

# Desenvolvimento Web no lado cliente – CSS: Responsividade e Layout Flexível (Flexbox) 2023/1

GSI019 - Programação para Internet

Prof. Dr. Rafael D. Araújo

*[rafael.araujo@ufu.br](mailto:rafael.araujo@ufu.br)*

<http://www.facom.ufu.br/~rafaelaraujo>

# Design responsivo

- Técnica de criar páginas para a Web que se adaptam a diferentes resoluções, telas, dispositivos, sem precisar criar uma página específica para cada situação
- Pode envolver a utilização de vários recursos, como a meta tag *viewport*, media queries, unidades relativas, flexbox, grid etc.

# Viewport

- ***Viewport*** é a área visível de renderização da página no navegador
- Na *viewport*, o dimensionamento (largura e altura) é tratado utilizando o conceito de **pixel CSS**
- 1 pixel CSS de largura na *viewport* pode não corresponder a exatamente 1 pixel físico da tela do dispositivo, pois é levado em conta a densidade de pixels da tela, o que determina o **pixel ratio** do dispositivo
- Por exemplo, em um celular com pixel ratio = 4, ao definir `margin-left: 10px`, a margem efetivamente ocupará a largura de  $4 \times 10 = 40$  pixels da resolução horizontal

# Resolução, Pixel Ratio e Viewport

Aparelho	Resolução da tela (pixels)	Densidade de pixels da tela	Pixel Ratio padrão	Resolução da Viewport (pixels CSS)
Galaxy S20 6,2"	1440 x 3200	563	4	360 x 800
iPhone 12 Pro 6,1"	1170 x 2532	460	3	390 x 844
LG K10 5,3"	720 x 1280	277	2	360 x 640

De uma forma geral, os navegadores calculam o pixel ratio do dispositivo como uma aproximação da equação: densidade de pixels da tela / 150

# Simulação de dispositivo móvel no navegador

- Nos principais navegadores é possível simular a tela de dispositivos móveis pelo módulo do desenvolvedor



# Simulação de dispositivo móvel no navegador



# Meta Tag Viewport

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

- Viabiliza a responsividade de acordo com o dispositivo e sua tela
- Faz com que o **pixel ratio** do dispositivo móvel seja considerado quando a página for acessada pelo dispositivo
- Portanto, o dimensionamento dos elementos passa a considerar a densidade de pixels da tela do dispositivo
- Resultado: página melhor escalonada em dispositivos com alta resolução e tela pequena

# Meta Tag Viewport

- Utilizar a meta tag *viewport* não faz com que a página se torne totalmente responsiva
- Essa tag é apenas o **primeiro passo** para o design responsivo
- Normalmente ela é utilizada em conjunto com outros recursos como:
  - Media queries
  - Unidades relativas (como %)
  - Módulo Flexbox, Grid etc.



# Exercício

- Criar a seguinte página:

```
<body>  
  <h1>Teste de viewport</h1>  
  <h2>Responsividade</h2>  
  <label for="email">E-mail</label>  
  <input type="email" id="email">  
  <br><br>  
    
    
</body>
```

- Na sequência, utilize a meta tag viewport e utilize os recursos do navegador para visualização em diferentes dispositivos

## Teste de viewport

### Responsividade

E-mail



# Media Queries

- Permite ao desenvolvedor testar condições a respeito do navegador ou o dispositivo do usuário para aplicar ou não as regras CSS
- Por exemplo, é possível aplicar estilos CSS apenas quando a tela do dispositivo tenha uma largura mínima ou máxima; ou esteja em determinada posição, como na vertical ou na horizontal

# Media Queries

**all** - para todos os dispositivos  
**screen** - para dispositivos com tela  
**print** - para impressão (ex. modo Ctrl-P)

**MediaType** é opcional.  
Se omitido será  
considerado **all**

```
@media MediaType AND MediaCondition {  
    /* Código CSS */  
}
```

(min-width: 400px)  
(max-width: 900px)  
(min-width: 400px) and (max-width: 900px)  
(400px <= width <= 900px)  
(orientation: portrait)  
(orientation: landscape)

Exemplos de expressões (**media condition**). **min-width**,  
**max-width** e **orientation** são exemplos de **media features**

# Media Queries

```
body {  
  width: 60%;  
  padding: 2% 0;  
  margin: 0 auto;  
}  
  
@media (max-width: 480px) {  
  body {  
    width: 95%;  
  }  
}
```

Neste exemplo, o corpo da página aparecerá centralizado e com largura de 60% em dispositivos com telas largas.

Porém, em dispositivos com largura de *viewport* menor ou igual a 480px (smartphones), a página poderá se expandir e ocupar 95% da largura.

`max-width` é uma *media feature*. Há outras opções, como *orientation*.

# Media Queries

```
<style>
  body {
    width: 60%;
    padding: 2% 0;
    margin: 0 auto;
    line-height: 2.0;
  }

  @media print and (orientation: portrait) {
    body {
      line-height: 1.1;
      font-size: 10pt;
    }
  }
</style>
```

Neste exemplo, o espaçamento entre linhas e o tamanho da fonte serão reduzidos quando o documento estiver em modo de impressão na orientação retrato (Ctrl+P).

# Media Query com Lista de Expressões

- É possível combinar uma lista de media queries separando as expressões com vírgula
- O código CSS será aplicado quando pelo menos uma das expressões for verdadeira

```
@media screen and (orientation: landscape),
screen and (min-width: 900px) {
  body {
    background-color:  gray;
  }
}
```

O cor de fundo será cinza quando a “media” for do tipo “tela” e a orientação for “horizontal” ou quando a “media” for do tipo “tela” e a largura for de pelo menos 900px.  
**OBS:** não existe o operador **OR** em media queries, mas a vírgula tem papel similar.

