



# Desenvolvimento Web II

---

Módulo 2

Responsividade e Layout Flexível (Flexbox)

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Conteúdo protegido por direito autoral, nos termos da Lei nº 9 610/98

A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

# Conteúdo da Aula

- Introdução a Design Responsivo e Viewport
- Introdução a Media Queries
- CSS Flexbox

# Design Responsivo

- Baseado na ideia de que o website deve ter a capacidade de se ajustar para permitir uma boa exibição em todos os dispositivos, seja um desktop, um notebook, um tablet ou um smartphone.
- Evita a necessidade de ter diferentes versões do website para diferentes dispositivos e tamanhos de tela.
- Pode envolver a utilização de vários recursos como a meta tag viewport, media queries, unidades relativas, flexbox, grid etc.

# Conceito de 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 **nem sempre** corresponde a 1 pixel físico da tela do dispositivo, pois pode ser levado em conta o fator de escala conhecido como **pixel ratio**.
- A ideia do **pixel ratio** é permitir que dispositivos com alta densidade de pixels, como smartphones, mostrem os elementos da página ampliados.
- Para ativar o fator de escala é necessário utilizar a tag de responsividade:

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

# Página de Exemplo sem Tag de Responsividade

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Teste de Viewport</title>
</head>
<body>
    <h1>Teste de Responsividade</h1>
    <h2>Sem tag de responsividade</h2>
    <label for="email">E-mail</label>
    <input type="email" id="email"><br><br>
    
</body>
</html>
```

Veja no próximo slide como os elementos desta página são apresentados em um desktop e em um smartphone.

# Visualização da Página sem Tag de Responsividade

Teste de Viewport

127.0.0.1:5500/CSS-Ex60-view-01-B.html

## Teste de Responsividade

### Sem tag de responsividade

E-mail



Visualização da página em computador desktop de 14", resolução 1920x1080

Sem a tag de responsividade, os elementos da página aparecem com tamanho muito reduzido no smartphone devido à alta densidade de pixels de sua tela. O smartphone tem uma grande quantidade de pixels em uma tela muito pequena (ao ser comparado a um desktop, por exemplo).

Teste de Responsividade

Sem tag de responsividade

E-mail



Visualização da página no Galaxy S20, resolução 1440x3200

Programação para Internet

Prof. Dr. Daniel A. Furtado - Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

6

# Página de Exemplo com Tag de Responsividade

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <title>Teste de Viewport</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
    <h1>Teste de Responsividade</h1>
    <h2>Sem tag de responsividade</h2>
    <label for="email">E-mail</label>
    <input type="email" id="email"><br><br>
    
</body>
</html>
```

Mesma página do exemplo anterior, porém com o acréscimo da tag de responsividade. Observe os tamanhos dos elementos quando a página é visualizada em um smartphone (próximo slide).

# Visualização da Página com Tag de Responsividade



## Teste de Responsividade

Com tag de responsividade

E-mail



Visualização da página no  
Galaxy S20, resolução  
1440x3200

# Tag de Responsividade e Pixel Ratio

- Repare que ao utilizar a tag de responsividade (exemplo anterior), o pixel ratio do dispositivo é levado em conta quando a página é exibida em dispositivos móveis.
- No caso do Galaxy S20, o pixel ratio é igual a 4, o que significa que os elementos da página, de uma forma geral, são ampliados em 4x.
- Por exemplo, ao definir `margin-left: 10px`, estamos definindo a margem esquerda para 10 pixels CSS, mas a margem ocupará efetivamente a largura de  $10 \times 4 = 40$  pixels da resolução horizontal do Galaxy S20.

# Quanto maior a densidade de pixels, maior o pixel ratio

Aparelho	Resolução da Tela (pixels)	Densidade de Pixels da Tela	Pixel Ratio Padrão	Res. da Viewport (pixels CSS)
Galaxy S20 6,2"	1440 x 3200	563	4	360 x 800
iPhone 14 Pro 6,1"	1179 x 2556	460	3	393 x 852
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*

**Ex:** Galaxy S20:  $563 / 150 = 3,753 \rightarrow$  pixel ratio no Google Chrome: 4

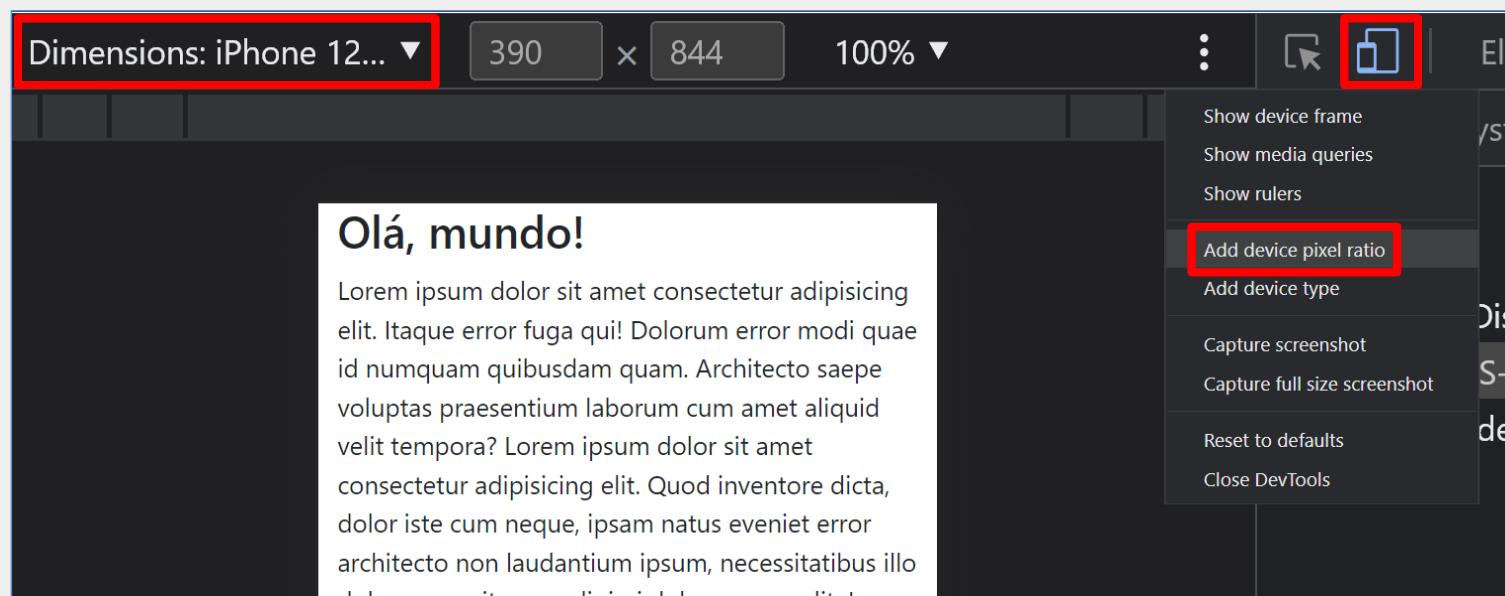
**Ex:** iPhone 12:  $460 / 150 = 3,06 \rightarrow$  pixel ratio no Google Chrome: 3

**Ex:** Google Pixel 5:  $432 / 150 = 2,8 \rightarrow$  pixel ratio no Google Chrome: 2,8

**OBS:** no código JavaScript é possível utilizar a propriedade `window.devicePixelRatio` para obter o pixel ratio do dispositivo que está acessando a página

# Simulação de Dispositivo Móvel no Navegador

- Nos principais navegadores é possível simular a tela de dispositivos móveis.
- No Google Chrome em um desktop, tecle F12, clique em  e depois selecione o aparelho desejado.
- Para visualizar o pixel ratio do dispositivo simulado, clique nos três pontinhos  e selecione [Add device pixel ratio](#).



# Simulação do Dispositivo Móvel no Navegador



# Resumo sobre a 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 móvel.
- **Resultado:** página melhor escalonada em dispositivos com alta resolução e tela pequena.

# Observação sobre a Meta Tag Viewport

- Utilizar a meta tag viewport **não faz com que a página se torne totalmente responsiva.**
- A meta tag viewport 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 da CSS, Grid etc.

