horizontais em resoluções de tela menores.



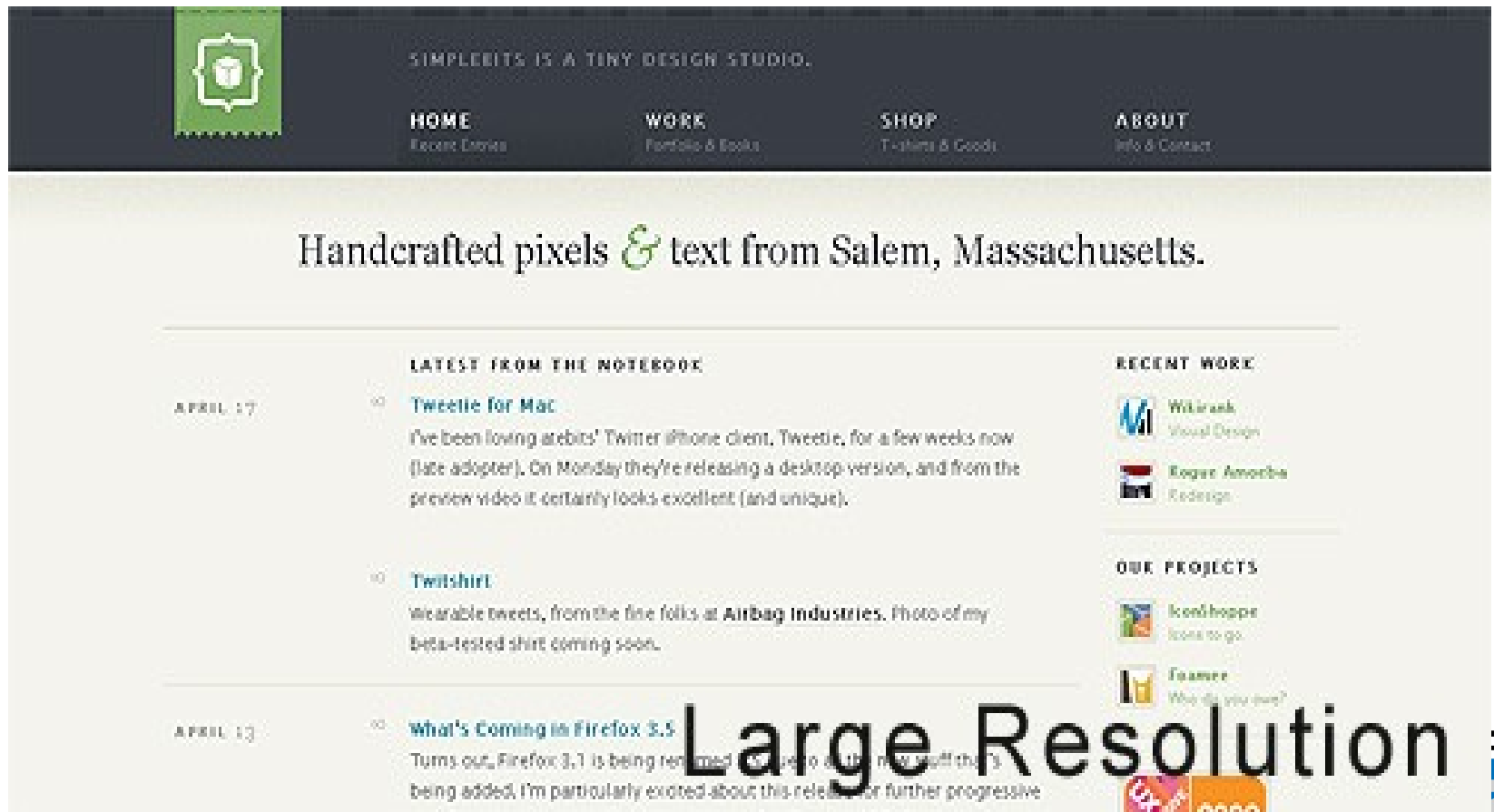
CSS Layouts

- Layout Flexível



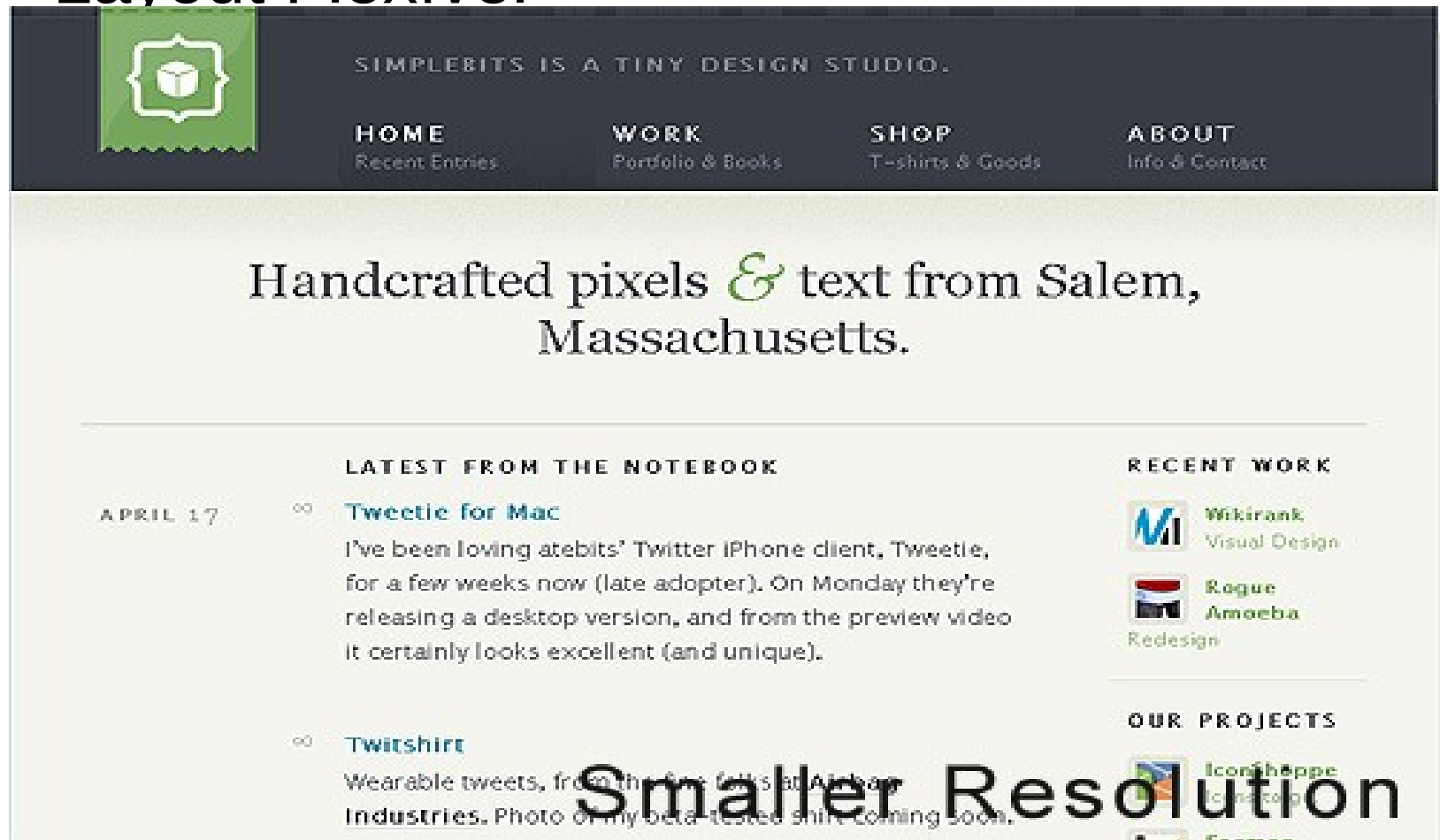
CSS Layouts

- Layout Flexível



CSS Layouts

- Layout Flexível



CSS Layouts

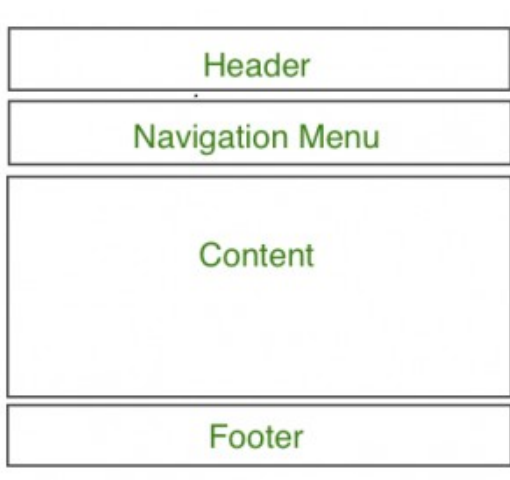
- Layout Flexível

- ✗ O designer tem menos controle sobre o que o usuário vê e pode ignorar os problemas porque o layout parece bom em sua resolução de tela específica.
- ✗ Imagens, vídeos e outros tipos de conteúdo com larguras definidas podem precisar ser configurados em várias larguras para acomodar diferentes resoluções de tela.
- ✗ Com resoluções de tela incrivelmente grandes, a falta de conteúdo pode criar excesso de espaço em branco que pode diminuir o apelo estético.



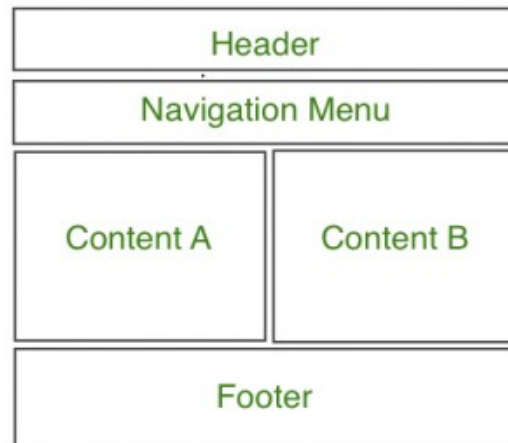
Tipos de Layouts

1 Coluna



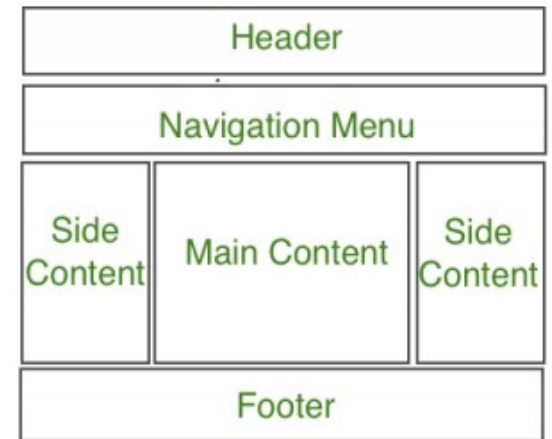
Mobile

2 Coluna



Tablets ou laptops

3 Coluna

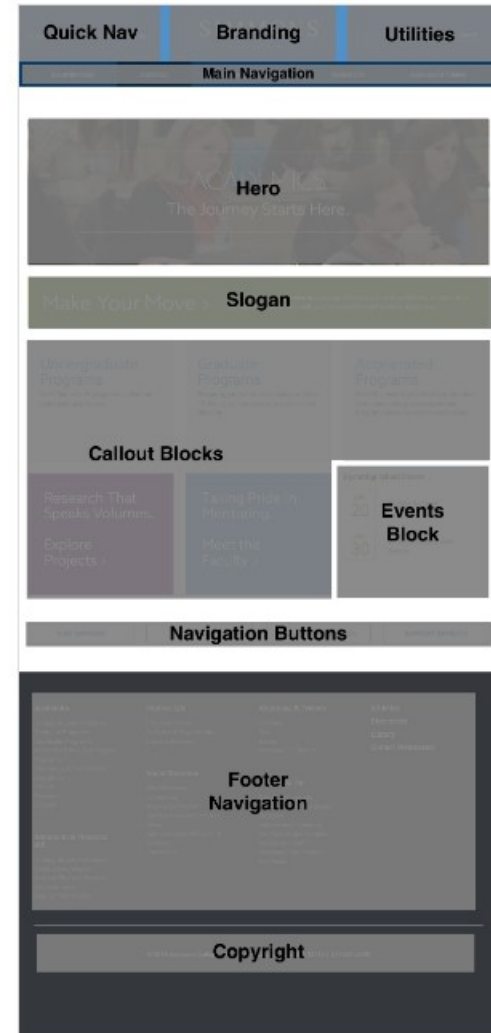
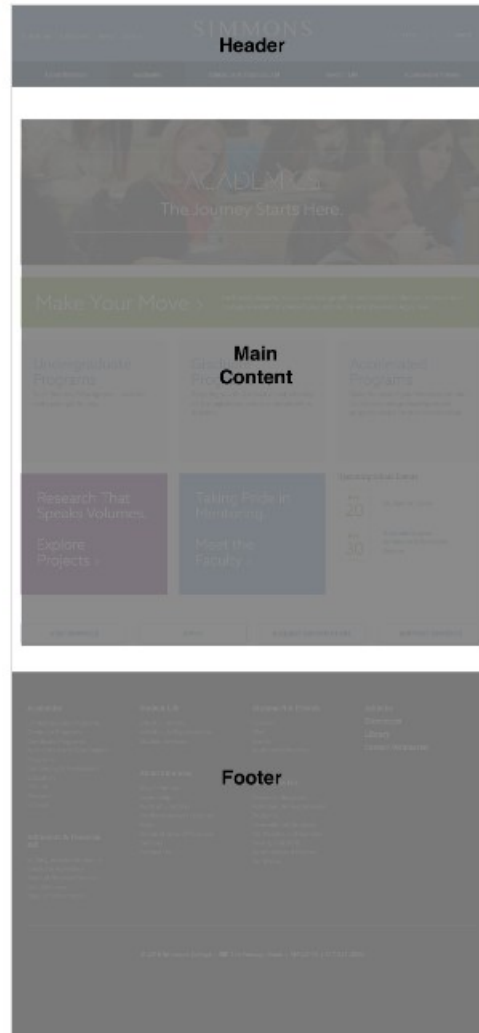
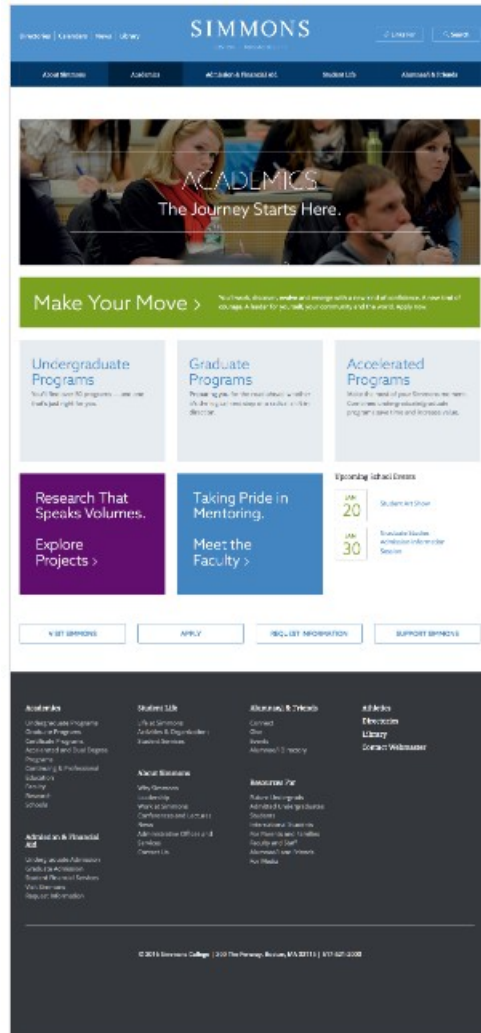