# CSS Flexbox

- **Flexbox** é um módulo de layout flexível da CSS para criação de designs responsivos e dinâmicos
- Flexbox fornece recursos para controle do alinhamento, distribuição e ordenação dos elementos dentro de um container
- O container é geralmente chamado de container pai ou container flexível (**flex**)
- Os elementos-filhos dentro do container são comumente denominados itens flexíveis
- Para ativar o layout flexível, o container pai deve ser estilizado com `display: flex`

# Itens organizados SEM flexbox

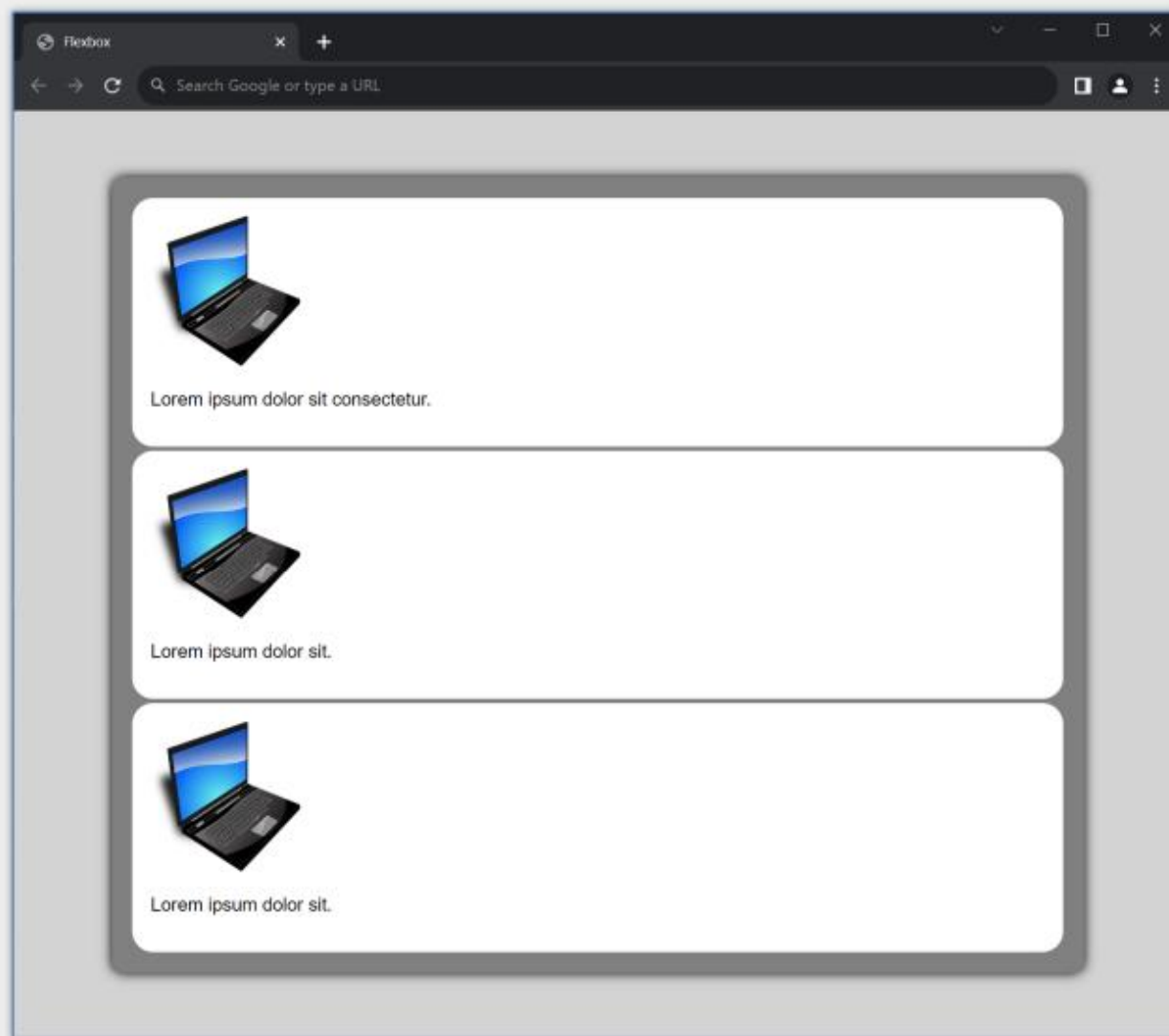
CSS

```
.container {  
  background-color: gray;  
  padding: 1rem;  
}
```

```
.item {  
  background-color: white;  
  padding: 1rem;  
  border: 2px solid gray;  
  border-radius: 20px;  
}
```

HTML

```
<div class="container">  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit consectetur.</p>  
  </div>  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
</div>
```





# Itens organizados COM flexbox

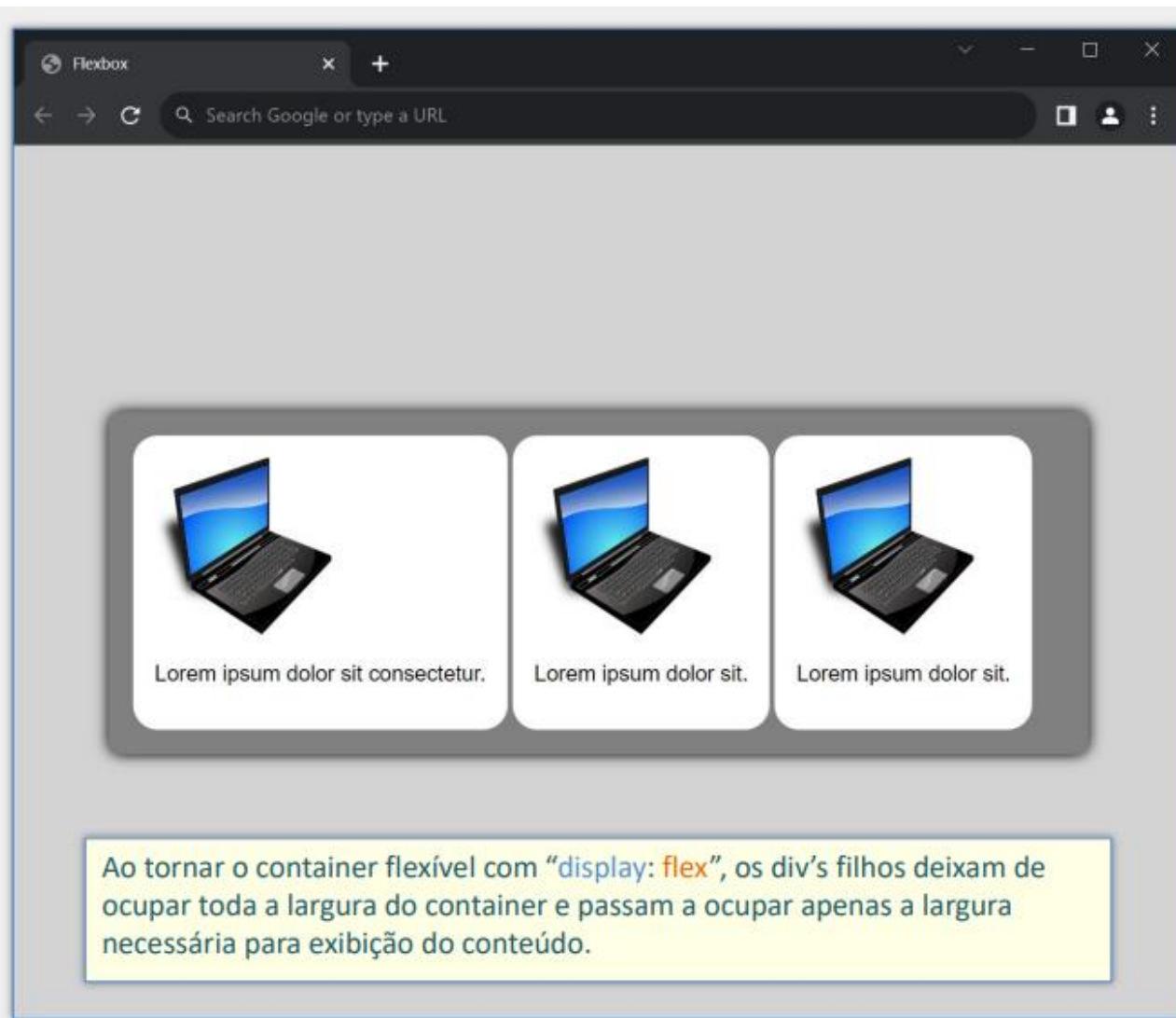
CSS

```
.container {  
  background-color: gray;  
  padding: 1rem;  
  display: flex;  
}
```

```
.item {  
  background-color: white;  
  padding: 1rem;  
  border: 2px solid gray;  
  border-radius: 20px;  
}
```