# Media Queries

# Introdução à Media Queries

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

# Introdução à Media Queries

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

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

**MediaCondition** é a condição a ser satisfeita para que o código CSS dentro das chaves seja considerado

```
@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 **media condition**  
min-width, max-width e orientation são **media features**

# Introdução à Media Queries

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

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 segunda regra também será utilizada, permitindo que a página se expanda e ocupe 95% da largura.

max-width é uma *media feature*

**OBS:** observe que neste exemplo a media query foi inserida **depois** da 1ª regra, pois queremos que a propriedade **width: 95%** da 2ª regra se sobreponha à **width: 60%** da primeira quando a largura da viewport for inferior a 480px.

# Introdução à Media Queries

```
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;  
    }  
}
```

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)

# Introdução à Media Queries

```
<style>
  body {
    width: 60%;
    padding: 2% 0;
    margin: 0 auto;
  }
  @media (max-width: 480px) {
    body {
      width: 100%;
    }
    header {
      display: none;
    }
  }
</style>
```

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

Porém em smartphones, na vertical, a página se expandirá para ocupar 100% da largura e o cabeçalho (header) não será apresentado.

# 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

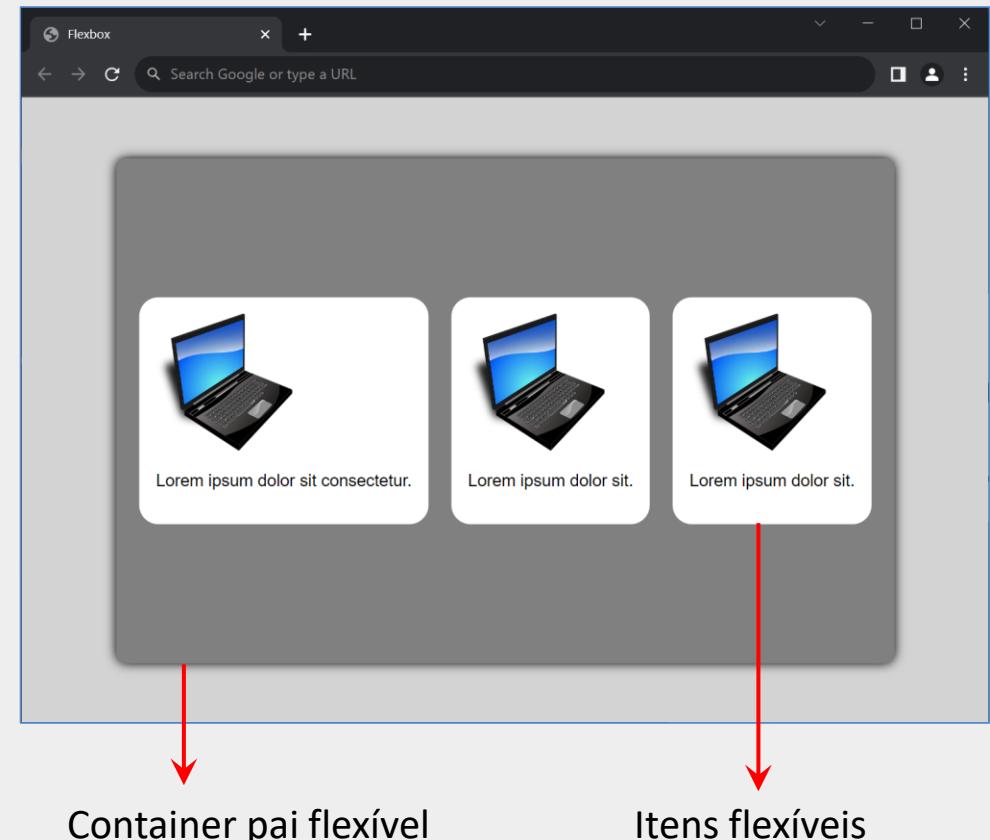
```
@media screen and (orientation: landscape),  
screen and (min-width: 900px) {  
    body {  
        background-color: gray;  
    }  
}
```

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 – Introdução

- 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**".



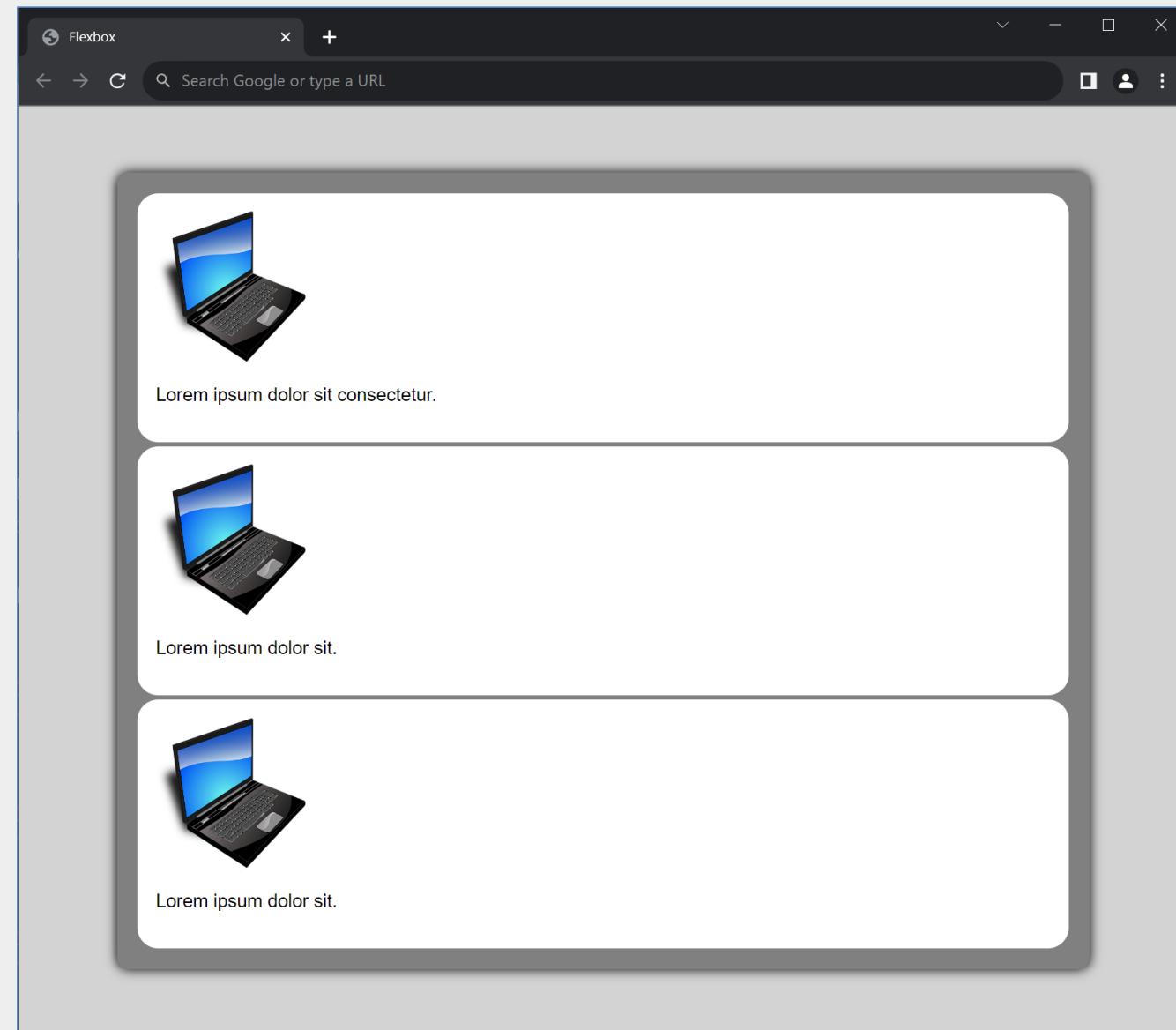
# Itens Organizados SEM flexbox

CSS

```
.container {  
    background-color: gray;  
    padding: 1rem;  
}  
  
.item {  
    background-color: white;  
    padding: 1rem;  
    border-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>
```



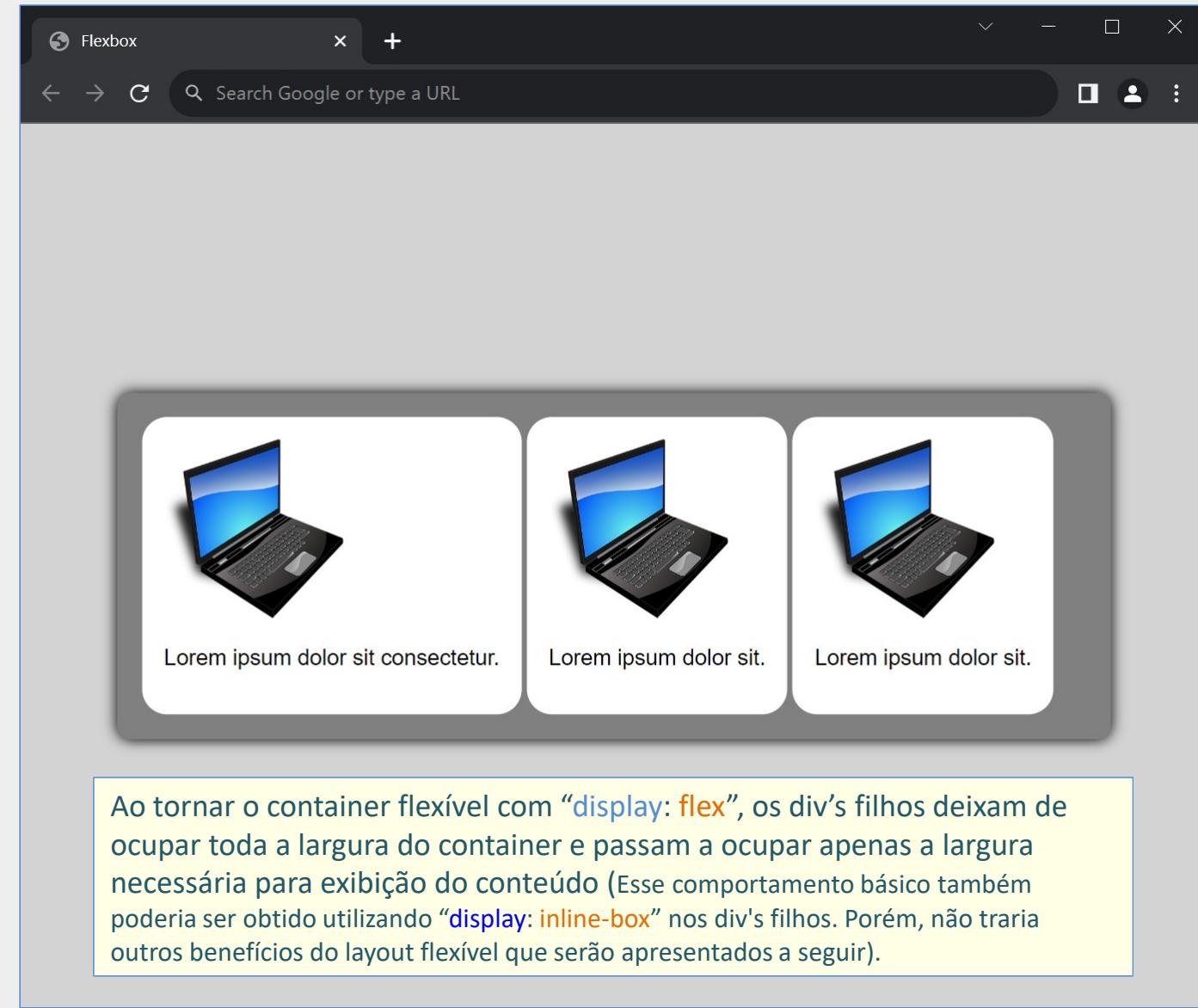
# Itens Organizados COM flexbox

CSS