Desktops

Fonte: <https://www.geeksforgeeks.org/css-website-layout/>



CSS Layouts



Fonte: <https://abrir.link/MliOo>



CSS Layouts

- Métodos que podem mudar como os elementos são dispostos:
 - **display**: block | inline | inline-block
 - **float** : left | right
 - **position**: static | fixed | absolute | relative
 - **display**: grid | flex



CSS Layouts

- Métodos que podem mudar como os elementos são dispostos:

→ **display**: block | inline | inline-block

→ **float** : left | right

→ **position**: static | fixed | relative

→ **display**: grid | flex

podem alterar como os elementos se comportam no fluxo normal, por exemplo, fazendo com que um elemento de nível de bloco se comporte como um elemento de nível inline.



CSS Layouts

- Métodos que podem mudar como os elementos são dispostos:

→ **display**: block | inline | inline-block

→ **float** : left | right

→ **position**: static | fixed | absolute | relative

→ **display**: grid | flex

pode fazer com que os elementos em nível de bloco sejam quebrados ao longo de um lado de um elemento,



CSS Layouts

- Métodos que podem mudar como os elementos são dispostos:

→ **display**: block | inline | inline-block

→ **float** : left | right

→ **position**: static | fixed | absolute | relative

→ **display**: grid | flex

permite controlar com precisão a colocação de caixas dentro de outras caixas.



CSS Layouts

- Métodos que podem mudar como os elementos são dispostos:

→ **display**: block | inline | inline-block

→ **float** : left | right

→ **position**: static | fixed | absolute | relative

→ **display**: grid | flex

Métodos de layout completos que são habilitados por meio de valores de exibição específicos, que alteram como os elementos filhos são dispostos dentro de seus pais.

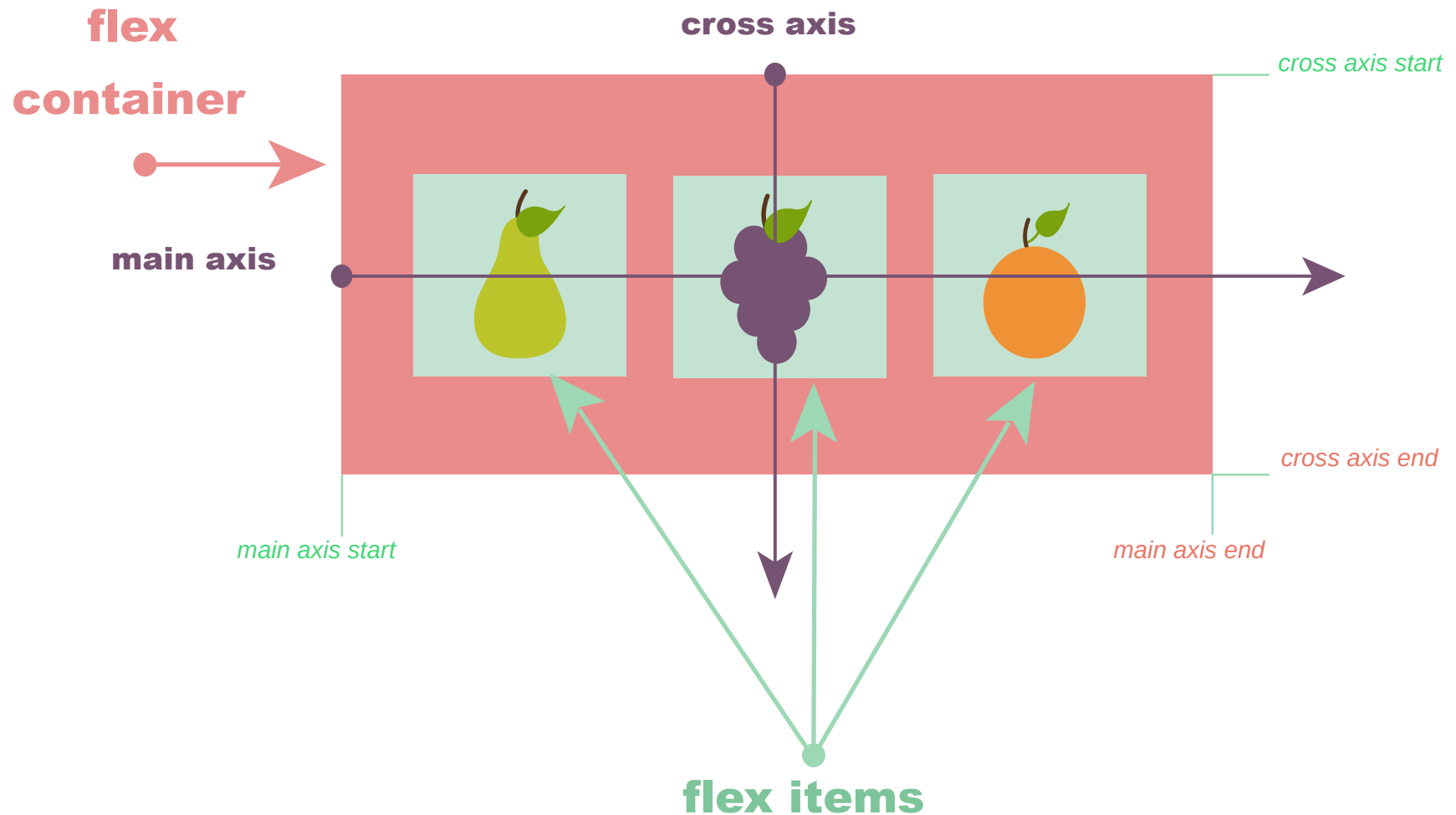


Flex Layout

- Essas propriedades, entre outras coisas, podem alterar a forma como os itens se flexionam, permitindo que eles se expandam ou contraíam de acordo com o espaço disponível.
- O conteúdo do contêiner flexível é chamado de itens flexíveis, que podem se tornar eles próprios contêineres flexíveis.
- Os filhos do flex container devem ser dispostos: da esquerda para a direita, da direita para a esquerda, de cima para baixo ou até mesmo de baixo para cima



Flex Layout



Flex Layout

- Para iniciar com o uso do flexbox é necessário criar um **flex-container**.

Definição do flex layout.

```
<style>
  .flex-container{
    display: flex;
    background-color: dodgerblue;}

  .flex-container > div{
    background-color: #f1f1f1;
    margin: 10px;
    padding: 20px;
    font-size: 30px;}
</style>

<h1>Criando um Flex Container</h1>
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```



Flex Layout

- Para iniciar com o uso do flexbox é necessário criar um **flex-container**.

```
<style>
  .flex-container{
    display: flex;
    backgr

  .flex-cont
    backgr
    margin
    paddin
    font-s
```

Criando um Flex Container



```
</style>

<h1>Criando um Flex Container</h1>
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```



Propriedades do Flex Container

- **display:** define um flex container, habilitando o contexto flex para todos os seus filhos (flex itens)

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

- **flex-direction:** estabelece o main-axis, definindo a direção dos itens dentro de um flex container

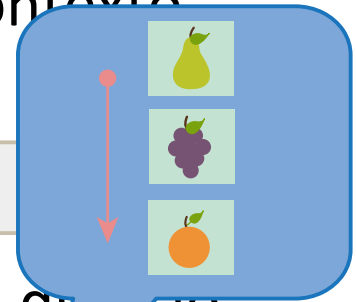
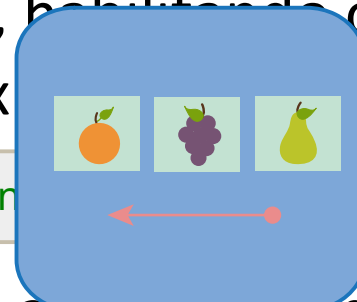
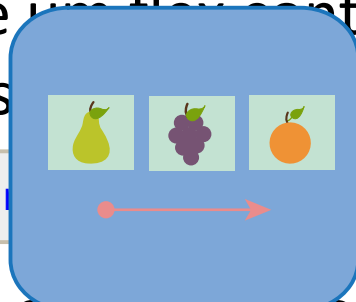
```
.flex-container { flex-direction: row | row-reverse | column |  
column-reverse; }
```



Propriedades do Flex Container

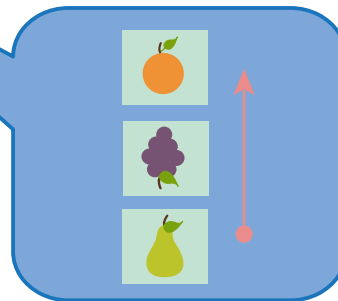
- **display:** define um flex container, habilitando o contexto flex para todos os elementos filhos (flex container)

```
.flex-container { display: flex; }
```



- **flex-direction:** estabelece o main-axis, definindo a direção dos itens dentro de um flex container

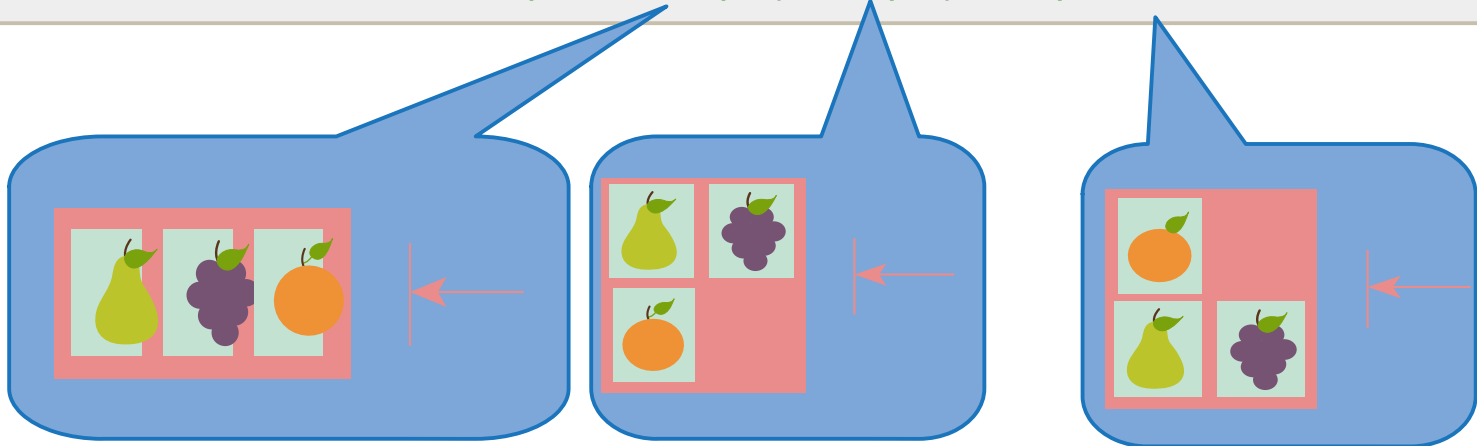
```
.flex-container { flex-direction: row | row-reverse | column | column-reverse; }
```



Propriedades do Flex Container

- **flex-wrap**: define se os itens de um container podem ou não mover para uma nova linha caso não caibam na primeira

```
.flex-container{ flex-wrap: nowrap | wrap | wrap-reverse; }
```



Propriedades do Flex Container

- **justify-content:** define o alinhamento horizontal dos itens quando esses não usam todo o espaço do container