HTML

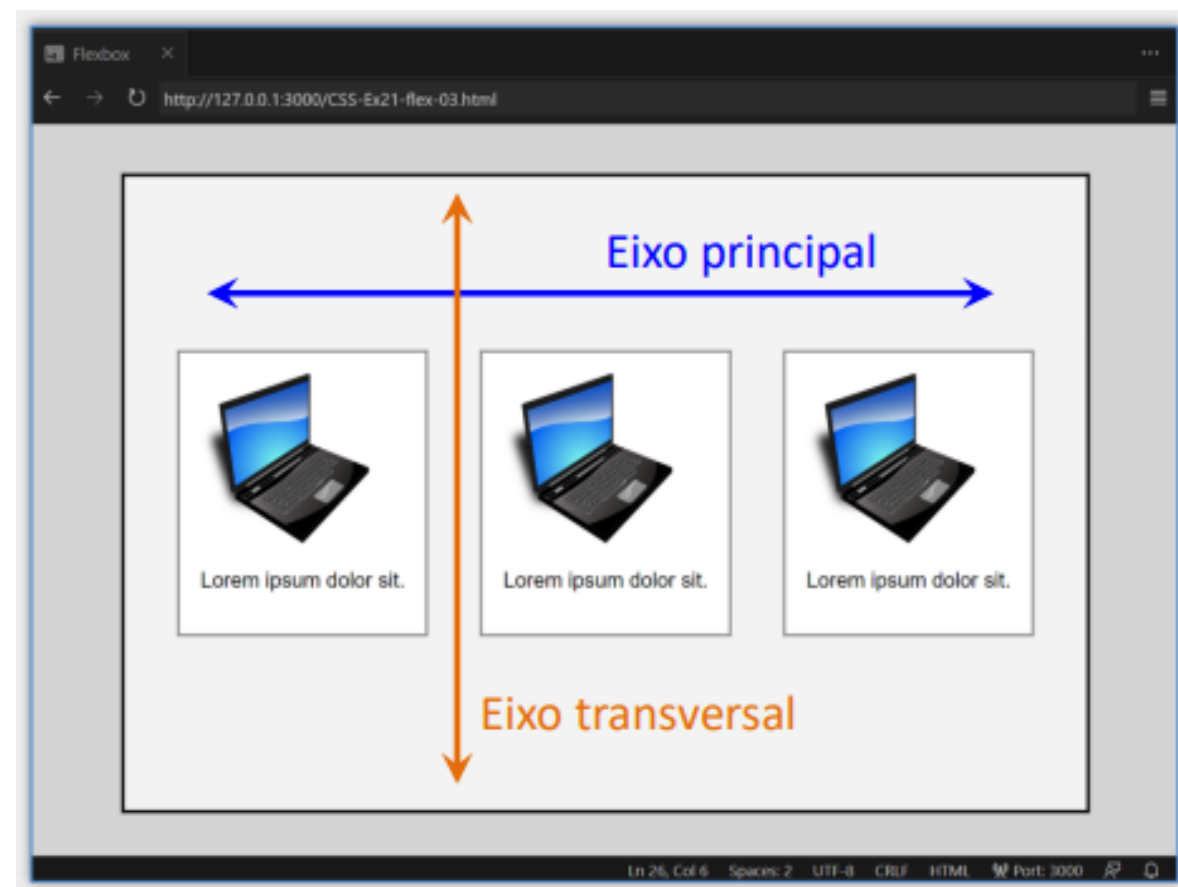
```
<div class="container">  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit consectetur.  
  </div>  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
</div>
```



Ao tornar o container flexível com “display: flex”, os div’s filhos deixam de ocupar toda a largura do container e passam a ocupar apenas a largura necessária para exibição do conteúdo.

# Eixo Principal e Eixo Transversal

- Flexbox utiliza dois eixos: o **eixo principal** e o **eixo transversal** (*cross axis*)
- Por padrão, o eixo principal é horizontal e o eixo transversal é vertical (mas esse comportamento pode ser alterado)
- O eixo principal determina a direção na qual os itens flexíveis são dispostos
- O eixo transversal é perpendicular ao eixo principal



# Propriedade justify-content

- A propriedade `justify-content` pode ser usada em containers flexbox e containers grid
- Em um container flexbox, `justify-content` determina o espaçamento entre os itens ao longo do eixo principal
- Por padrão, o eixo principal é horizontal e `justify-content` alinhará os itens na horizontal
- Se o eixo principal for alterado para vertical, então `justify-content` alinhará os itens na vertical
- `justify-content` deve ser usada no container (e não nos itens)