```
.container {  
    background-color: gray;  
    padding: 1rem;  
    display: flex; // Red box highlights this line  
}  
  
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
}
```

HTML

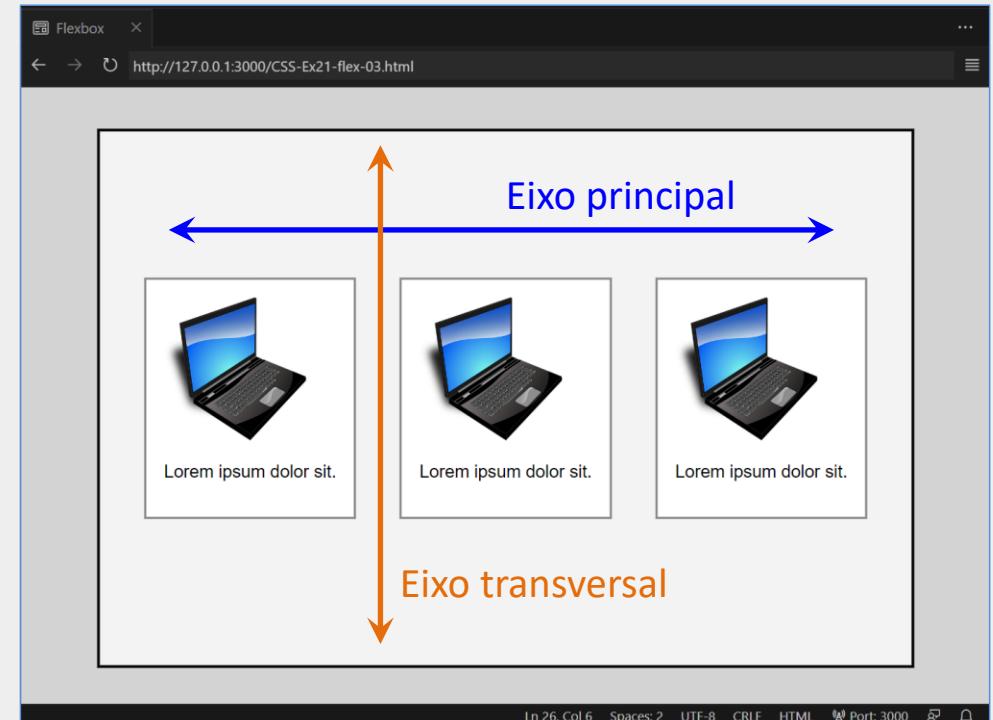
```
<div class="container">  
    <div class="item">  
          
        <p>Lorem ipsum dolor sit consectetur</p>  
    </div>  
    <div class="item">  
          
        <p>Lorem ipsum dolor sit.</p>  
    </div>  
    <div class="item">  
          
        <p>Lorem ipsum dolor sit.</p>  
    </div>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



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 (Esse comportamento básico também poderia ser obtido utilizando “display: inline-box” nos div’s filhos. Porém, não traria outros benefícios do layout flexível que serão apresentados a seguir).

# Eixo Principal e Eixo Transversal

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



# Propriedade justify-content

- Em um container flexbox é possível ajustar o alinhamentos dos itens no **eixo principal** utilizando a propriedade **justify-content**.
- 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).
- Pode ser usada também em containers **grid**.

# Propriedade justify-content

justify-content: **flex-start**



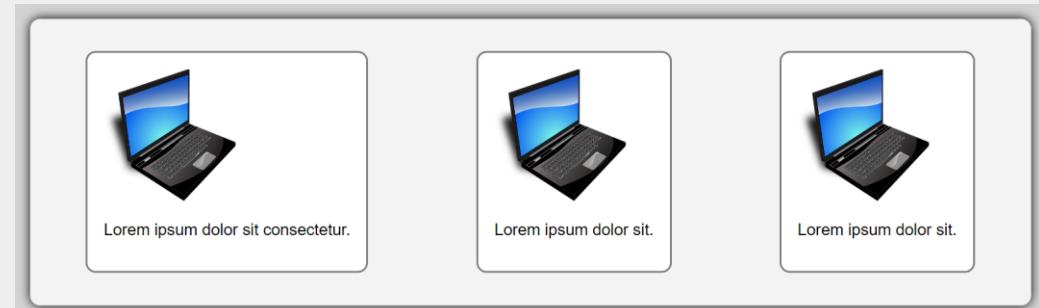
justify-content: **space-between**



justify-content: **flex-end**



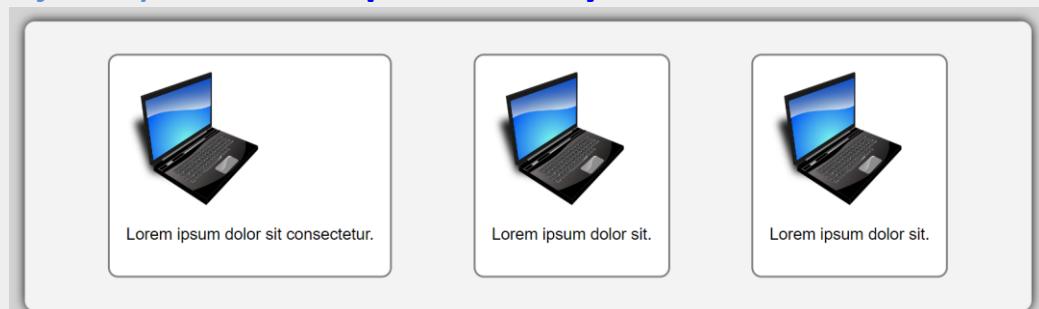
justify-content: **space-around**



justify-content: **center**



justify-content: **space-evenly**



# Propriedade flex-direction

- **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**.
- Deve ser utilizada no container.
- Outros valores possíveis:
  - **flex-direction: row-reverse** (itens na hor. organizados da direita para esquerda);
  - **flex-direction: column-reverse** (itens na vert. organizados de baixo para cima);
  - Em ambos os casos os itens são apresentados na ordem invertida.

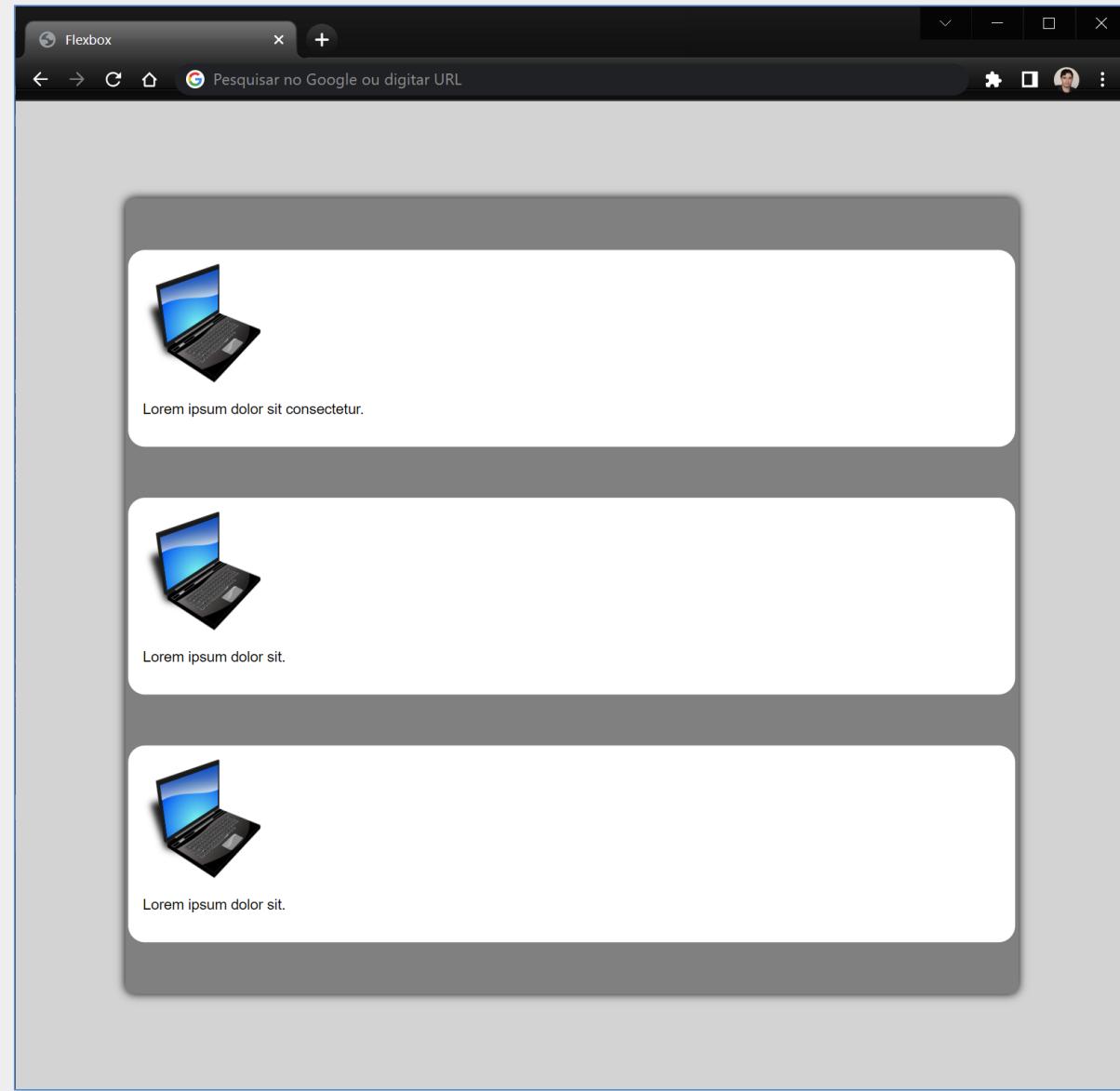
# Eixo principal na vertical: flex-direction: column