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



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



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



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



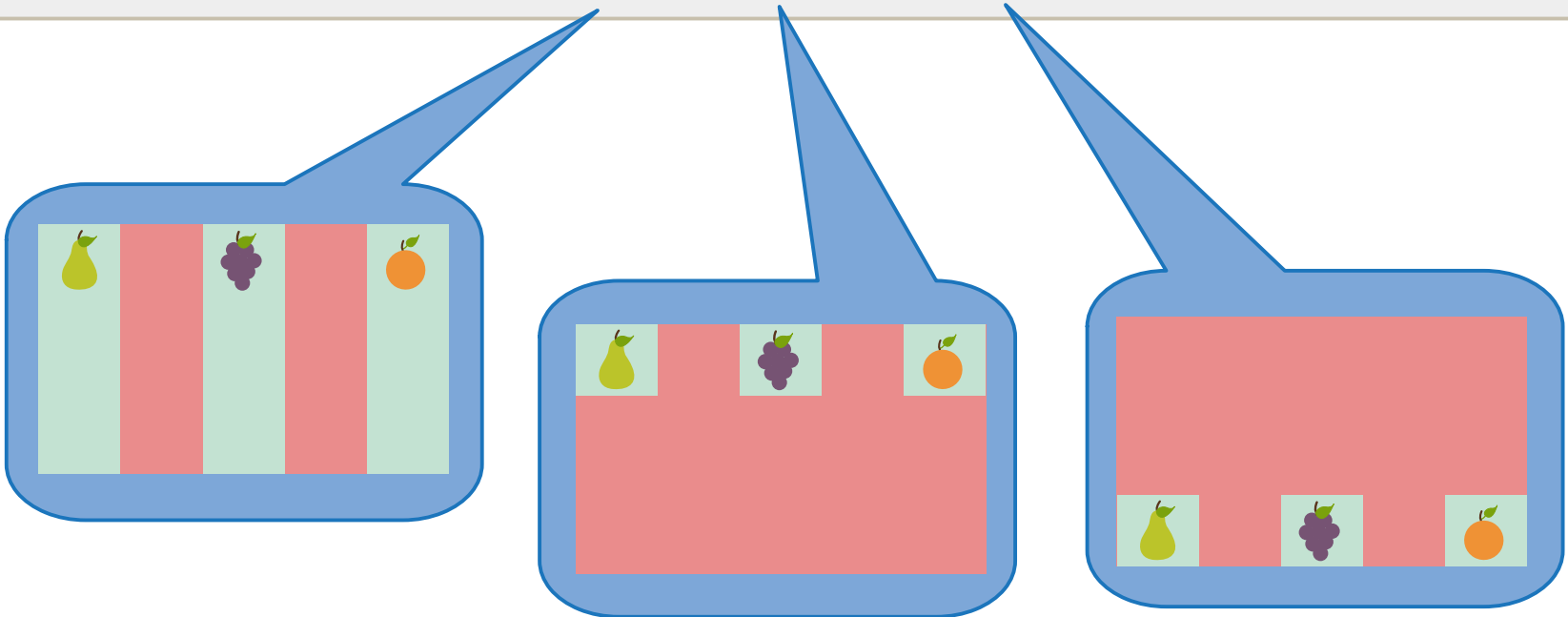
```
.flex-container{ justify-content: space-around;}
```



Propriedades do Flex Container

- **align-items:** Define o alinhamento dos itens no cross-axis. É similar ao justify-content, porém usado no eixo transversal

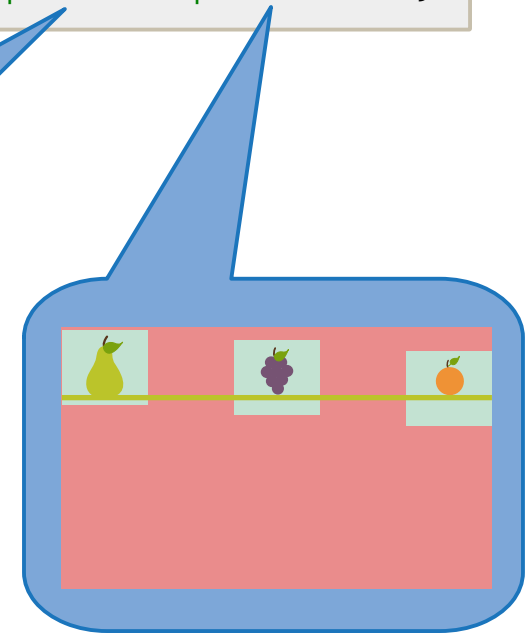
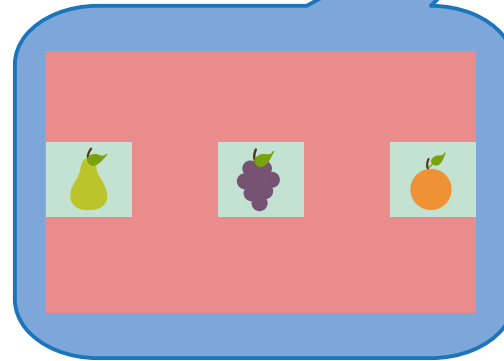
```
.flex-container{ align-items: stretch | flex-start | flex-end | center | baseline }
```



Propriedades do Flex Container

- **align-items:** Define o alinhamento dos itens no cross-axis. É similar ao justify-content, porém usado no eixo transversal

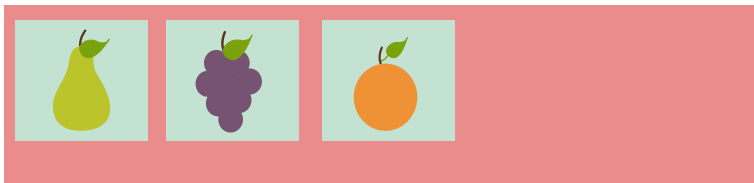
```
.flex-container{ align-items: stretch | flex-start | flex-end | center | baseline }
```



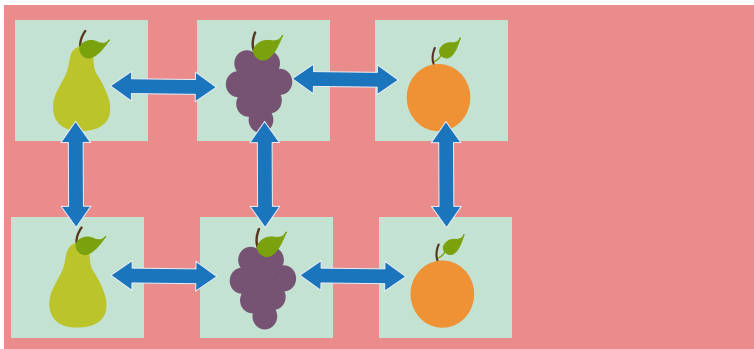
Flex Layout

- gap, row-gap, column-gap

gap: 10px

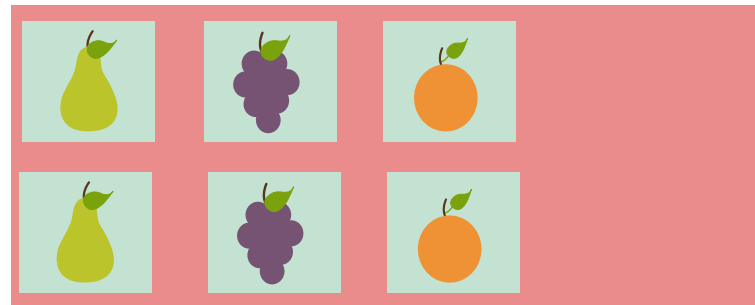


gap: 30px



Atalho para gap
gap: row column.

gap: 10px 30px



Propriedades dos Flex Items

- **flex-grow:** define a proporção com que um item deve crescer, caso seja necessário

```
.item { flex-grow: <número>; /* padrão 0 */ }
```

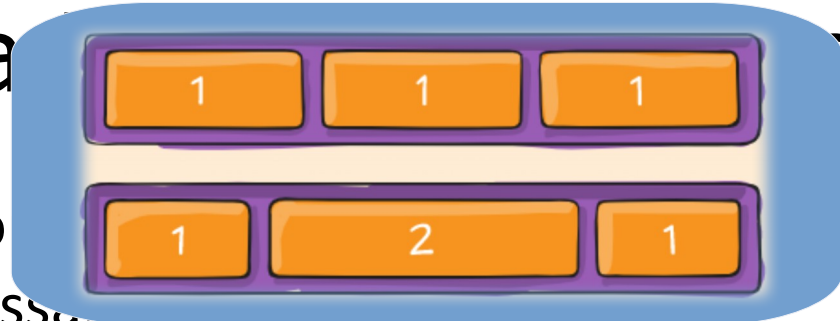
- **flex-shrink:** define a proporção com que um item deve encolher, caso seja necessário

```
.item { flex-shrink: <número>; /* padrão 1 */ }
```

- **flex-basis:** define o tamanho inicial que um item deve ter antes que o espaço ao seu redor seja distribuído

```
.item { flex-basis: <tamanho> | auto; /* padrão auto */ }
```

Propriedades



- **flex-grow:** define a proporção com a qual um item deve crescer, caso seja necessário

```
.item { flex-grow: <número>; /* padrão 0 */ }
```

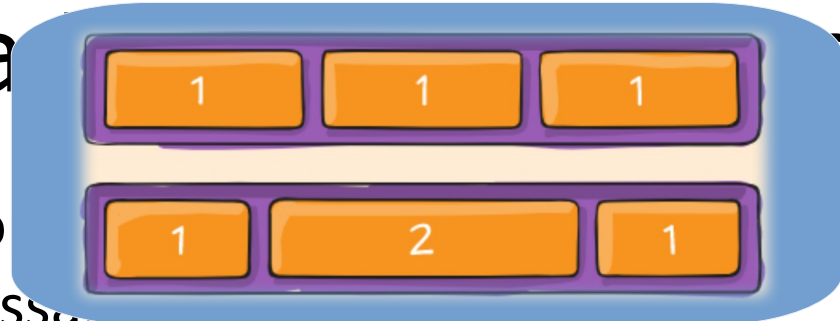
- **flex-shrink:** define a proporção com a qual um item deve encolher, caso seja necessário

```
.item { flex-shrink: <número>; /* padrão 1 */ }
```

- **flex-basis:** define o tamanho inicial que um item deve ter antes que o espaço ao seu redor seja distribuído

```
.item { flex-basis: <tamanho> | auto; /* padrão auto */ }
```

Propriedades



- **flex-grow:** define a proporção com a qual um item deve crescer, caso seja necessário

```
.item { flex-grow: <número>; /* padrão 0 */ }
```

- **flex-shrink:** define a proporção com que um item deve encolher, caso seja necessário

```
.item { flex-shrink: <número>; /* padrão 1 */ }
```

- **flex-basis:** define o tamanho inicial que um item deve ter antes que o espaço ao seu redor seja distribuído

```
.item { flex-basis: <tamanho> | auto; /* padrão auto */ }
```

Propriedades dos Flex Itens

- **flex:** atalho para as propriedades flex-grow, flex-shrink e flex-basis, nesta ordem

```
.item { flex: <flex-grow> <flex-shrink> <flex-basis> }  
/* valor padrão: 0 1 auto */
```

- **order:** modifica a ordem dos flex itens

```
.item { order: <numero> } /* valor padrão: 0 */
```

- **align-self:** define o alinhamento específico de um único flex item dentro do container

```
.item { order: stretch | flex-start | flex-end | center | baseline }  
/* valor padrão: flex-start */
```

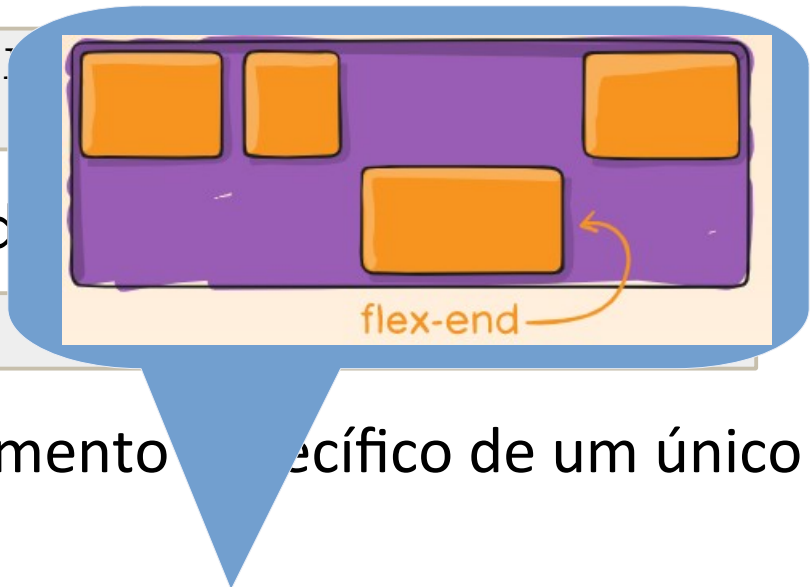
Propriedades dos Flex Itens

- **flex:** atalho para as propriedades flex-grow, flex-shrink e flex-basis, nesta ordem

```
.item { flex: <flex-grow> <flex-shrink> <flex-basis> ;  
/* valor padrão: 0 1 auto */
```

- **order:** modifica a ordem de exibição dos itens

```
.item { order: <numero> } /*
```



- **align-self:** define o alinhamento específico de um único flex item dentro do container

```
.item { order: stretch | flex-start | flex-end | center | baseline }  
/* valor padrão: flex-start */
```

Propriedades dos Flex Itens

```
<style type="text/css">
  html{
    font-family: sans-serif;}

  body{
    width: 70%; max-width: 960px; margin: 20px auto;}

  button{
    font-size: 18px; line-height: 1.5; width: 15%;}

  div {
    height: 100px; border: 1px solid black;}

</style>
<body>
  <div>
    <button> Smile </button>
    <button> Laugh </button>
    <button> Wink </button>
    <button> Shrug </button>
    <button> Blush </button>
  </div>
</body>
```


Propriedades dos Flex Items

```
<style type="text/css">
```

```
html{  
  font-family:
```

```
body{  
  width: 70%
```

```
button{  
  font-size:
```

```
div {  
  height: 10
```

```
</style>
```

```
<body>
```

```
  <div>
```

```
    <button> Smile </button>
```

```
    <button> Laugh </button>
```