# Propriedade justify-content

**justify-content: flex-start**



**justify-content: space-between**



**justify-content: flex-end**



**justify-content: space-around**



**justify-content: center**



**justify-content: space-evenly**



# Propriedade flex-direction

- A propriedade `flex-direction` permite alterar o eixo principal de horizontal para vertical
- Seu valor padrão é `flex-direction: row`
- Alterando para `flex-direction: column`, o eixo principal passa a ser vertical
  - Neste caso, `justify-content` alinhará os itens na vertical
- Outros valores possíveis:
  - `flex-direction: row-reverse` (itens na horizontal organizados da direita para esquerda)
  - `flex-direction: column-reverse` (itens na vertical organizados de baixo para cima)
  - Em ambos os casos, os itens são apresentados na ordem invertida

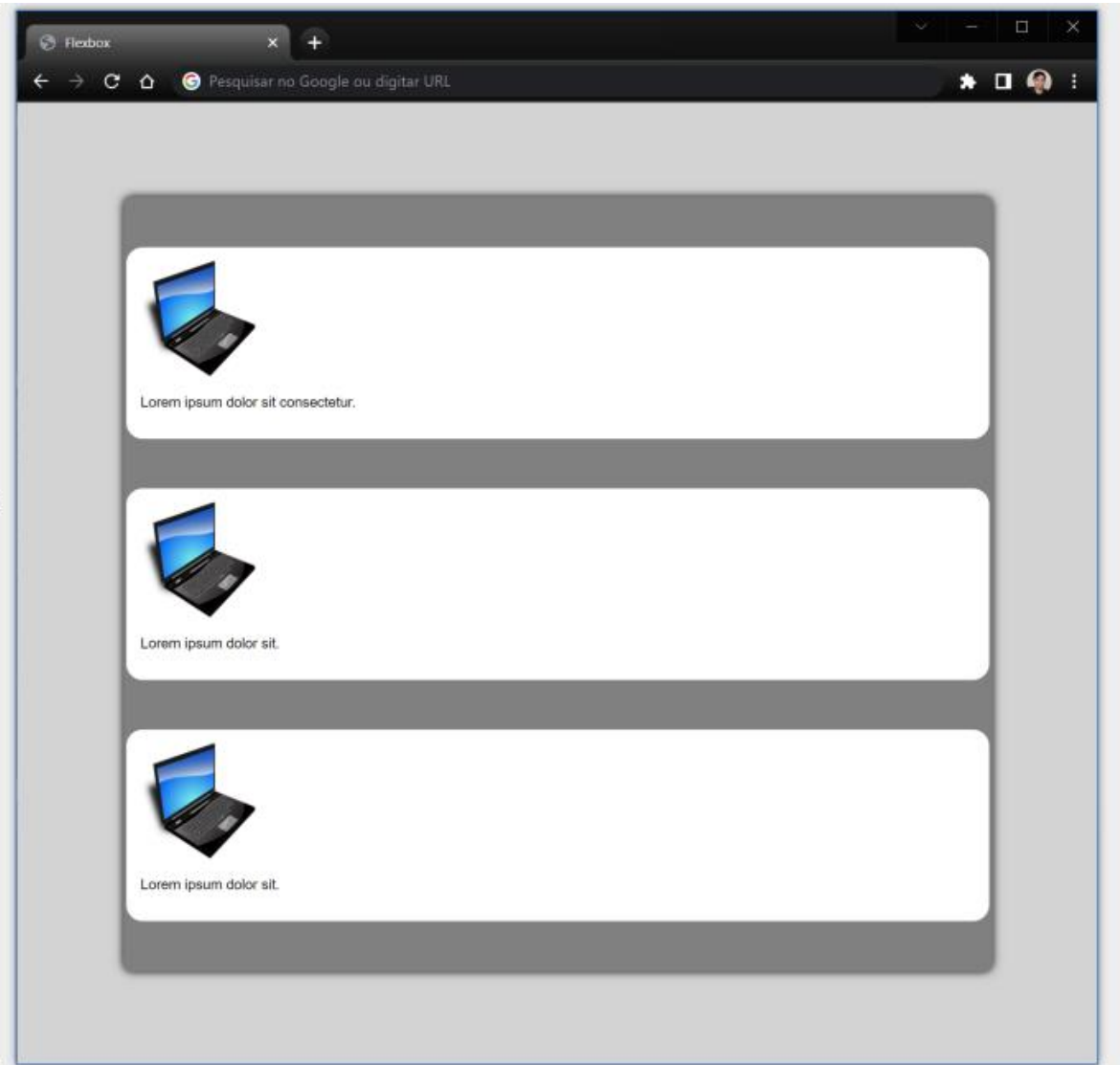
# flex-direction: column

CSS

```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  padding: 2px;  
  
  display: flex;  
  height: 80vh;  
  flex-direction: column;  
  justify-content: space-evenly;  
}
```

HTML

```
<div class="container">  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
  <div class="item">  
      
    <p>Lorem ipsum dolor sit.</p>  
  </div>  
</div>
```





# flex-direction: column e justify-content



# Container de containers

## HTML

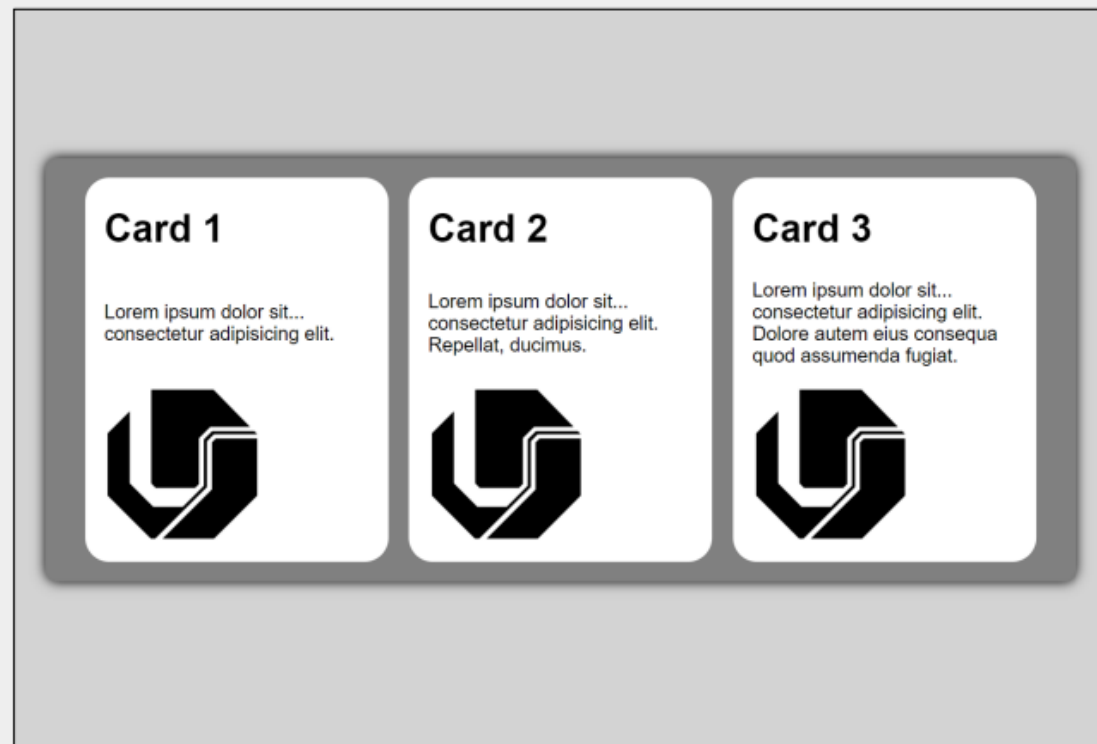
```
<div class="container">
  <div class="card">
    <h1>Card 1</h1>
    <p>Lorem ipsum dolor sit...
    
  </div>
  <div class="card">
    <h1>Card 2</h1>
    <p>Lorem ipsum dolor sit...
    
  </div>
  <div class="card">
    <h1>Card 3</h1>
    <p>Lorem ipsum dolor sit...
    
  </div>
</div>
```