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



# justify-content com flex-direction: column

justify-content: **flex-start**



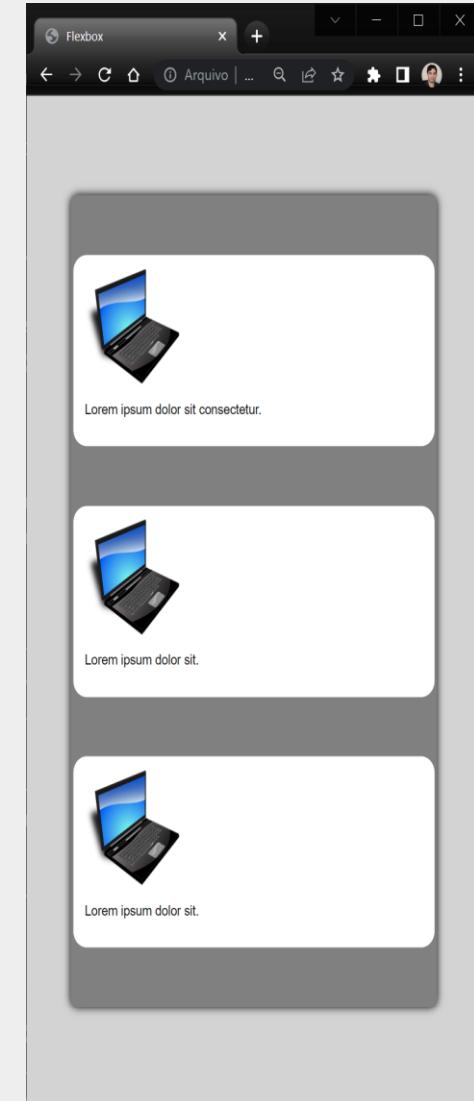
justify-content: **flex-end**



justify-content: **center**



justify-content: **space-evenly** justify-content: **space-between**



# Container de containers

HTML

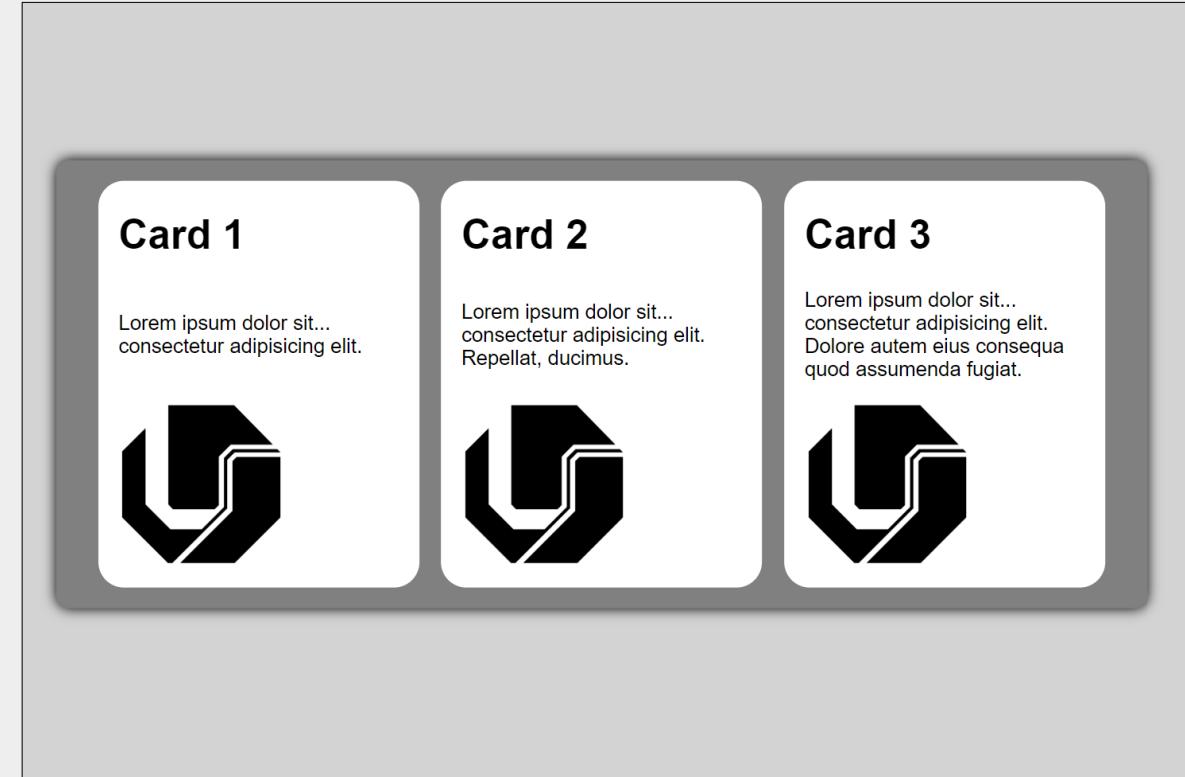
```
<div class="container">
  <div class="card">
    <h1>Card 1</h1>
    <p>Lorem ipsum dolor sit...
      
    </div>
  <div class="card">
    <h1>Card 2</h1>
    <p>Lorem ipsum dolor sit...
      
    </div>
  <div class="card">
    <h1>Card 3</h1>
    <p>Lorem ipsum dolor sit...
      
    </div>
  </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, cada **div** interno (**card**) também é um container flex, mas com **eixo principal na vertical**. Seus itens flexíveis são 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, que a imagem fique alinhada à base do card e o parágrafo alinhado ao centro do espaço vertical restante.

# Alinhamento no Eixo Transversal com align-items

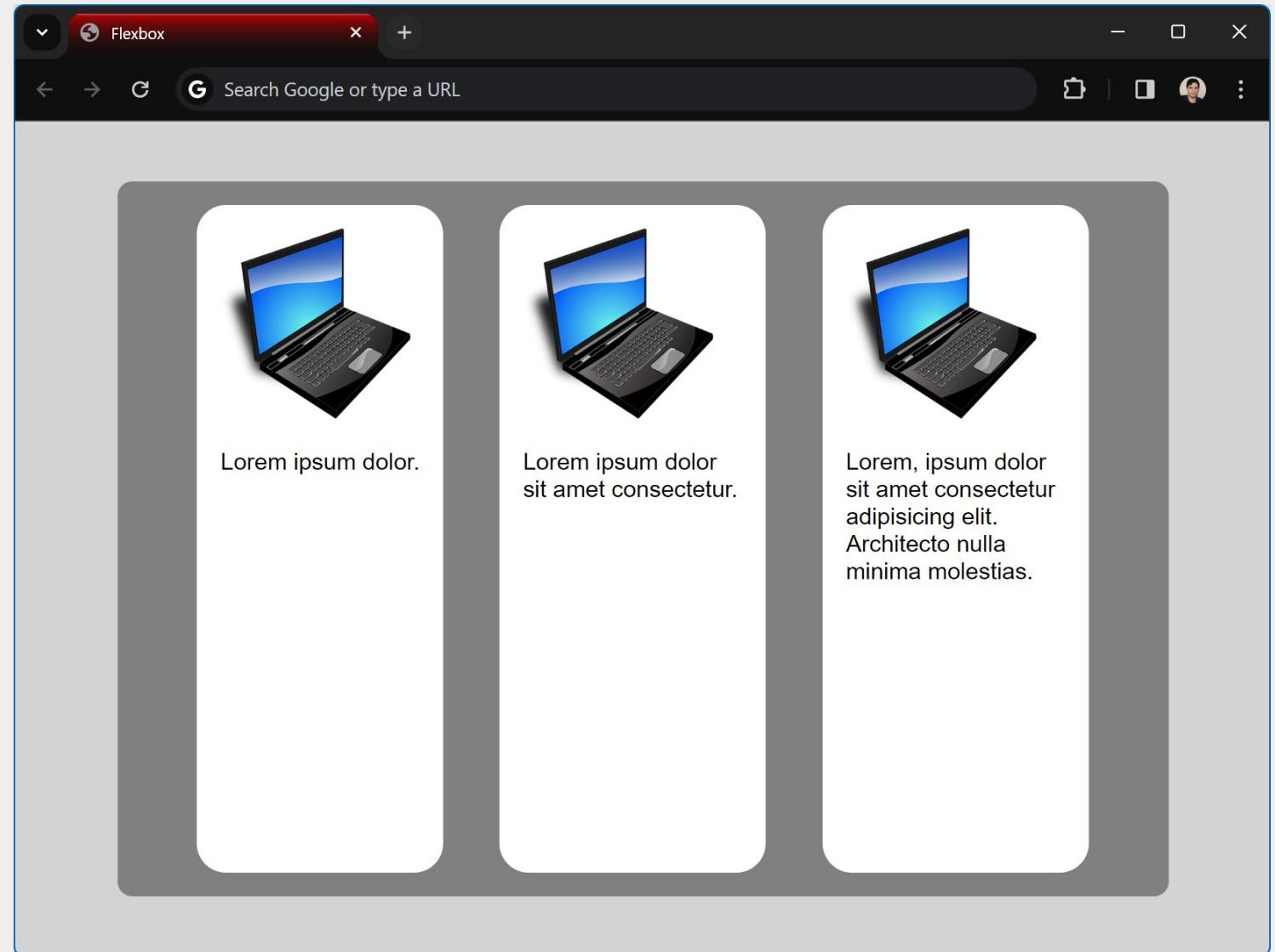
- Para alinhar os itens dentro de um container flex no **eixo transversal** pode-se utilizar a propriedade **align-items** no container.
- Há um total de 22 valores possíveis para **align-items**. Alguns deles são:
  - **stretch**
  - **flex-start**
  - **flex-end**
  - **center**
- O valor padrão de **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.

**OBS:** Quando o eixo principal está na horizontal e há múltiplas linhas de itens, **align-items** alinha os itens no eixo transversal em suas **respectivas** linhas.

# align-items com valor padrão: itens esticados no eixo transversal