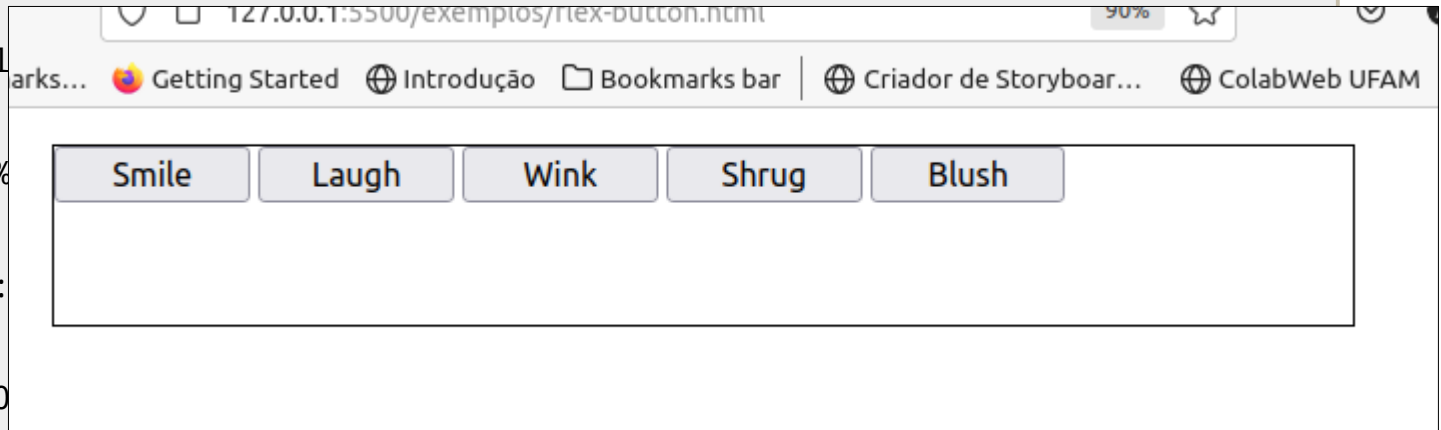
```
    <button> Wink </button>
```

```
    <button> Shrug </button>
```

```
    <button> Blush </button>
```

```
  </div>
```

```
</body>
```



Propriedades dos Flex Itens

```
<style type="text/css">
  html{
    font-family: sans-serif;}

  body{
    width: 70%; max-width: 960px; margin:

  button{
    font-size: 18px; line-height: 1.5; wi

  div {
    height: 100px; border: 1px solid black

</style>
<body>
```

Atividade

- 1) display: flex
- 2) align-items: center
- 3) justify: space-around
- 4) button:first-child align-self in flex-end



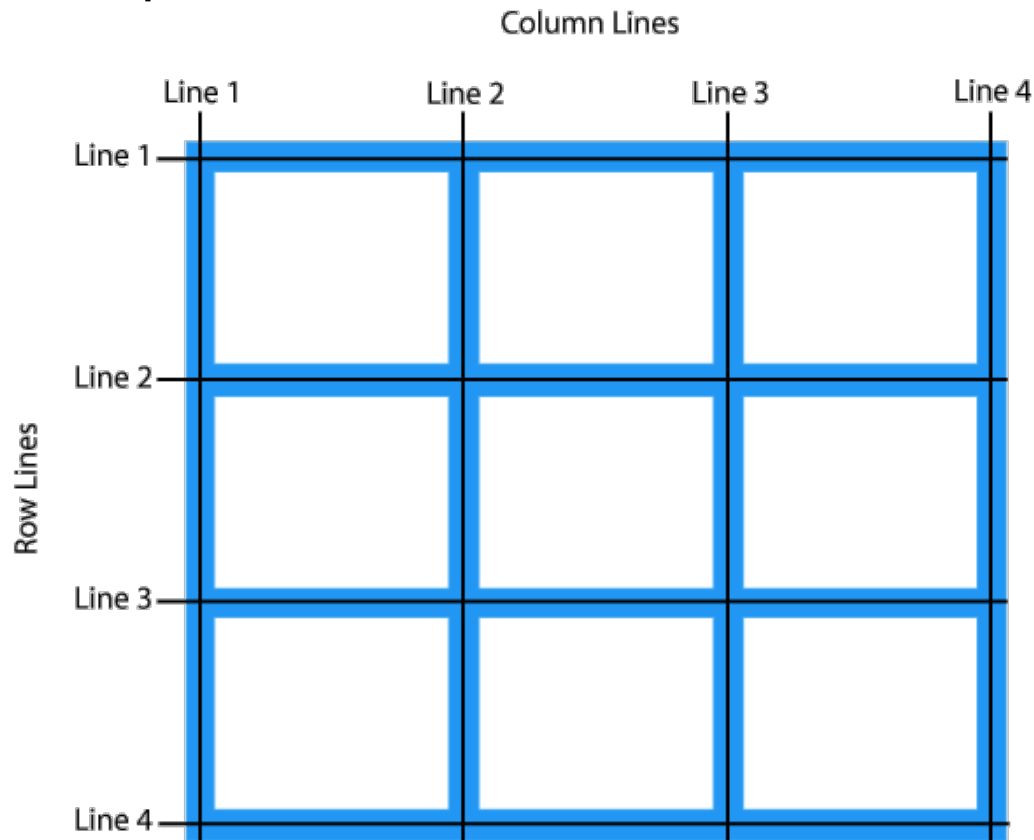
Grid Layout

- É um layout bidimensional para a web.
- Coleção de linhas horizontais e verticais que criam um padrão contra o qual podemos alinhar nossos elementos de design.
- Permite colocar o conteúdo em linhas e colunas.
- Ajudam a criar layouts nos quais nossos elementos não saltam ou mudam de largura.



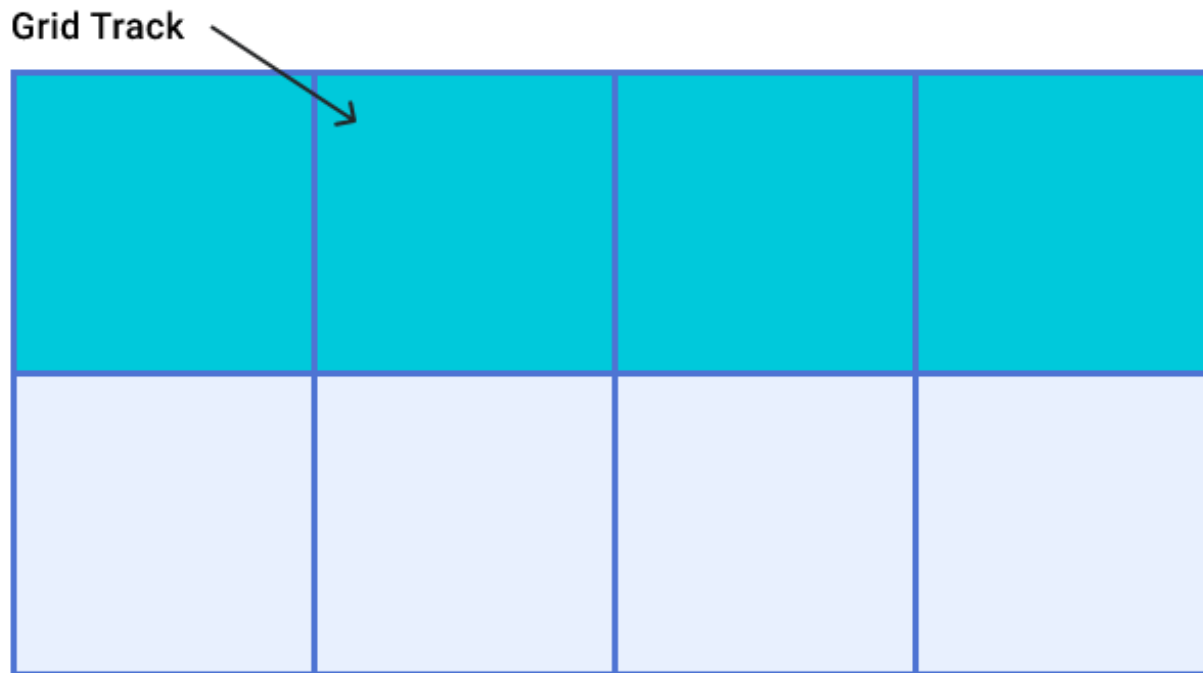
Grid Layout

- **Grid Line:** correm na horizontal e vertical. Se sua grade tiver três colunas, ela terá quatro linhas de coluna, incluindo a que sucede a última.



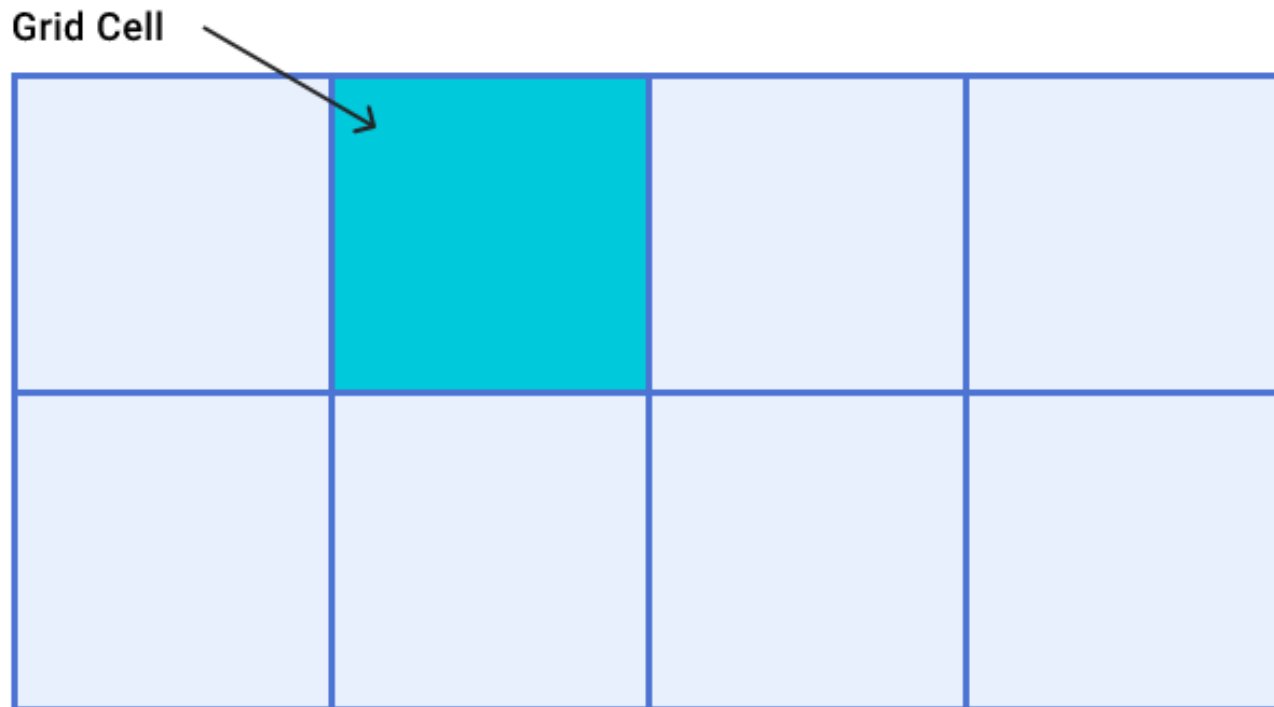
Grid Layout

- **Grid Track:** está entre duas linhas horizontais e uma trilha de coluna está entre duas linhas verticais.



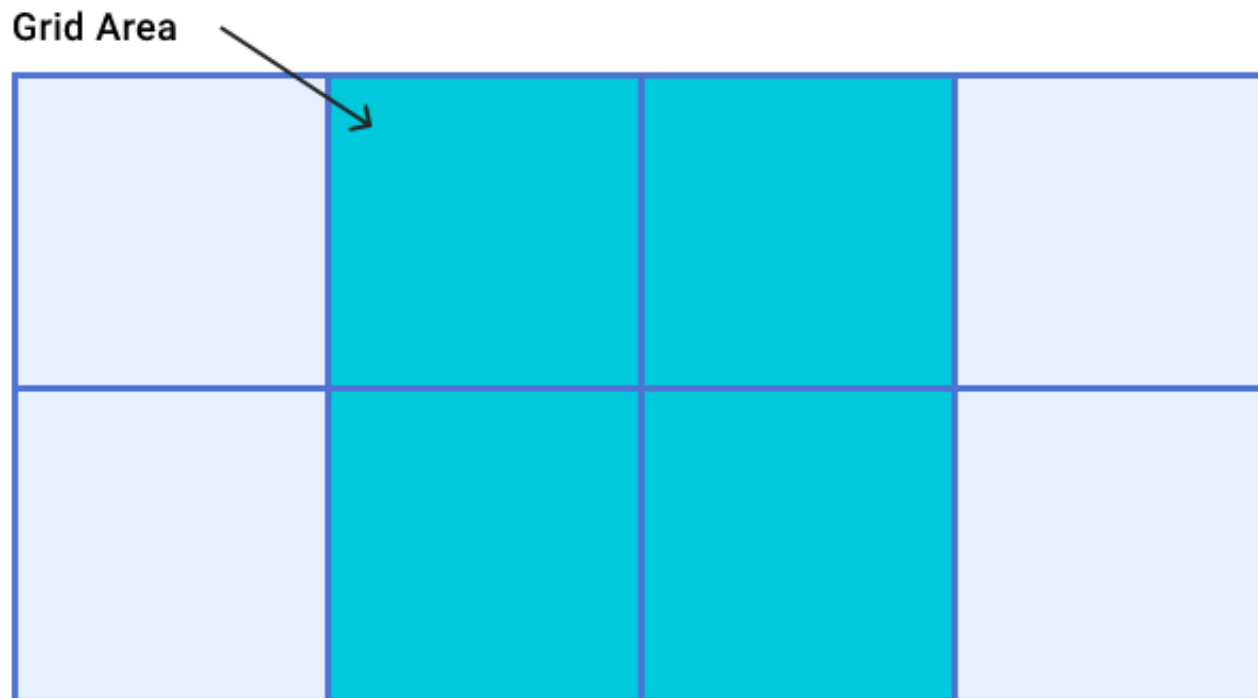
Grid Layout

- **Grid Cell:** é o menor espaço definido pela intersecção de trilhas de linha e coluna. É como a célula de uma tabela ou em uma planilha.