## CSS

```
.container {
  background-color: gray;
  box-shadow: 0 0 10px;
  padding: 1em;
  display: flex;
  justify-content: space-evenly;
}

.card {
  background-color: white;
  padding: 1rem;
  max-width: 220px;
  border-radius: 20px;
  display: flex;
  flex-direction: column;
  justify-content: space-between
}
```

## Resultado



Este exemplo utiliza um **container externo** flex, com fundo cinza, com eixo principal na **horizontal** tendo como itens flexíveis os elementos **div** correspondentes aos **cards**. Entretanto, observe que cada **div** interno também é um container flex com eixo principal na vertical tendo como filhos flexíveis os elementos **h1**, **p** e **img**. Neste caso, repare que “**justify-content: space-between**” dos **div**’s internos está organizando os elementos verticalmente dentro de cada card, fazendo com que o título **h1** fique alinhado à margem superior, a imagem alinhada à base e o parágrafo alinhado ao centro do espaço vertical restante.



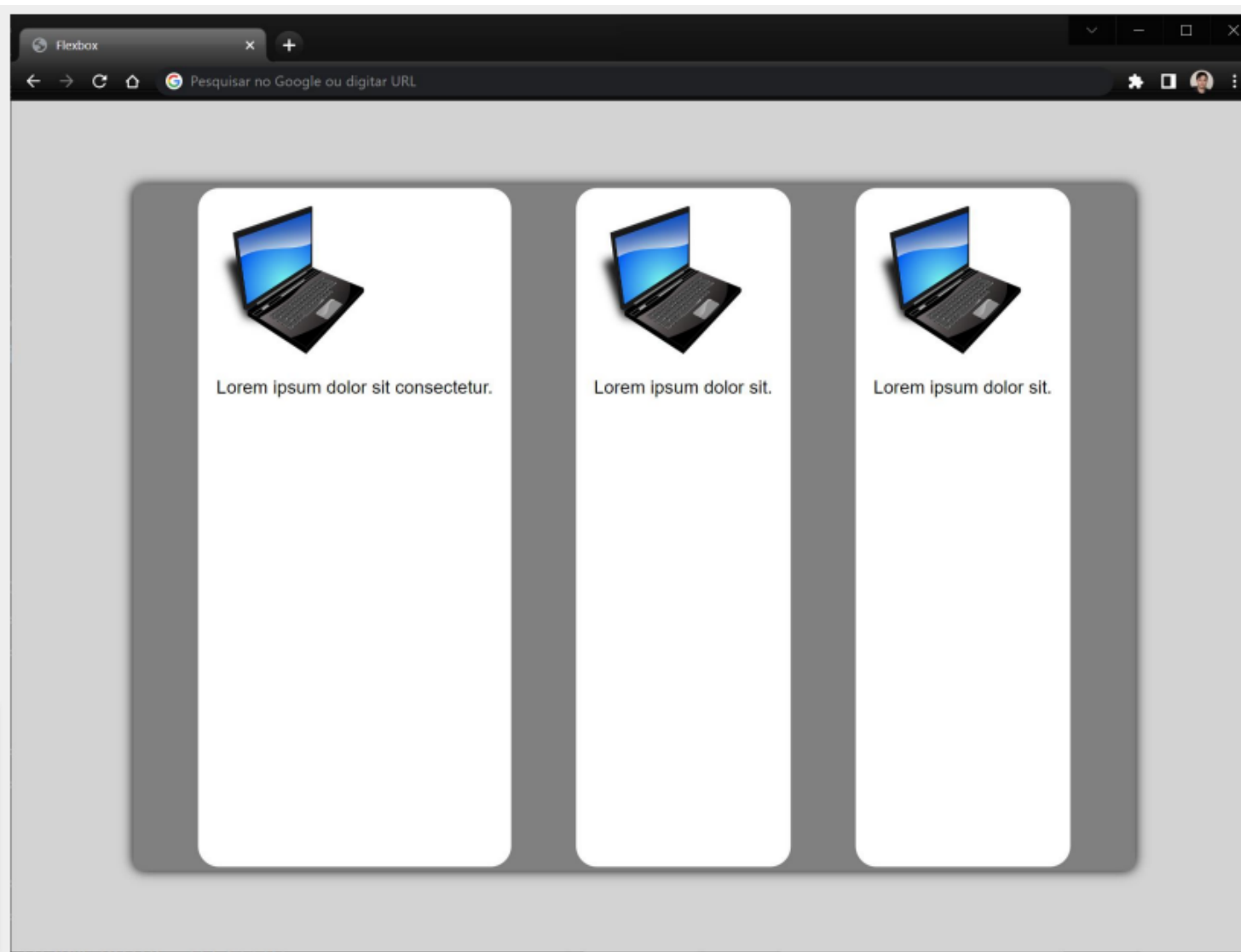
# Alinhamento no eixo transversal

- Para alinhar os itens dentro de um container flex no eixo transversal, pode-se utilizar a propriedade `align-items` do container
- Há um total de 22 valores possíveis para `align-items`. Exemplos:
  - `stretch`
  - `flex-start`
  - `flex-end`
  - `center`
- O valor padrão de `align-items` é `stretch`, o que faz com que os itens sem tamanho definido sejam esticados para ocupar todo o espaço disponível no eixo transversal