```
.container {  
    background-color: gray;  
    padding: 1rem;  
    height: 80vh;  
    display: flex;  
    justify-content: space-evenly;  
}  
  
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
    max-width: 150px;  
}
```

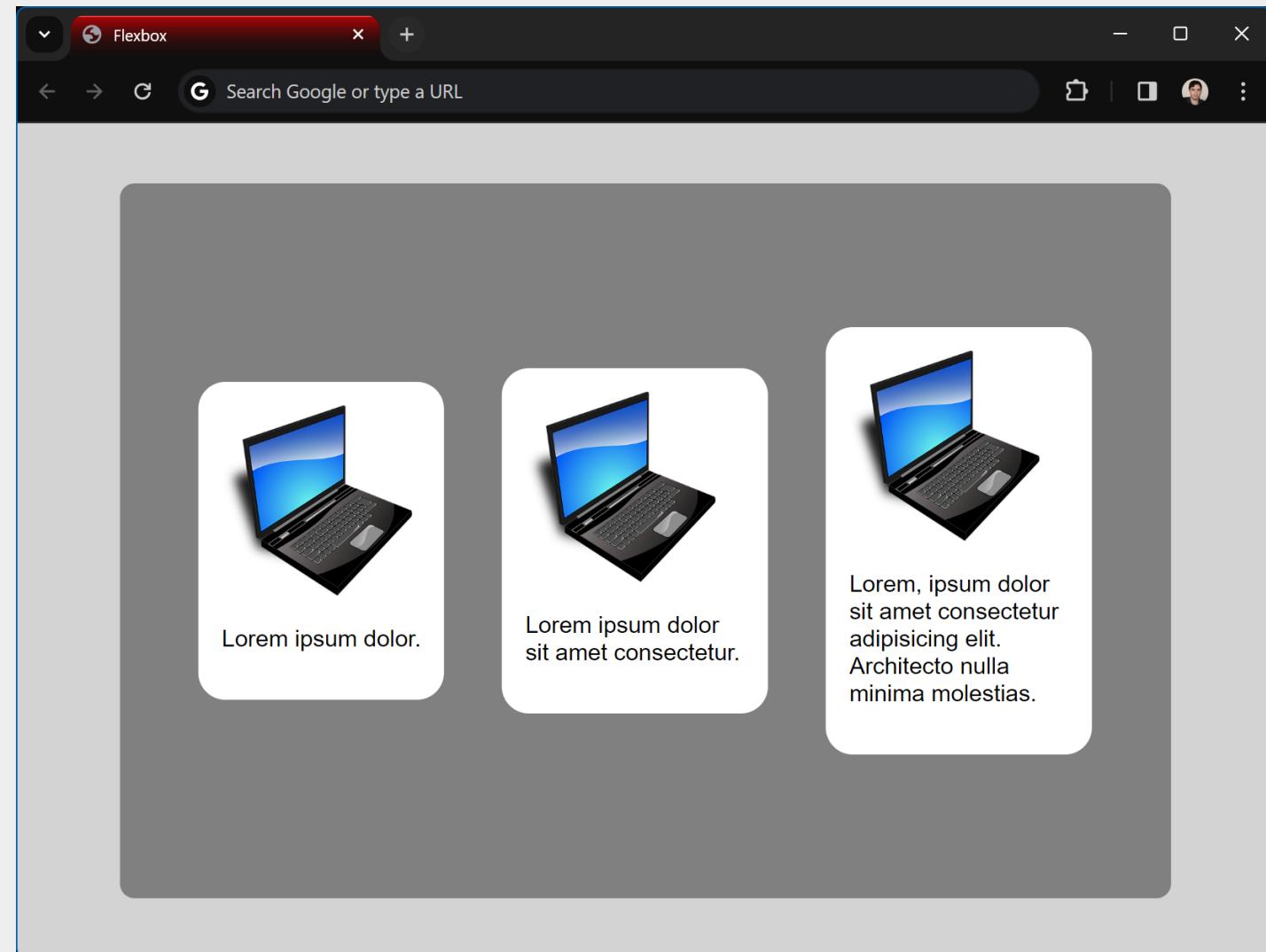
Neste exemplo o container possui uma altura definida de 80vh e os itens são esticados no eixo transversal para preencher toda essa altura, pois a propriedade `align-items` não foi alterada e portanto tem o valor padrão `stretch`.



# align-items: center

```
.container {  
    background-color: gray;  
    padding: 1rem;  
    height: 80vh;  
    display: flex;  
    justify-content: space-evenly;  
    align-items: center;  
}  
  
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
    max-width: 150px;  
}
```

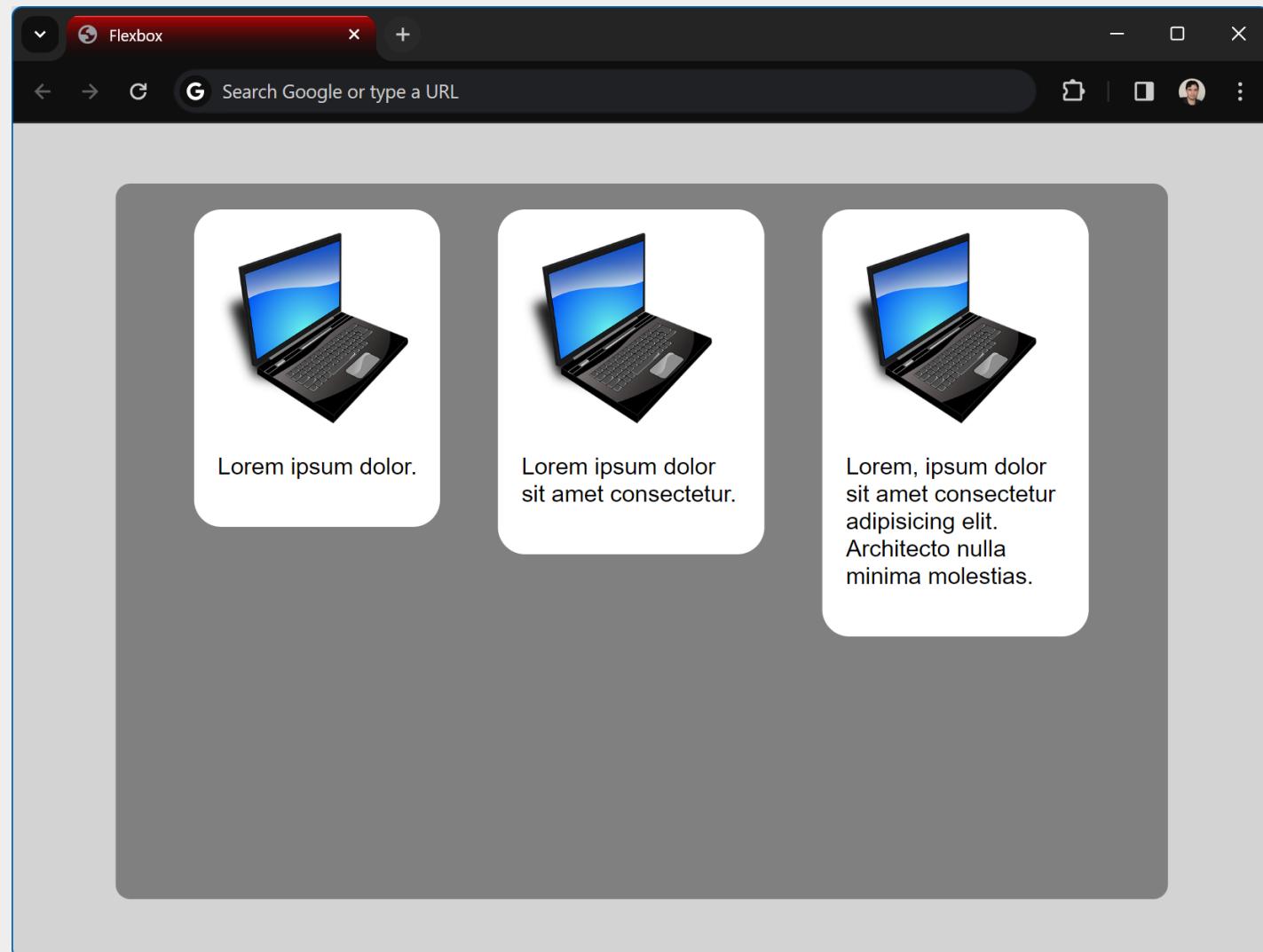
Neste exemplo os itens são alinhados ao centro no eixo transversal, pois foi utilizado `align-items: center` no container flexível.



# 