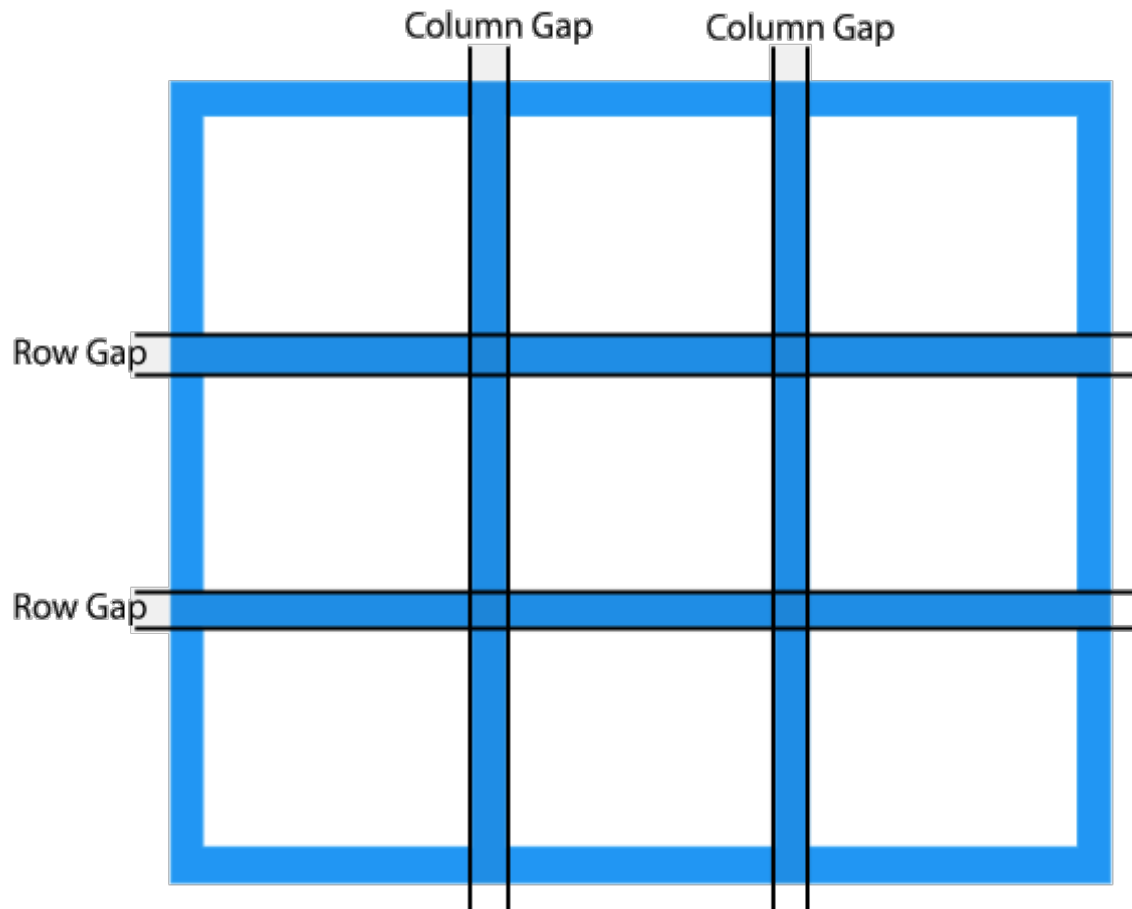
Grid Layout

- **Grid Area:** uma junção de várias **Grid Cell**, é criada ao fazer com que um item ocupe diversas trilhas.



Grid Layout

- **Grid gap:** São os espaçamentos ou separadores entre as trilhas. Para fins de dimensionamento, eles atuam como uma trilha normal.



Grid Layout

- Para iniciar com o uso do Grid Layout é necessário criar um **grid-container**.

```
<style>
  .grid-container{
    display: grid;}
  .item{
    border: 1px solid orangered;
    background-color: orange;
    padding: 10px;}
</style>

<h1>Criando um Grid Container</h1>
<div class="grid-container">
  <div class="item">Um</div>
  <div class="item">Dois</div>
  <div class="item">Três</div>
  <div class="item">Quatro</div>
  <div class="item">Cinco</div>
</div>
```

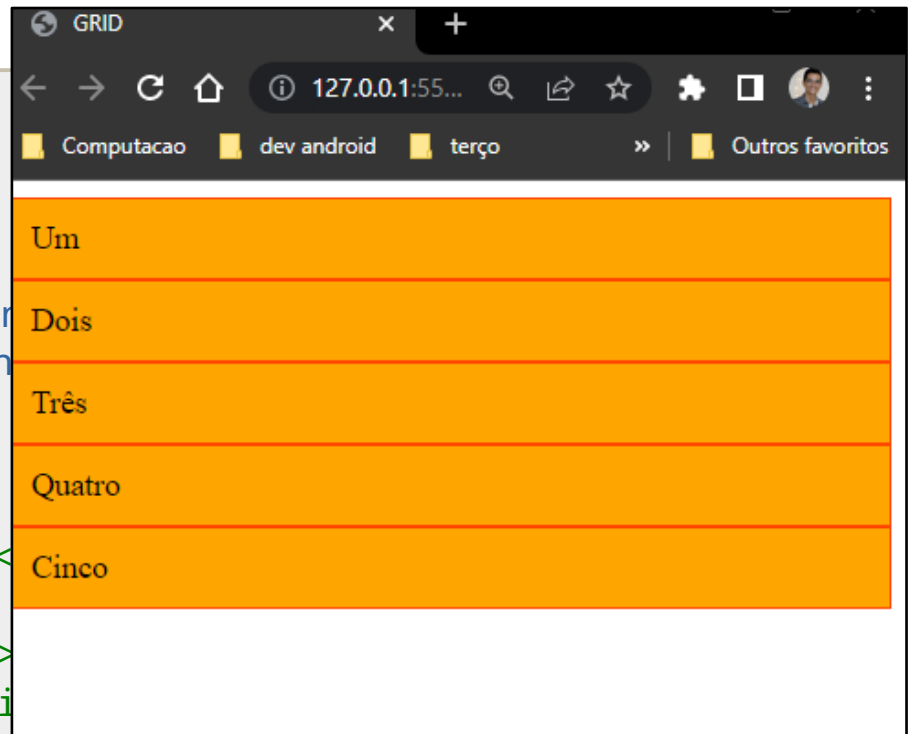


Grid Layout

- Para iniciar com o uso do Grid Layout é necessário criar um **grid-container**.

```
<style>
  .grid-container{
    display: grid;}
  .item{
    border: 1px solid orange;
    background-color: orange;
    padding: 10px;}
</style>

<h1>Criando um Grid Container</h1>
<div class="grid-container">
  <div class="item">Um</div>
  <div class="item">Dois</div>
  <div class="item">Três</div>
  <div class="item">Quatro</div>
  <div class="item">Cinco</div>
</div>
```



Grid Layout

- Para inserir colunas no **grid-container** utilizamos a propriedade **grid-template-columns**.

```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 200px 200px 200px;}
  .item{
    border: 1px solid orangered;
    background-color: orange;
    padding: 10px;}
</style>

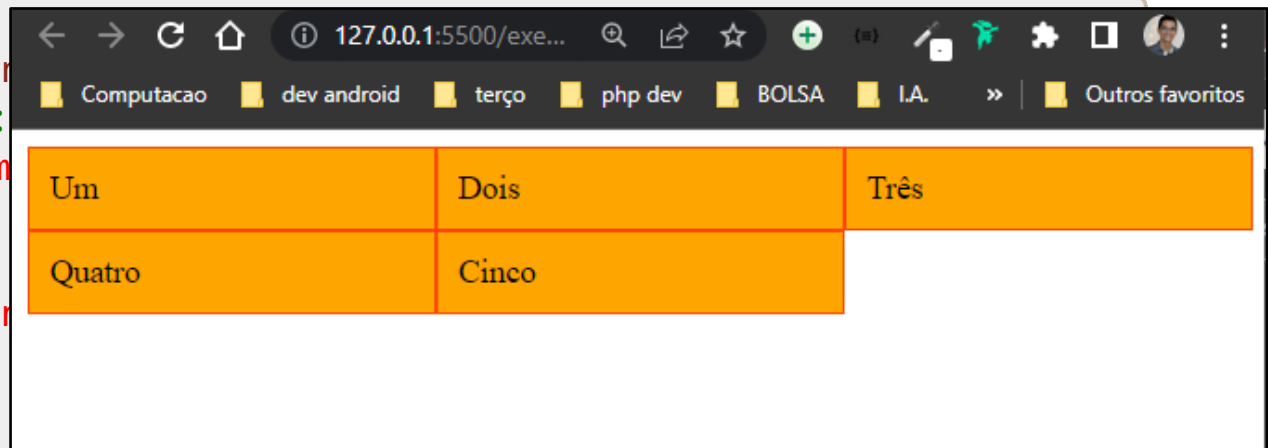
<h1>Criando um Grid Container</h1>
<div class="grid-container">
  <div class="item">Um</div>
  <div class="item">Dois</div>
  <div class="item">Três</div>
  <div class="item">Quatro</div>
  <div class="item">Cinco</div>
</div>
```



Grid Layout

- Para inserir colunas no **grid-container** utilizamos a propriedade **grid-template-columns**.

```
<style>
.grid-container
  display:
  grid-tem
.item{
  border:
  backgrou
  padding:
</style>
```



```
<h1>Criando um Grid Container</h1>
<div class="grid-container">
  <div class="item">Um</div>
  <div class="item">Dois</div>
  <div class="item">Três</div>
  <div class="item">Quatro</div>
  <div class="item">Cinco</div>
</div>
```



Unidade fr

- A unidade fr representa uma fração do espaço disponível no contêiner da Grid para dimensionar com flexibilidade as linhas e colunas.
- Ela distribui o espaço depois que os itens foram dispostos. Portanto, para ter três colunas com a mesma divisão do espaço disponível, faça desta forma:

```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;}
</style>
```



Unidade fr

- A unidade fr representa uma fração do espaço

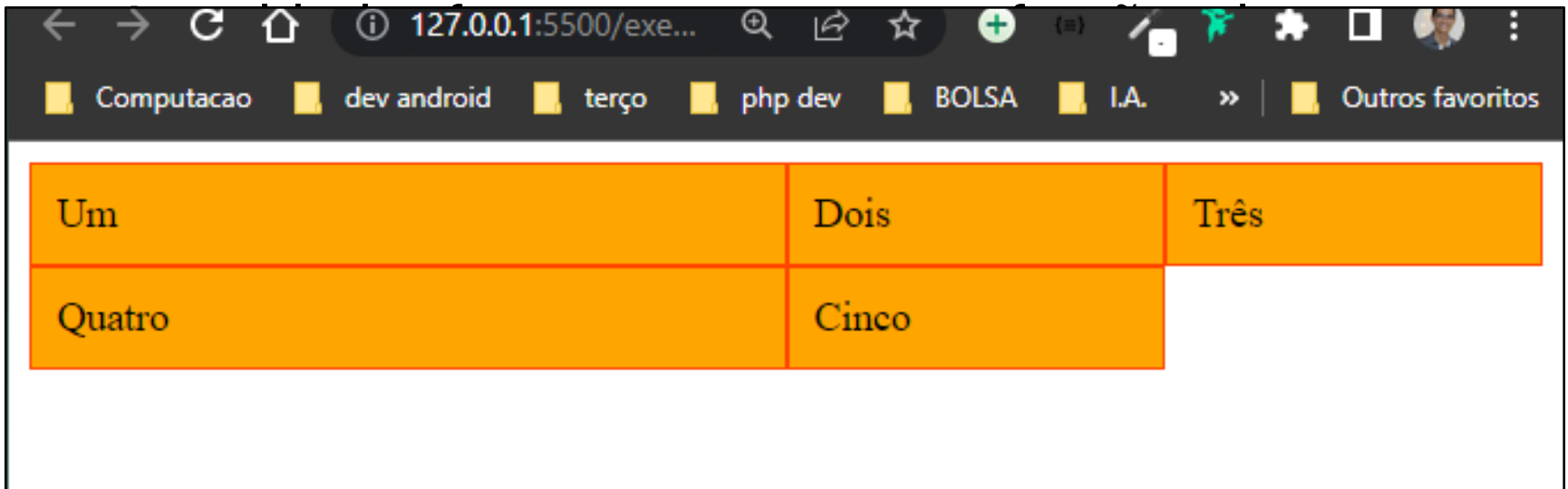


Um Dois Três

Quatro Cinco

```
.grid-container {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr;}  
  
</style>
```

Unidade fr



```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 2fr 1fr 1fr;
  }
</style>
```

Redimensionar uma das colunas do grid.



Notação repeat()

- Pode ser usada para repetir qualquer seção da lista de trilhas. Por exemplo, é possível repetir um padrão de trilhas. Você também pode ter algumas trilhas regulares e uma seção de repetição.

```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 200px repeat(2,1fr 2fr) 200px;}
</style>
```

Primeira coluna tem 200px.

São criadas 4 colunas com dimensões flexíveis.

Última coluna com 200px.