# Alinhamento no eixo transversal

```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  padding: 2px;  
  
  display: flex;  
  height: 80vh;  
  justify-content: space-evenly;  
}  
  
.item {  
  background-color: white;  
  padding: 1rem;  
  border: 2px solid gray;  
  border-radius: 20px;  
}
```

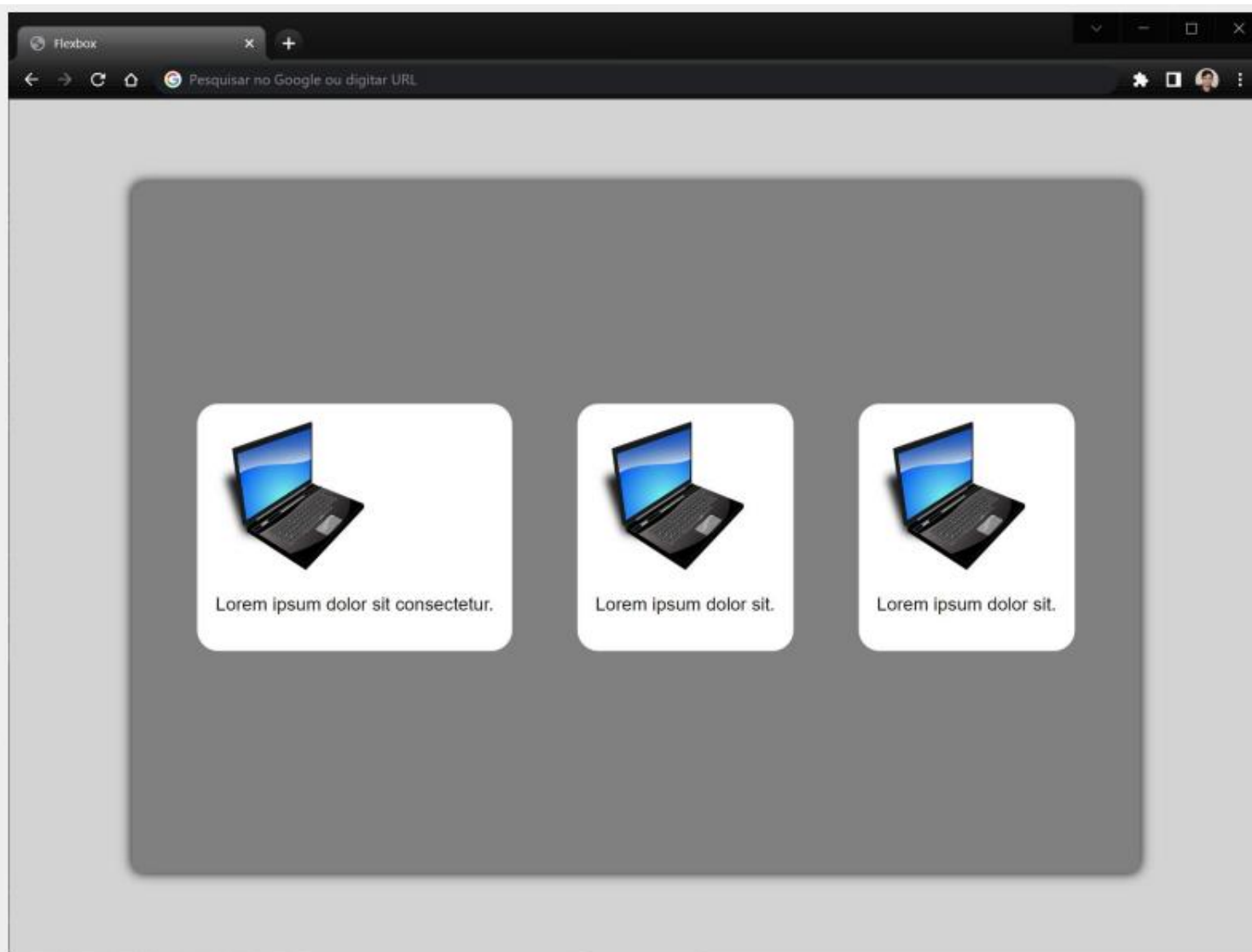
Neste exemplo é definida uma altura para o container (80vh) e os itens são esticados no eixo transversal para preencher toda essa altura, pois a propriedade `align-items` não foi alterada e portanto tem o valor padrão **stretch**.



# Alinhamento no eixo transversal

```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  padding: 2px;  
  
  display: flex;  
  height: 80vh;  
  justify-content: space-evenly;  
  align-items: center;  
}  
  
.item {  
  background-color: white;  
  padding: 1rem;  
  border: 2px solid gray;  
  border-radius: 20px;  
}
```