```
.container {  
    background-color: gray;  
    padding: 1rem;  
    height: 80vh;  
    display: flex;  
    justify-content: space-evenly;  
    align-items: flex-start;  
}  
  
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
    max-width: 150px;  
}
```

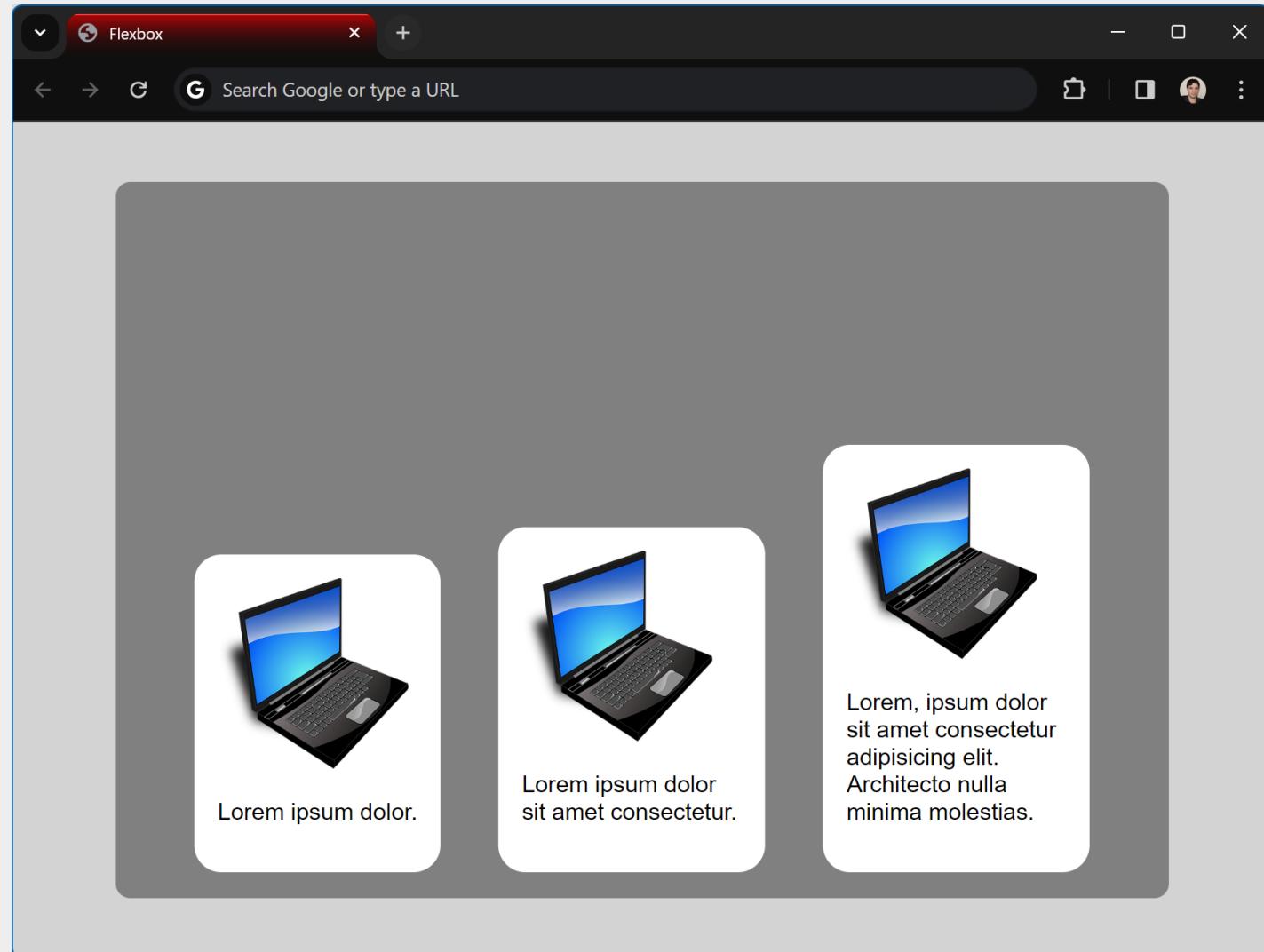
Neste exemplo os itens são alinhados no início do eixo transversal utilizando align-items: flex-start.



# align-items: flex-end