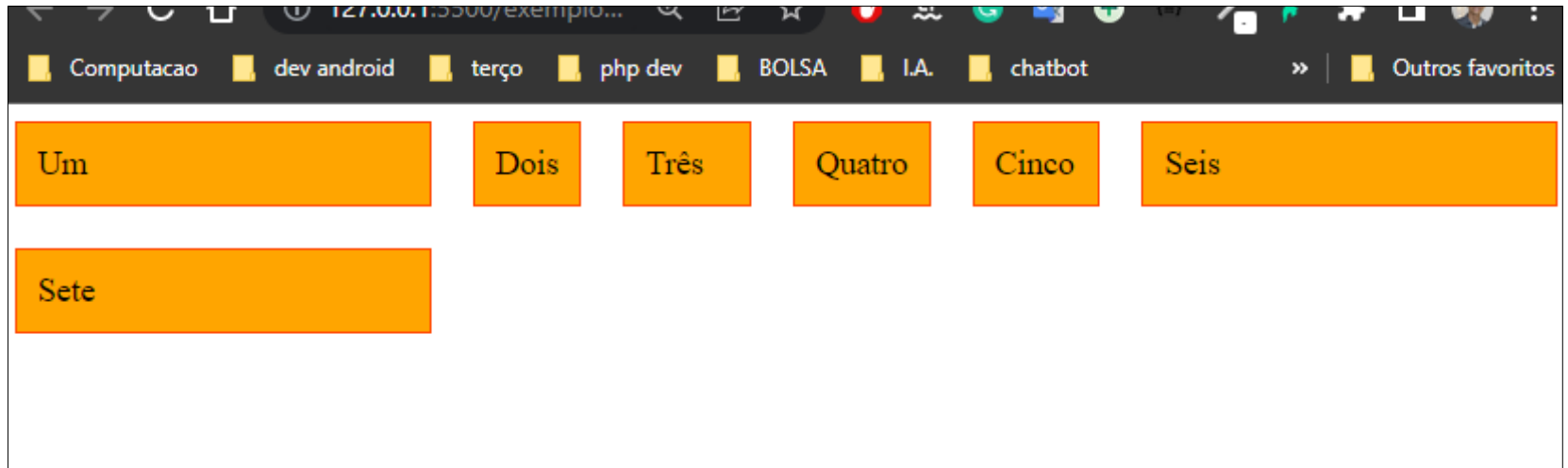
Propriedade Gap

- Para criar 'gaps' entre as linhas, utilize as seguintes propriedades:
- **column-gap:** para criar 'gaps' entre as colunas
- **row-gap:** para criar entre as linhas
- **gap:** é um atalho para as propriedades citadas

```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 200px repeat(2,1fr 2fr) 200px;
    gap: 20px;}
</style>
```



Propriedade Gap



```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 200px repeat(2,1fr 2fr) 200px;
    gap: 20px;}
</style>
```

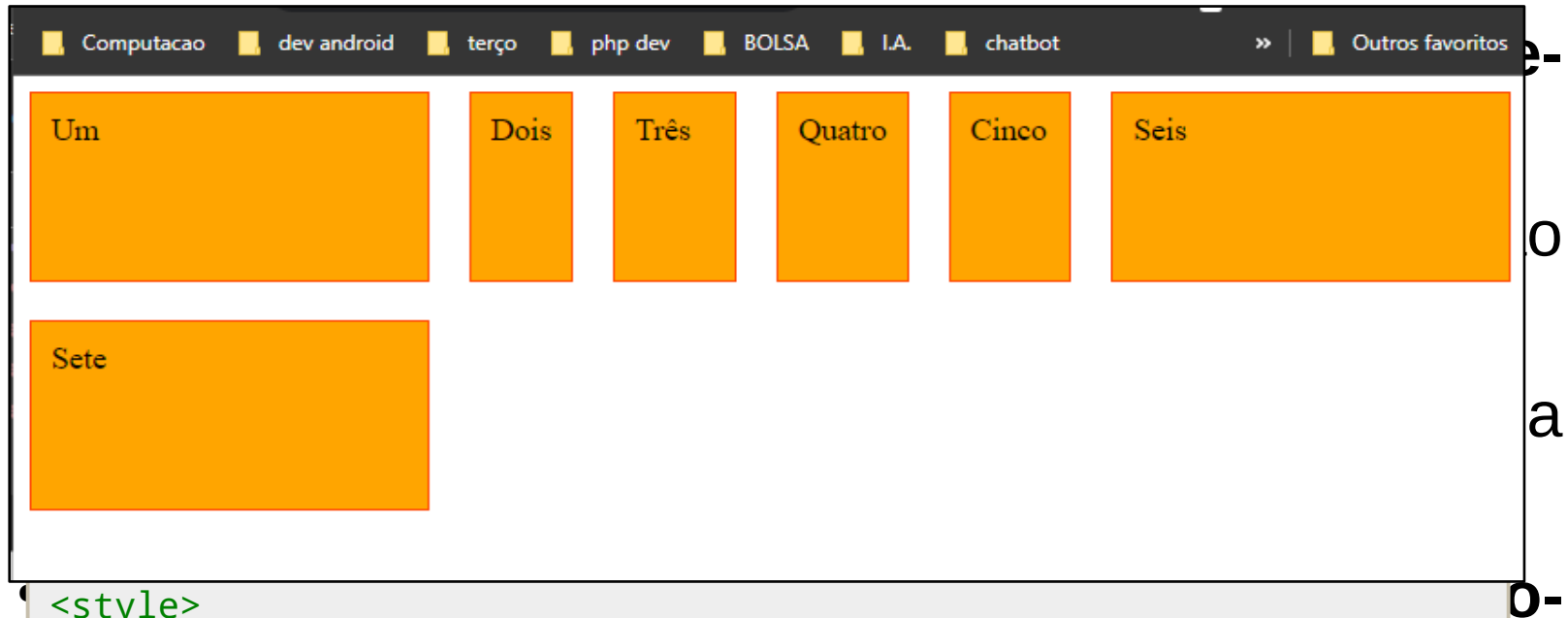


Redimensionando as trilhas do Grid

- Ao configurar um grid utilizamos **grid-template-column** ou **grid-template-row**
- O dimensionamento das linhas e colunas são realizadas automaticamente
- Mas podemos dar as trilhas um tamanho que possa caber qualquer conteúdo adicionado.
- Para isso utilizados **grid-auto-row**; ou **grid-auto-column**



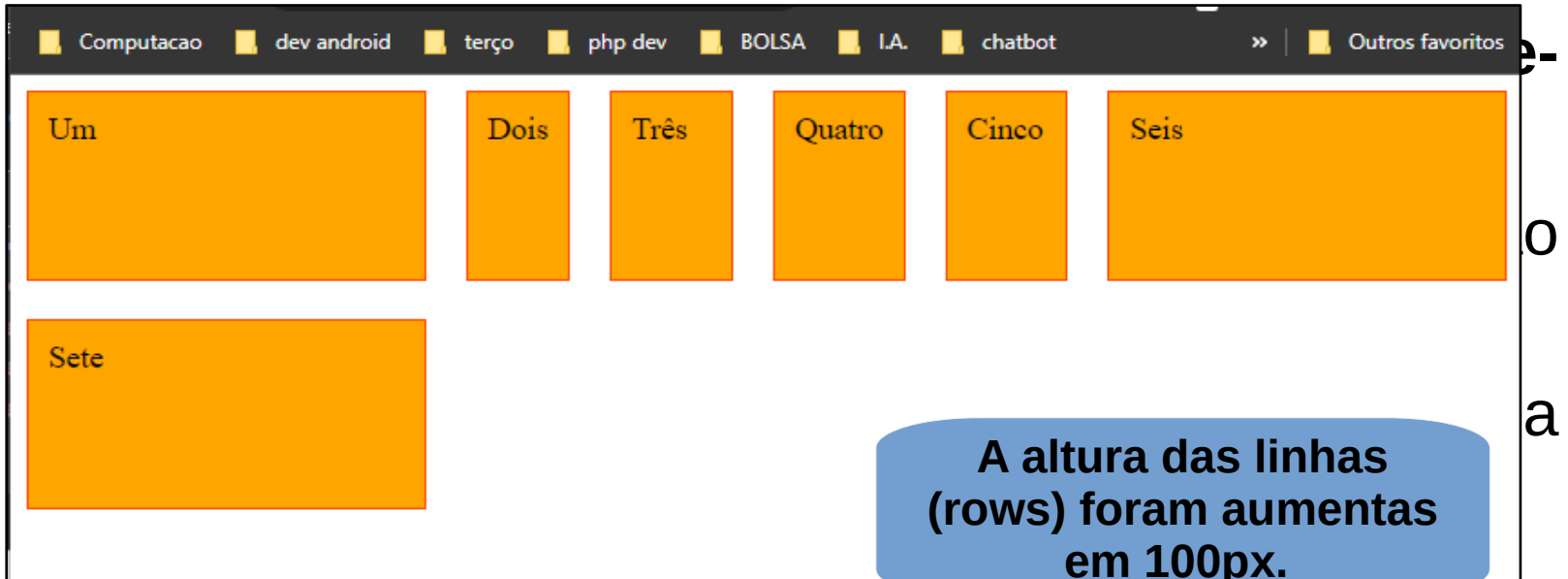
Redimensionando as trilhas do Grid



```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 200px repeat(2,1fr 2fr) 200px;
    gap: 20px;
    grid-auto-row: 100px;}
</style>
```



Redimensionando as trilhas do Grid



Um Dois Três Quatro Cinco Seis

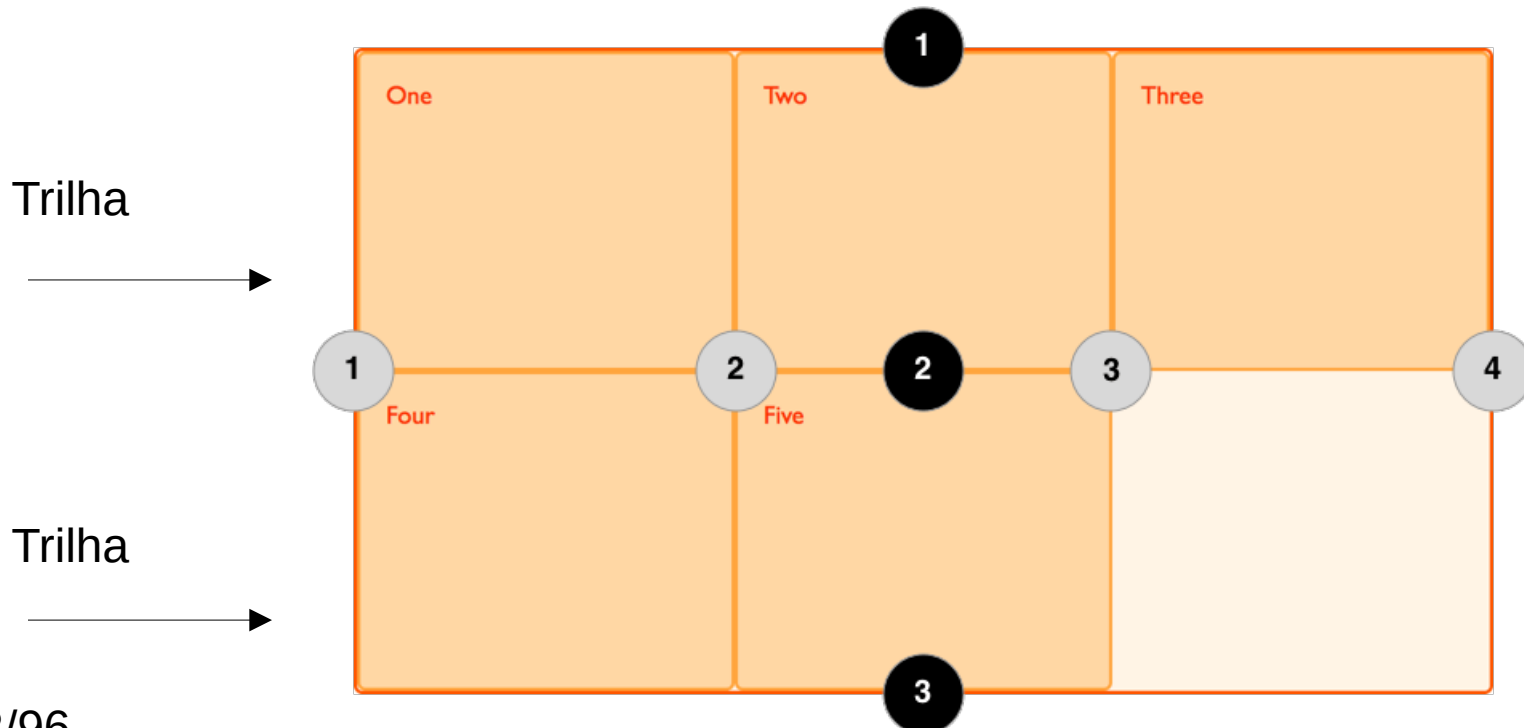
Sete

A altura das linhas (rows) foram aumentas em 100px.

```
<style>
  .grid-container{
    display: grid;
    grid-template-columns: 200px repeat(2,1fr 2fr) 200px;
    gap: 20px;
    grid-auto-row: 100px;}
</style>
```

Posicionamento em Linha

- Quando definimos uma grade, definimos as trilhas da grade, não as linhas
- Grid nos dá linhas numeradas para usar ao posicionar itens



Posicionamento em Linha

- Podemos organizar as coisas de acordo com essas linhas, especificando a linha inicial e final. Fazemos isso usando as seguintes propriedades:
 - `grid-column-start`
 - `grid-column-end`
 - `grid-row-start`
 - `grid-row-end`
- Ou pode utilizar o atalho **`grid-column`** ou **`grid-row`**
- **`Grid-column`** e **`grid-row`** permitem que você especifique as linhas inicial e final de uma só vez, separadas por uma barra