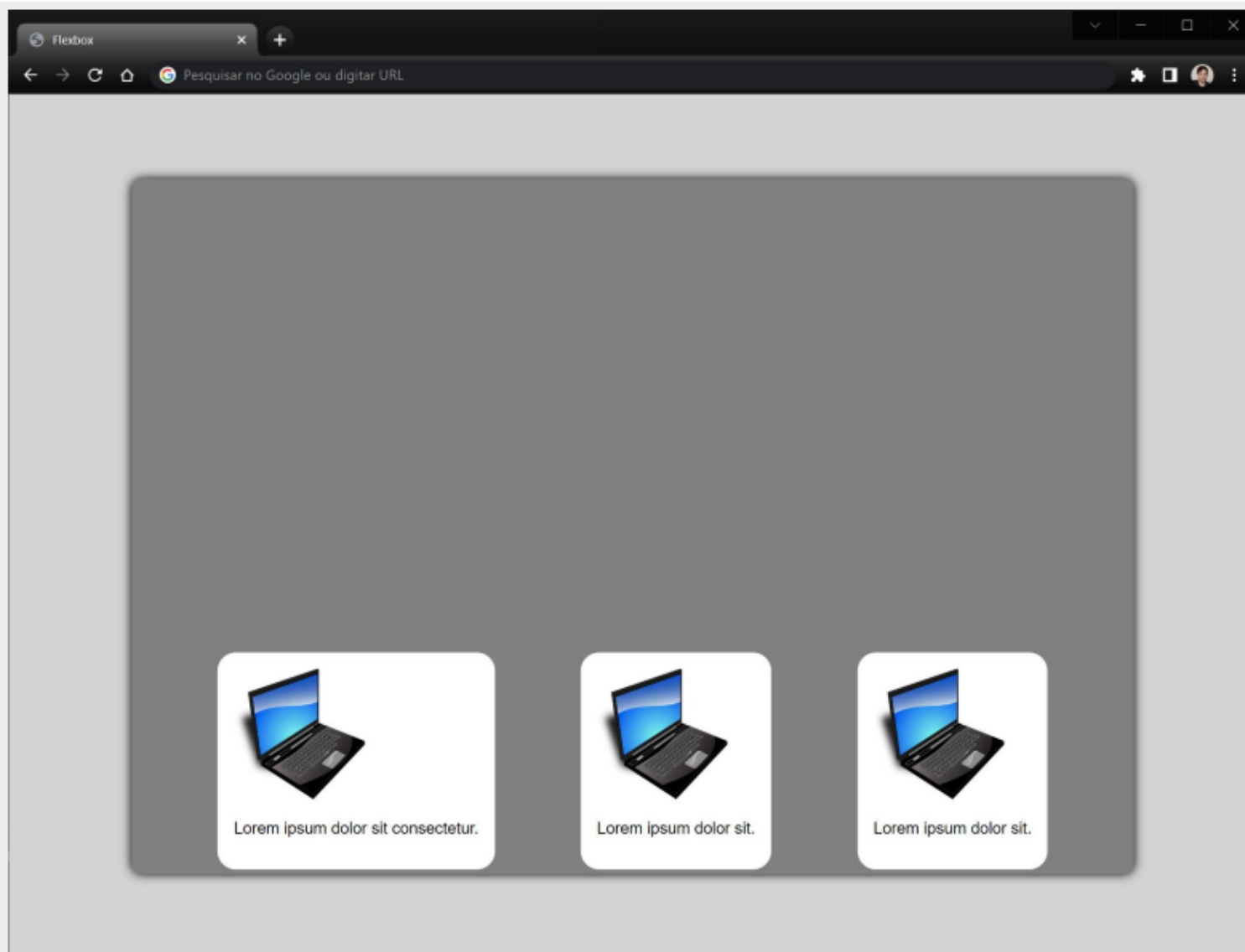
Neste exemplo os itens são alinhados ao centro no eixo transversal, pois foi utilizado `align-items: center` no container flex.



# Alinhamento no eixo transversal

```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  padding: 2px;  
  
  display: flex;  
  height: 80vh;  
  justify-content: space-evenly;  
  align-items: flex-end;  
}  
  
.item {  
  background-color: white;  
  padding: 1rem;  
  border: 2px solid gray;  
  border-radius: 20px;  
}
```

Neste exemplo os itens são alinhados no final do eixo transversal utilizando `align-items: flex-end`.



# Propriedade flex-wrap

- Por padrão, os itens flex são dispostos em uma única linha e ocorrerá *overflow* quando o container não comportar todos os itens
- Uma forma de evitar o overflow e tornar o layout mais flexível é utilizar a propriedade `flex-wrap` com o valor `wrap`
- Neste caso, os itens deixarão de ocupar uma única linha e poderão ocupar múltiplas linhas (ou seja, poderá haver quebra de linha)
  - O valor padrão de `flex-wrap` é `nowrap` (sem quebra)

# Propriedade align-content

- Quando `flex-wrap` é definida para `wrap` é possível utilizar a propriedade `align-content` para ajustar o alinhamento dos itens no eixo transversal
- Com `align-content` é possível ajustar o alinhamento de múltiplas linhas de uma forma que não é possível com `align-items`
- Porém, `align-content` não tem efeito quando `flex-wrap` tem o valor `nowrap`
- Os valores possíveis são similares àqueles de `justify-content` como `flexstart`, `flex-end`, `center`, `space-between`, `space-evenly` etc.

# align-items vs align-content

align-items: center



```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  padding: 2px;  
  
  display: flex;  
  height: 80vh;  
  justify-content: space-evenly;  
  flex-wrap: wrap;  
  align-items: center;  
}
```

align-content: center



```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  padding: 2px;  
  
  display: flex;  
  height: 80vh;  
  justify-content: space-evenly;  
  flex-wrap: wrap;  
  align-content: center;  
}
```

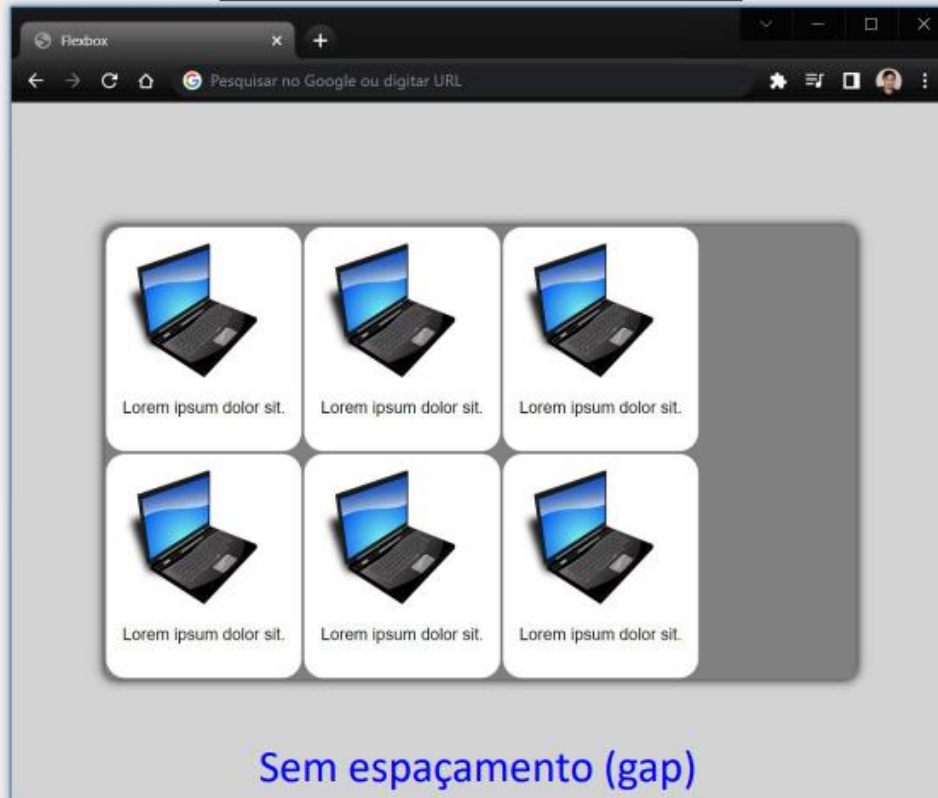
# Propriedade gap

- A propriedade `gap` do container permite inserir espaçamentos entre os itens, nas linhas e colunas
- Se utilizado um único valor (`gap: valor`), será adicionado o mesmo espaçamento entre as linhas e as colunas
- Com dois valores (`gap: val1 val2`), o primeiro valor será utilizado como espaçamento entre as linhas e o segundo, entre as colunas
- `gap` é uma propriedade abreviada de `row-gap` e `column-gap`