```
.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-radius: 20px;  
}
```

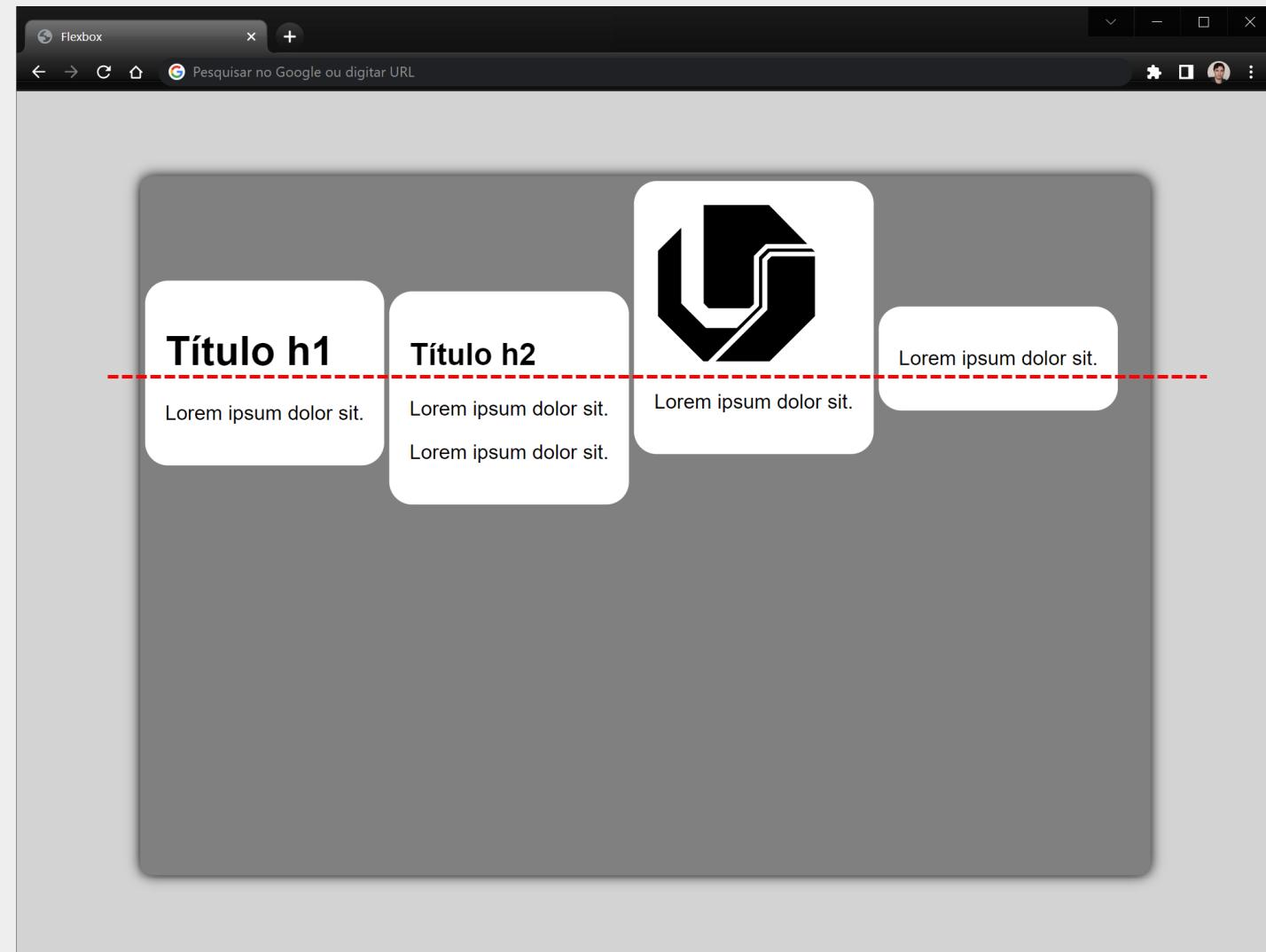
Neste exemplo os itens são alinhados no final do eixo transversal utilizando **align-items: flex-end**.



# 

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

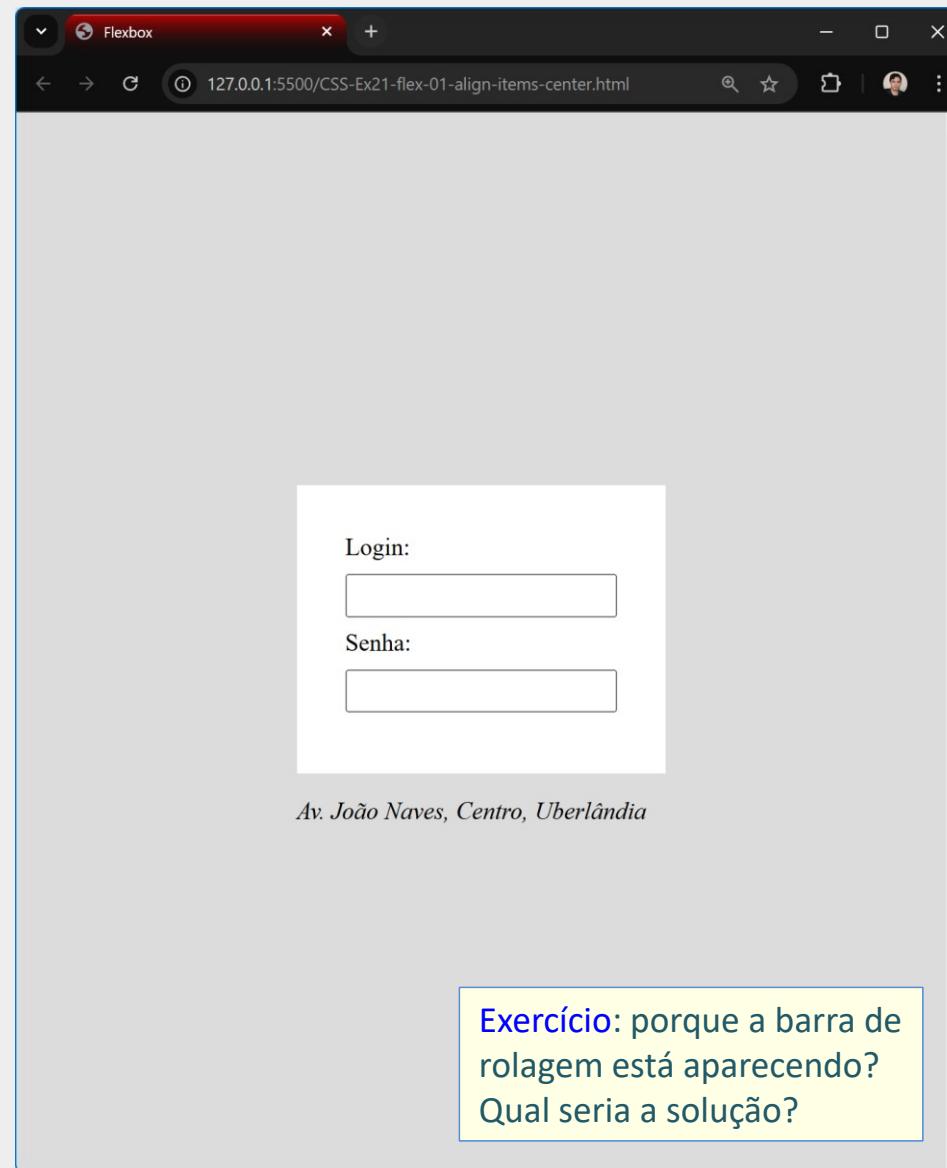
Neste exemplo os itens são alinhados no eixo transversal com base na primeira linha do conteúdo utilizando align-items: baseline. Exemplos adicionais:  
<https://developer.mozilla.org/en-US/docs/Web/CSS/align-items>



# Centralizando na Horizontal e Vertical com justify-content e align-items

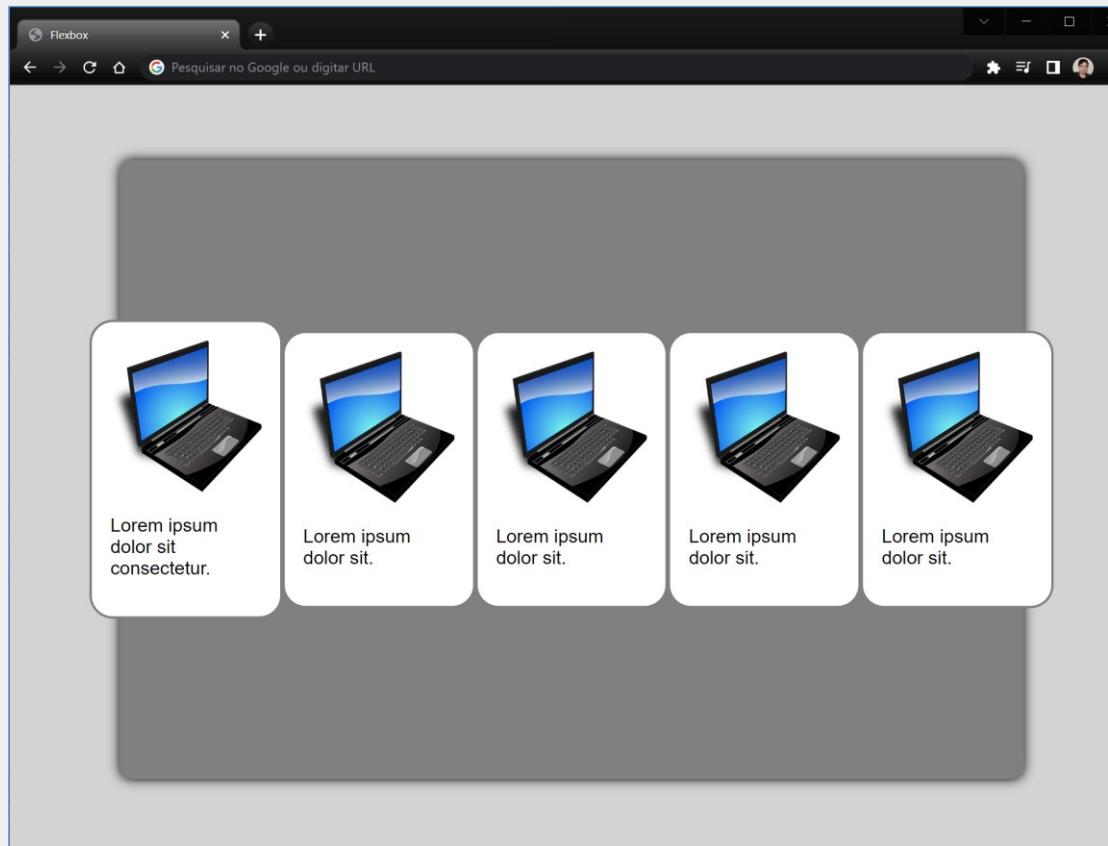
```
<style>
  body {
    min-height: 100vh;
    background-color: #gainsboro;
    display: flex;
    justify-content: center;
    align-items: center;
  }
  input {
    display: block;
    margin: 0.5rem 0;
    padding: 5px;
  }
  main {
    padding: 2rem;
    background-color: #white;
    margin-bottom: 1rem;
  }
</style>
</head>