Posicionamento em Linha

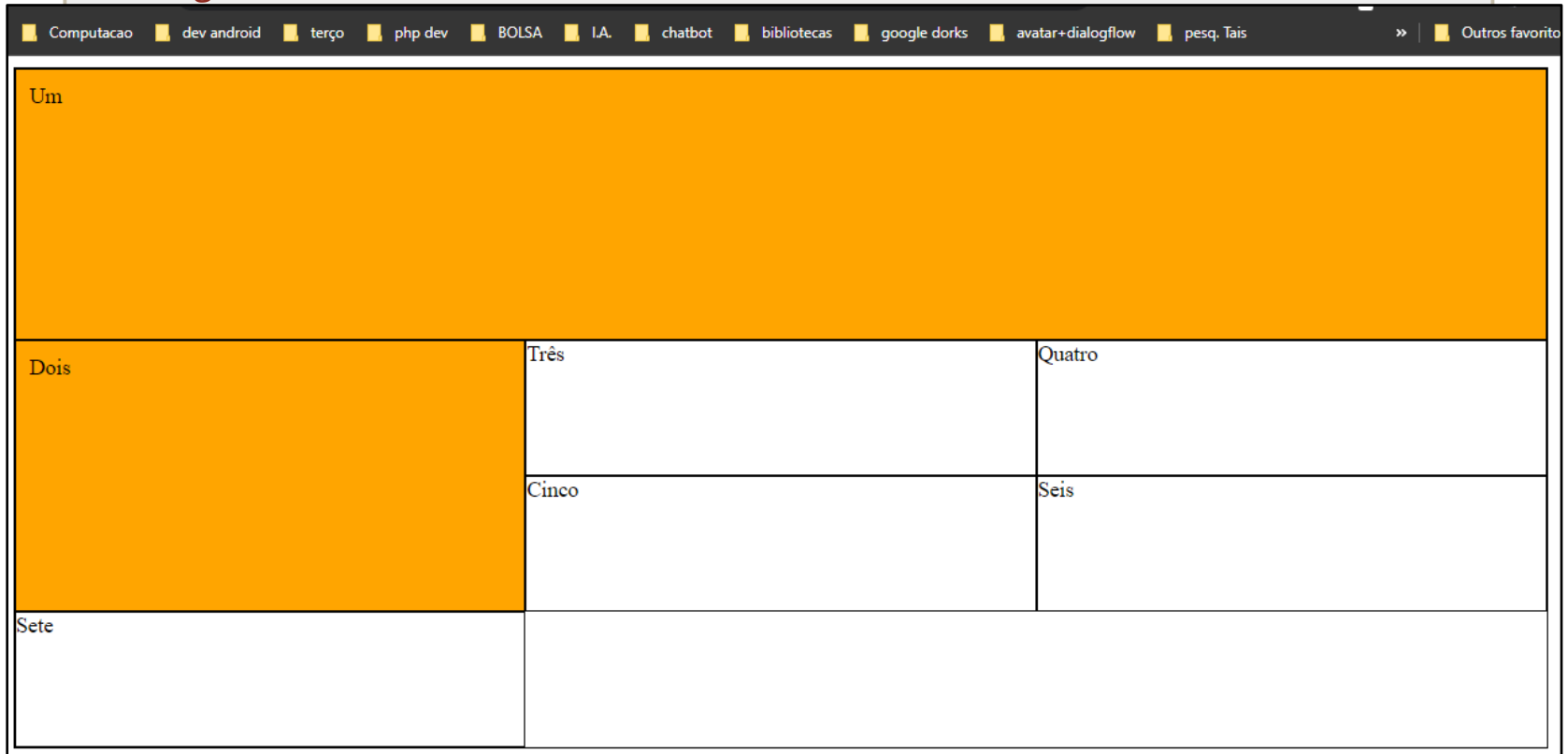
```
<style>
.grid-container{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;}
.item1{
  grid-column-start: 1; grid-column-end: 4;
  grid-row-start: 1; grid-row-end: 3 ;
  background-color: orange;
  padding: 10px;}
.item2{
  grid-column-start: 1;
  grid-row-start: 3; grid-row-end: 5 ;
  background-color: orange;
  padding: 10px;}
</style>
```

```
<h1>Criando um Grid Container</h1>
<div class="grid-container">
  <div class="item">Um</div>
  <div class="item">Dois</div>
  <div class="item">Três</div>
  <div class="item">Quatro</div>
  <div class="item">Cinco</div>
</div>
```



Posicionamento em Linha

```
<style>  
  .grid-container{
```



```
<div class="grid-container">  
  <div class="item">Um</div>  
  <div class="item">Dois</div>  
  <div class="item">Três</div>  
  <div class="item">Quatro</div>  
  <div class="item">Cinco</div>  
</div>
```



Posicionamento Grid

- Dado HTML e CSS faça o que se pede:

```
<style>
    .body{
        width: 90%;
        max-width: 900px;
        margin: 2em auto;
        font: Arial, Helvetica, sans-serif;}
/* Complete o código */
</style>

<div class="grid-container">
    <header> Este é meu site em Grid Layout </header>
    <article>
        <h1> Meu artigo </h1>
        <p> Cole qualquer texto aqui ... </p>
        <p> Cole qualquer texto aqui ... </p>
    </article>
    <aside>
        <h2> Lembretes </h2>
        <p> Cole qualquer texto aqui ... </p>
    </aside>
    <footer> Contato me@meusite.com </footer>
</div>
```



Instruções

- **Grid-container**

- defina o grid layout
- duas colunas flexíveis, 1fr 3fr
- gap 20px

- **Header, footer**

- borda redonda 5px
- padding 10px
- background rgb(207,232,220)
- borda 2px solid rgb(79,185,227)

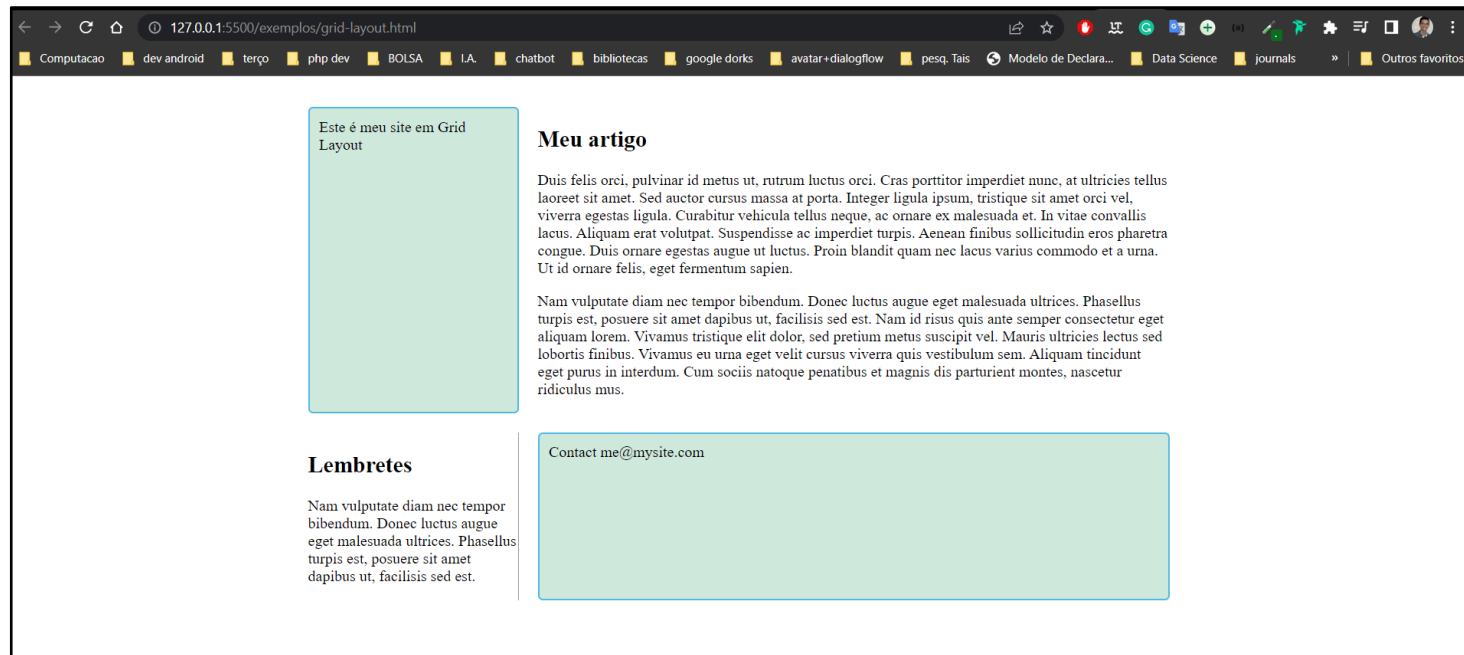
- **aside**

- borda direita 1px solid #999



Instruções

- Resultado esperado

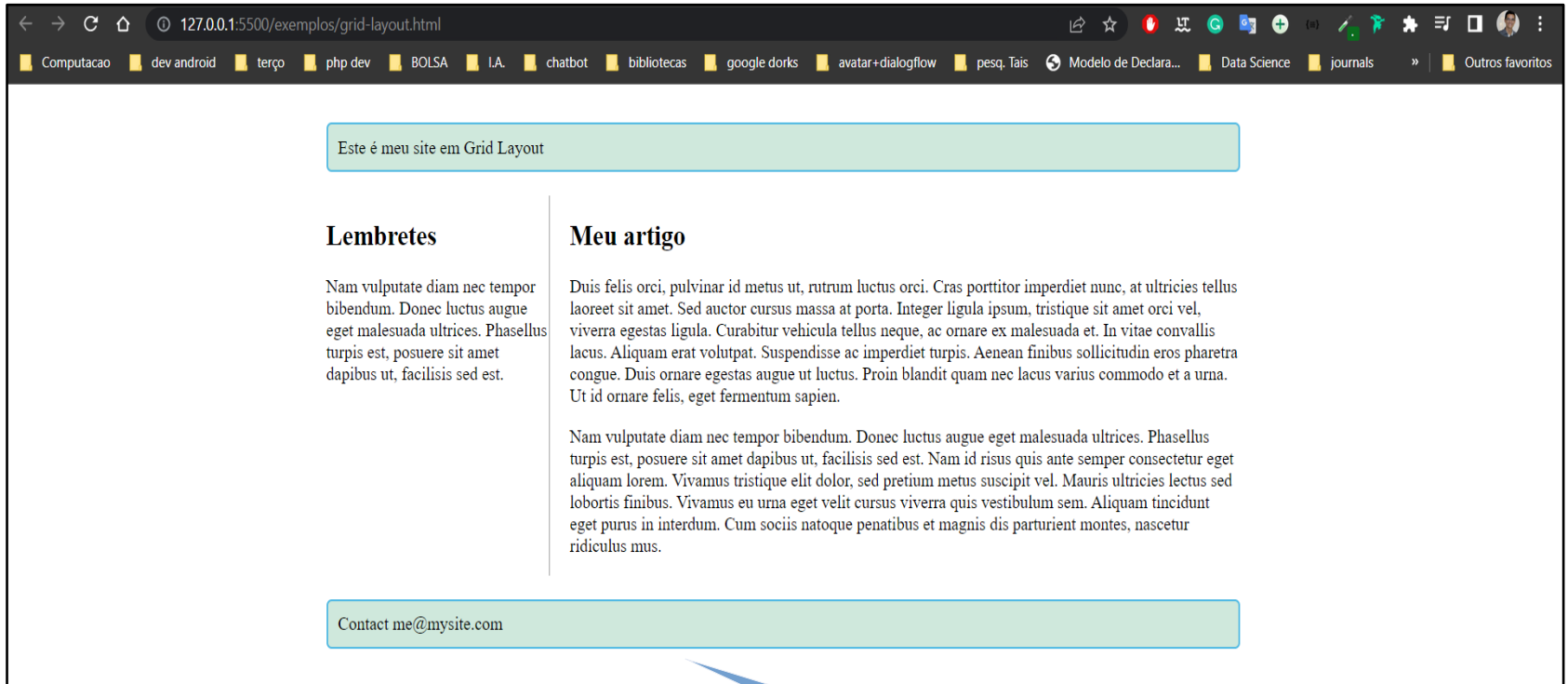


Instruções

- Em seguida utilize as propriedades:
- **grid-column-start**
- **grid-column-end**
- **grid-row-start**
- **grid-row-end**
- E, organize o layout anterior.



Instruções



Resultado esperado.