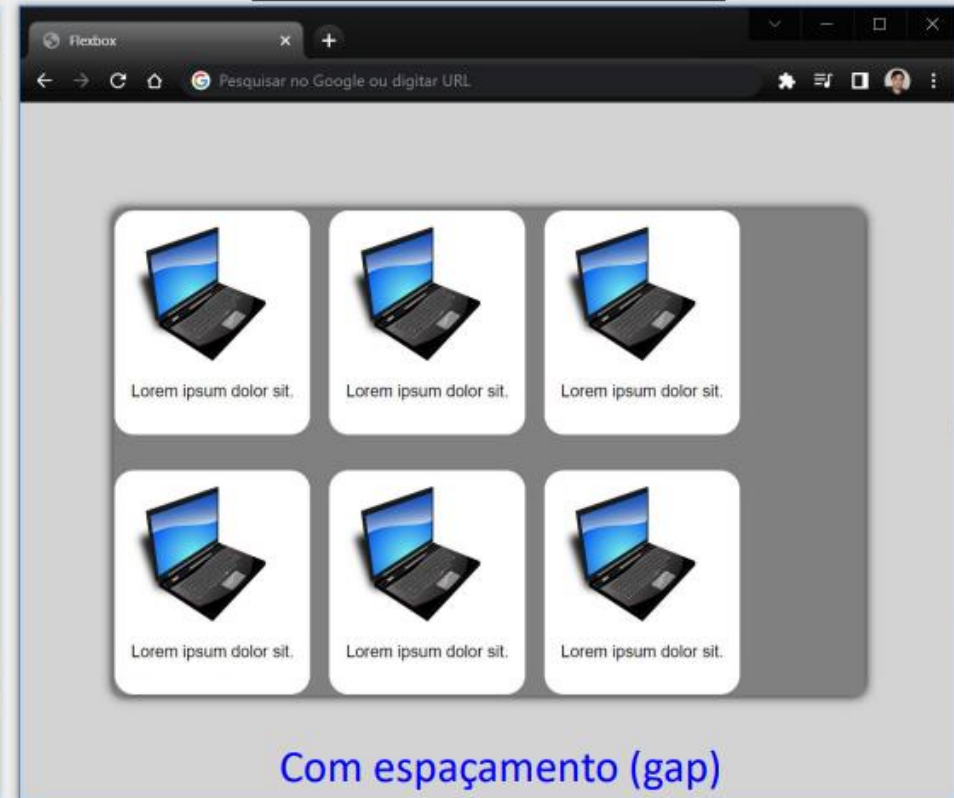


# Propriedade gap

```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  display: flex;  
  flex-wrap: wrap;  
}
```



```
.container {  
  background-color: gray;  
  box-shadow: 0 0 10px;  
  display: flex;  
  flex-wrap: wrap;  
  gap: 2rem 1rem;  
}
```



# Propriedades de item

- As propriedades apresentadas anteriormente são propriedades específicas para o **container** flexível, devendo ser utilizadas apenas no container
- Há também propriedades a serem utilizadas nos itens:
  - `align-self`: permite alinhar itens individualmente no eixo transversal
  - `flex-grow`: indica como os itens devem expandir para ocupar o espaço disponível no container. Valor padrão: 0, itens não expandem
  - `flex-shrink`: indica como os itens devem encolher quando não houver espaço suficiente no container. Valor padrão: 1, itens encolhem na mesma proporção
  - `flex-basis`: define o tamanho principal inicial do item. Valor padrão: auto

# Propriedade flex

- Propriedade abreviada que permite definir as três propriedades anteriores de uma vez: `flex-grow`, `flex-shrink` e `flex-basis`
  - Por exemplo, `flex: 1 0 20px` define `flex-grow: 1`, `flex-shrink: 0` e `flex-basis: 20px`
- Há outras formas de uso que permitem omitir valores. Nesses casos, as propriedades constituintes omitidas terão seus valores definidos automaticamente
- Por exemplo, ao utilizar `flex: 1` estamos definindo `flex-grow` para 1. A omissão dos demais valores permite ao sistema defini-los automaticamente (neste exemplo, `flex-shrink` se manterá em 1 e `flex-basis` será ajustado em 0)
- Portanto, utilizar `flex: 1` tem o mesmo efeito que “`flex-grow: 1, flex-basis: 0`”, permitindo aos itens se expandirem e ocuparem tamanho iguais no container

# Trabalho 3 – para entregar

[Início](#) [Séries](#) [Filmes](#) [Documentários](#) [Login](#)



>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Magnam ipsum, voluptas placeat similique, accusamus voluptatem dolor nam doloribus nisi laborum illum magni eius repellat ratione quidem modi incidunt qui dolorum.



>Lorem ipsum dolor sit amet consectetur adipisicing elit. Sed soluta quis possimus aspernatur quo quos. Doloribus, facilis facere. Aut tenetur odit odio deserunt fuga modi ipsam consequatur rerum commodi earum?



>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Magnam ipsum, voluptas placeat similique, accusamus voluptatem dolor nam doloribus nisi laborum illum magni eius repellat ratione quidem modi incidunt qui dolorum.



>Lorem ipsum dolor sit amet consectetur adipisicing elit. Sed soluta quis possimus aspernatur quo quos. Doloribus, facilis facere. Aut tenetur odit odio deserunt fuga modi ipsam consequatur rerum commodi earum?

# Referências

- <https://www.w3.org/Style/CSS/>
- <https://developer.mozilla.org/en-US/docs/Web/CSS>
- Agradecimento especial ao Prof. Daniel Furtado pela disponibilização do material: <https://furtado.prof.ufu.br>