Diferenças entre Grid e Flexbox

- Grid Layout é um sistema de **layout bidimensional** baseado em grade com linhas e colunas. É útil na criação de layouts mais complexos e organizados.
- O Flexbox é um **layout unidimensional**.
- É útil para alocar e alinhar o espaço entre os itens em um contêiner de grade.
- Facilita o design e a construção de páginas da Web responsivas sem usar muitas propriedades flutuantes e de posição no código CSS



Diferenças entre Grid e Flexbox



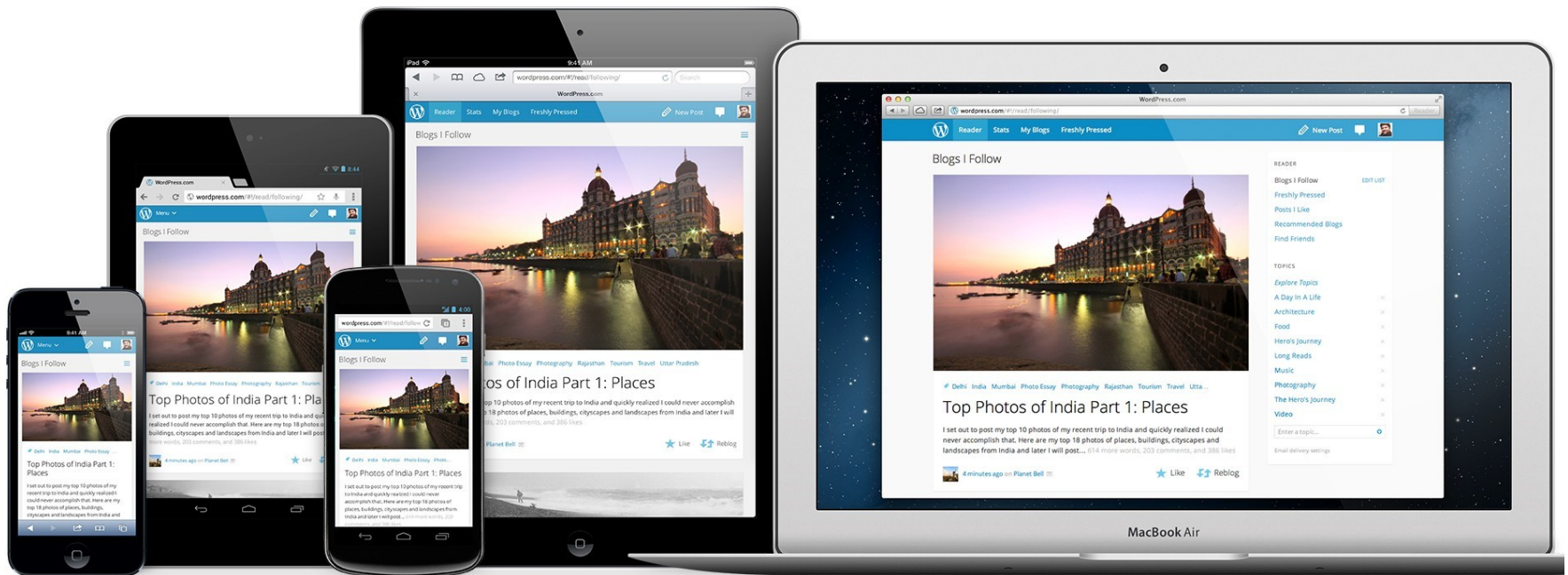
Diferenças entre Grid e Flexbox

- Considerar o uso do Grid Layout, quando:
 - ✓ Você tem um design complexo para trabalhar e deseja páginas da Web sustentáveis
 - ✓ Você deseja adicionar 'gaps' sobre os elementos do bloco
- Considerar o uso do Flexbox Layout, quando:
 - ✓ Você tem um design pequeno para trabalhar com algumas linhas e colunas
 - ✓ Você precisa alinhar o elemento
 - ✓ Você não sabe como seu conteúdo ficará na página e deseja que tudo se encaixe



Design Responsivo

- **Web Design Responsivo** é uma abordagem de web design destinada a elaborar sites que forneçam uma ótima experiência de visualização para uma ampla gama de dispositivos



Media Queries

- **Média Queries** são expressões CSS utilizadas para mudar o layout do site de acordo com o tipo do dispositivo usado para acessá-lo
- No CSS2 existia uma função chamada **Media Type**, usada para reconhecer o tipo de dispositivo
- Ao todo, eram 10 diferentes Media Types:
 - **all** – todos os dispositivos
 - **aural** – sintetizadores de voz
 - **handheld** – celulares e outros dispositivos de mão
 - **print** – impressoras convencionais
 - **projection** – apresentações de slides
 - **screen** – monitores coloridas
 - **tty** – teleimpressores e terminais
 - **tv** – televisores



Media Queries

- Para determinar estilos específicos para a impressão em papel, por exemplo, podíamos adotar duas formas:
 - Acrescentando um link para uma outra folha de estilos no cabeçalho do seu documento:

```
<link rel="stylesheet" href="estilos.css" media="screen">  
<link rel="stylesheet" href="impressao.css" media="print">
```

- Ou utilizando um CSS unificado com a função **@media**:

```
@media print {  
    /* estilos */  
}
```



Media Queries

- Com a evolução dos dispositivos, alguns aparelhos não se encaixavam em nenhuma destas categorias
 - Por exemplo, um **smartphone** moderno tem a tela e resolução muito melhores do que os celulares que se encaixam na categoria handheld
 - E o que dizer dos **tablets**?
- Por causa dessas limitações, o CSS3 introduziu o **Media Queries**, uma espécie de upgrade dos Media Types
- Media Query é uma expressão lógica que não verifica apenas o tipo do dispositivo, mas também a capacidade dele

```
@media (condição) {  
    /* estilos */  
}
```



Media Queries

- Media queries permitem que a página use diferentes regras de estilo CSS com base nas características do dispositivo onde o website está sendo exibido
- Características que podem ser avaliadas em Media Queries:
 - **Width** – Largura da janela do browser/device.

```
@media (max|min-width: 600px) {  
    /* estilos */  
}
```

- **Height** – Altura da janela do browser/device.

```
@media (max|min-height: 600px) {  
    /* estilos */  
}
```



Media Queries

- Media queries permitem que a página use diferentes regras de estilo CSS com base nas características do dispositivo onde o website está sendo exibido
- Características que podem ser avaliadas em Media Queries:
 - **Device-aspect-ratio** – Proporção da tela do dispositivo.

```
@media (max|min-aspect-ratio: 16/9) {  
    /* estilos para monitores widescreen 16:9*/  
}
```

- **Orientation** – Largura da janela do browser.

```
@media (orientation: landscape|portrait) {  
    /* estilos para monitores widescreen 16:9*/  
}
```



Media Queries

- Outras características que podem ser avaliadas por expressões Media Queries:
 - **Color** – Número de bits por cor. Se o valor for zero o dispositivo é monocromático.

```
@media (monochrome) {  
  body{  
    color: black;  
    background-color: white;  
  }  
}
```



Media Queries

- Outras características que podem ser avaliadas por expressões Media Queries:
 - **Resolution** – Resolução do dispositivo (densidade por pixel).

```
@media (min-resolution: 300dpi) {  
  /* A tela tem uma densidade de pixels de pelo menos 300  
    pontos por polegada*/  
}
```

```
@media (max-resolution: 300dpi) {  
  /* A tela tem uma densidade inferior de pixels de pelo menos  
    300 pontos por polegada*/  
}
```

```
@media (resolution: 300dpi) {  
  /* A tela tem uma densidade exata de pixels 300 pontos  
    por polegada*/  
}
```



Media Queries

- Outras características que podem ser avaliadas por expressões Media Queries:
 - **Operadores** – Através dos operadores not (não), and (e) e only (apenas) é possível combinar algumas características

Operador AND

- É utilizado para combinar múltiplas media features em uma mesma media query, requerendo que cada sequência de características, retorne verdadeiro na ordem para que a query seja verdadeira

```
@media (min-width: 769px) and (max-width:1024px) {  
    /* Estilizações em CSS*/  
}
```

```
@media screen and (max-width:1024px) {  
    /* Estilizações em CSS*/  
}
```



Media Queries

- Outras características que podem ser avaliadas por expressões Media Queries:
 - **Operadores** – Através dos operadores not (não), and (e) e only (apenas) é possível combinar algumas características

Operador ONLY

- Usamos este operador para navegadores que não reconhecem estilos aplicados com media query.

```
@media only screen and (max-width:1024px) {  
    /* Estilizações em CSS*/  
}
```



Media Queries

- Outras características que podem ser avaliadas por expressões Media Queries:
 - **Operadores** – Através dos operadores not (não), and (e) e only (apenas) é possível combinar algumas características

Operador OR

- É usado com o caractere (,) que simboliza que para ocorrer alteração no media, uma das condições definidas precisa ser verdadeira.

```
@media (max-width:1024px), print and (orientation: portrait){  
    /* Estilizações em CSS*/  
}
```

CSS



Media Queries

- Outras características que podem ser avaliadas por expressões Media Queries:
 - **Operadores** – Através dos operadores not (não), and (e) e only (apenas) é possível combinar algumas características

Operador NOT

- Usamos este operador lógico para negar uma característica na query, porém, ele não pode ser usado sozinho, pois necessita de algo a ser negado para cair na condição aplicada.

```
@media not all and (monochrome){  
    /* Estilizações em CSS*/  
}
```



Exemplos de Media Queries

- É possível criar um CSS específico para telas de, no máximo, 320px (como um iPhone em modo retrato):

```
<link rel="stylesheet" href="iphone.css"
      media="screen and (max-width: 320px)">
```

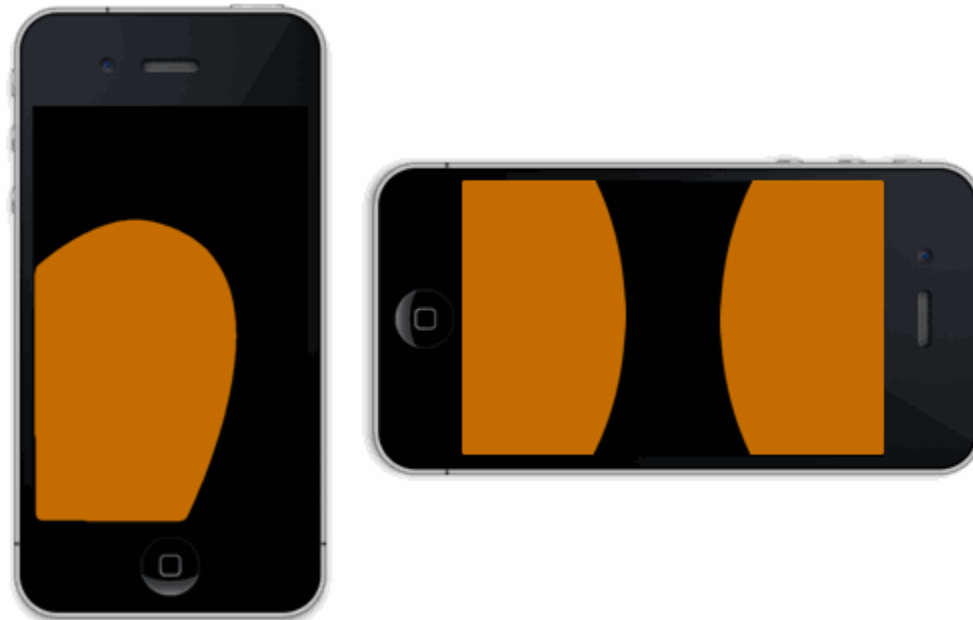
- Além dos media queries de tamanho, podemos adotar uma media query que pega a orientação do dispositivo:

```
<link rel="stylesheet" href="phones.css"
      media="screen and (min-width: 320px) and
            (orientation: portrait)">
```



Exemplos de Media Queries

- Por exemplo, podemos customizar o layout de acordo com a forma do usuário segurar o aparelho
 - É sabido que um celular em modo retrato é mais usável com navegação na parte de baixo à esquerda; e, no modo paisagem, com navegação no topo e nas laterais



Viewport

- É a área visível do usuário de uma página da web.
- A viewport varia de acordo com o dispositivo
- O HTML5 introduziu um método para permitir que os web designers assumam o controle da janela

```
<meta name="viewport" content="device-width, initial-scale=1.0">
```



Viewport

- É a área visível do usuário de uma página da web.
- A viewport varia de acordo com o dispositivo
- O HTML5 introduziu um método para permitir que os web designers assumam o controle da janela

```
<meta name="viewport" content="device-width, initial-scale=1.0">
```

define a largura da página para seguir a largura da tela do dispositivo.

define o nível de zoom inicial quando a página é carregada pela primeira vez pelo navegador.



Viewport

- É
- A
- O
- de



io de
do co
mét
ontro



s web



Viewport

- **DICAS:**

- NÃO use elementos grandes de largura fixa
- NÃO permita que o conteúdo dependa de uma largura específica da janela de visualização para renderizar bem
- Use consultas de mídia CSS para aplicar estilos diferentes para telas pequenas e grandes



Exemplos de Media Queries

- Lista com os breakpoints mais comuns utilizados para Media Query.
 - 320px—480px: Mobile devices
 - 481px—768px: iPads, Tablets
 - 769px—1024px: Small screens, laptops
 - 1025px—1200px: Desktops, large screens
 - 1201px and more — Extra large screens, TV



Exemplos de Media Queries

- Além de declarar as media queries na tag `<link>` no HTML, podemos também fazer direto dentro do CSS

```
/* regra aplicada em todo lugar */
body { background: blue; }

/* aplica somente a partir de 768px */
@media screen and (min-width: 768px) {
    body { font-size: 80%; }
}

/* aplica somente a partir de 1024px em landscape */
@media screen and (min-width: 1024px) and (orientation: landscape) {
    nav { display: flex; }
}
```



Exemplos de Media Queries

- Atividade. Dado o HTML e CSS faça o que se pede.

```
<style type="text/css">
  .exemplo{
    padding: 20px;
    color: white;
  }
</style>
<body>
  <div>
    <h2> Breakpoints Media Query </h2>
    <p class="exemplo">
      Redimensione a janela do navegador para ver
      como a cor de fundo deste parágrafo muda em diferentes tamanhos
      de tela
    </p>
  </div>
</body>
```



Exemplos de Media Queries

- Considerando a lista com 5 (cinco) breakpoints e o código, faça:
 1. Crie regras de Media Query para cada item da lista de breakpoints (slide 92).
 2. Deve considerar apenas (only) Media Type screen e sua largura (width)
 3. Para cada regra aponte uma cor diferente para o background do seletor **.exemplo**
 4. Se a orientação for landscape, o texto do H2 deve ter sua cor alterada para azul
 5. Utilize as ferramentas de desenvolvedores dos web browsers para fazer os testes.



Referências

- <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- <https://web.dev/learn/css/flexbox/>
- <https://web.dev/learn/css/grid/>
- <https://ishadeed.com/article/grid-layout-flexbox-components/>