<body>
  <div>
    <main>
      <label>Login: <input type="text"></label>
      <label>Senha: <input type="password"></label>
    </main>
    <footer>
      <address>Av. João Naves, Centro, Uberlândia</address>
    </footer>
  </div>
</body>
```



Exercício: porque a barra de rolagem está aparecendo?  
Qual seria a solução?

# Container flexível com overflow



Por padrão, os itens flexíveis são dispostos em uma única linha e ocorrerá *overflow* quando o container não comportar todos os itens, como mostrado neste exemplo.

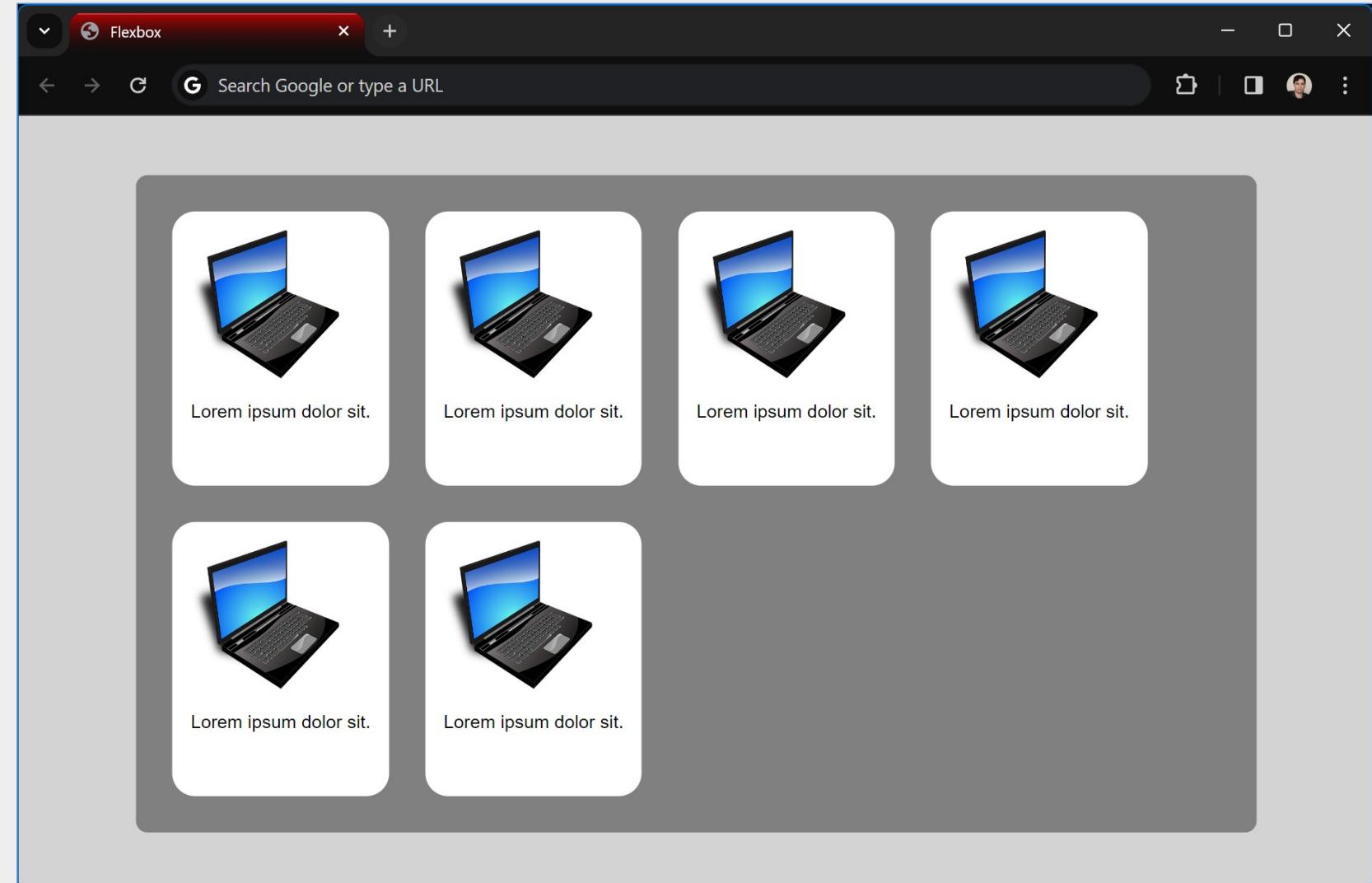
# Propriedade flex-wrap

- 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).
- Quando o eixo principal é **vertical** e o container tem altura definida, então **flex-wrap: wrap** permitirá a quebra de **coluna** caso os itens extrapolarem a altura do container.

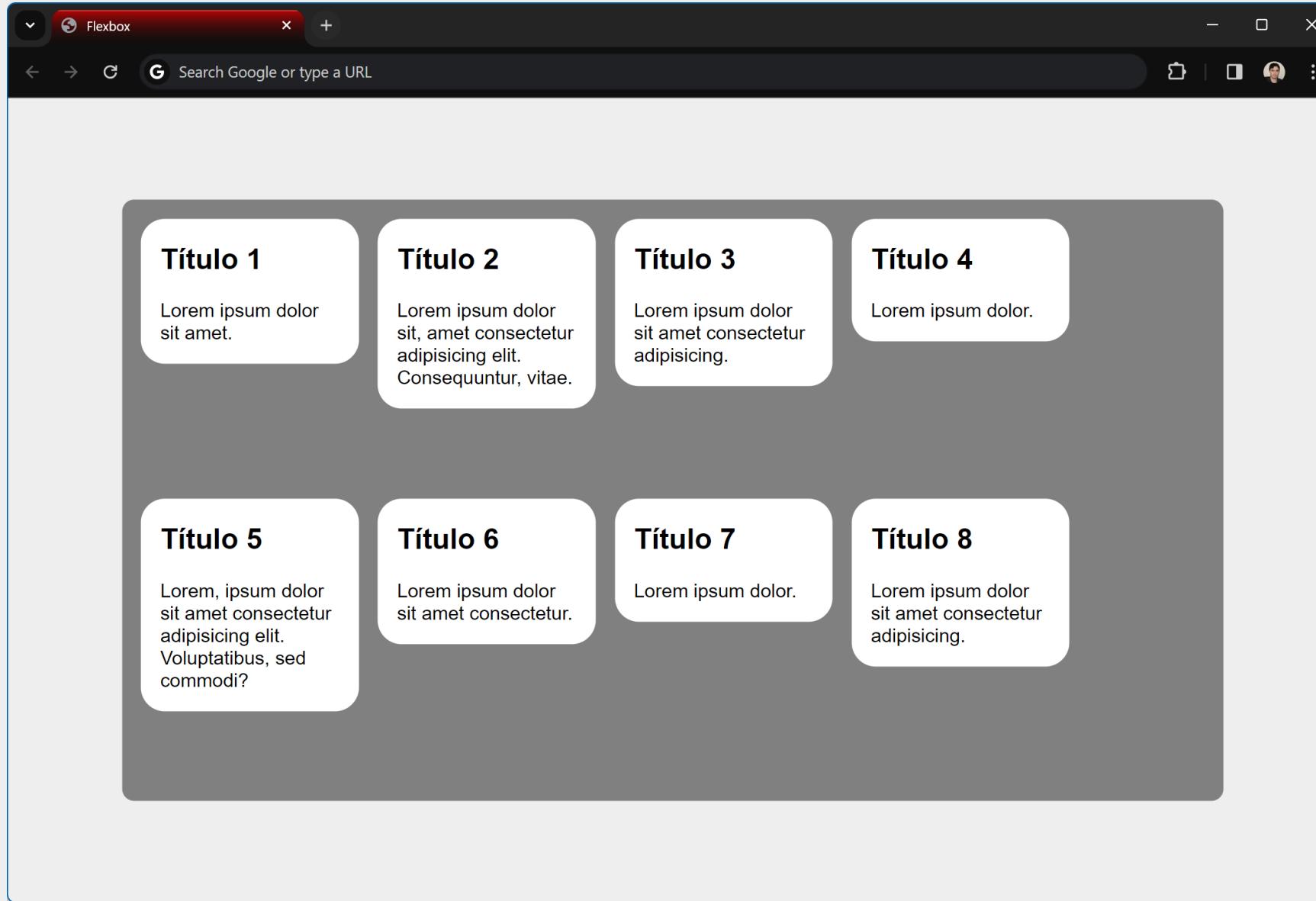
# Permitindo quebra de linha com flex-wrap: wrap

```
.container {  
    background-color: #gray;  
    padding: 1rem;  
    display: flex;  
    height: 80vh;  
    flex-wrap: wrap;  
}  
  
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
    margin: 1rem;  
}
```

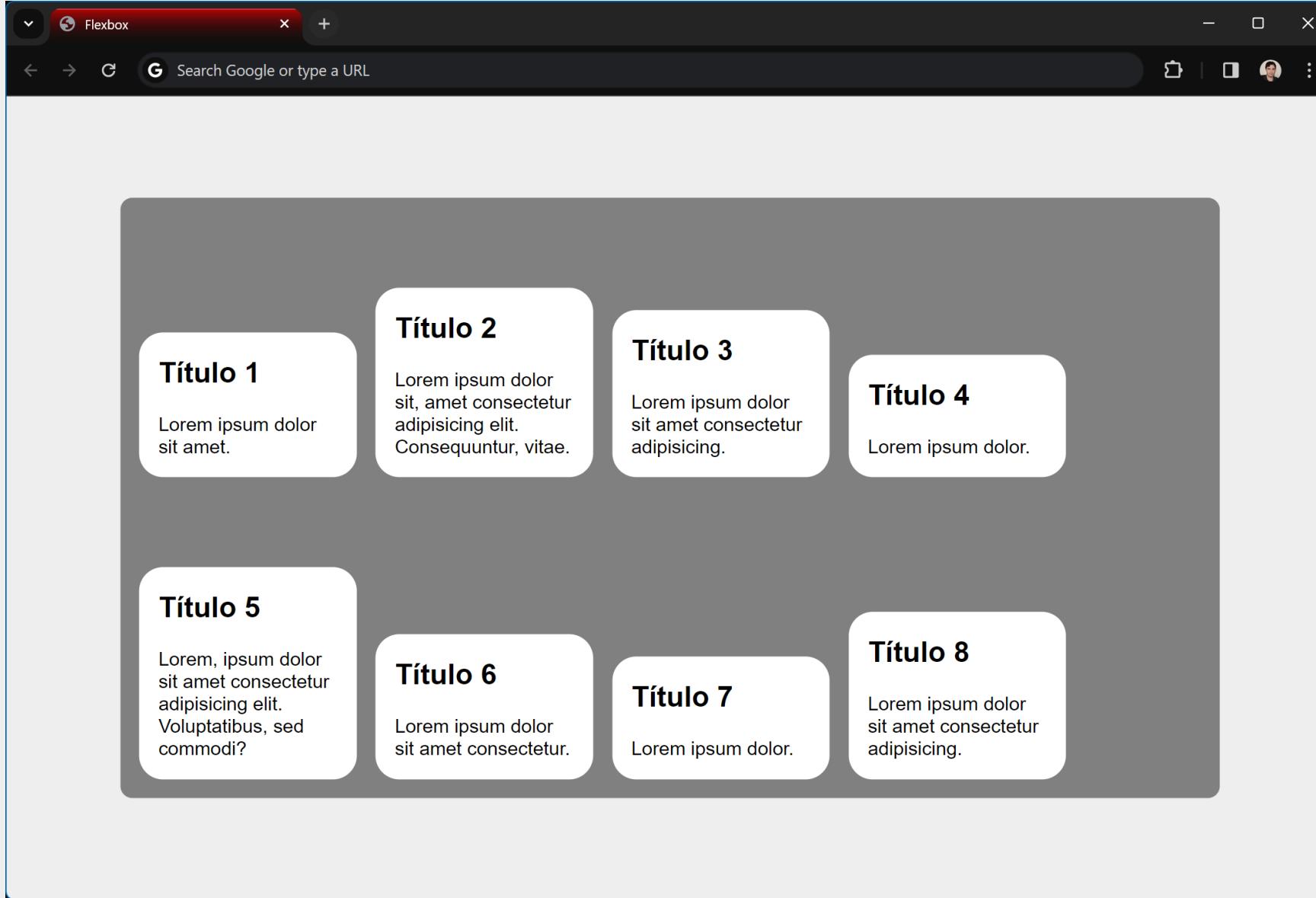
Múltiplas quebras de linha poderão ocorrer caso não haja espaço suficiente (por exemplo, em smartphones ou após redimensionar a janela do navegador, reduzindo a sua largura).



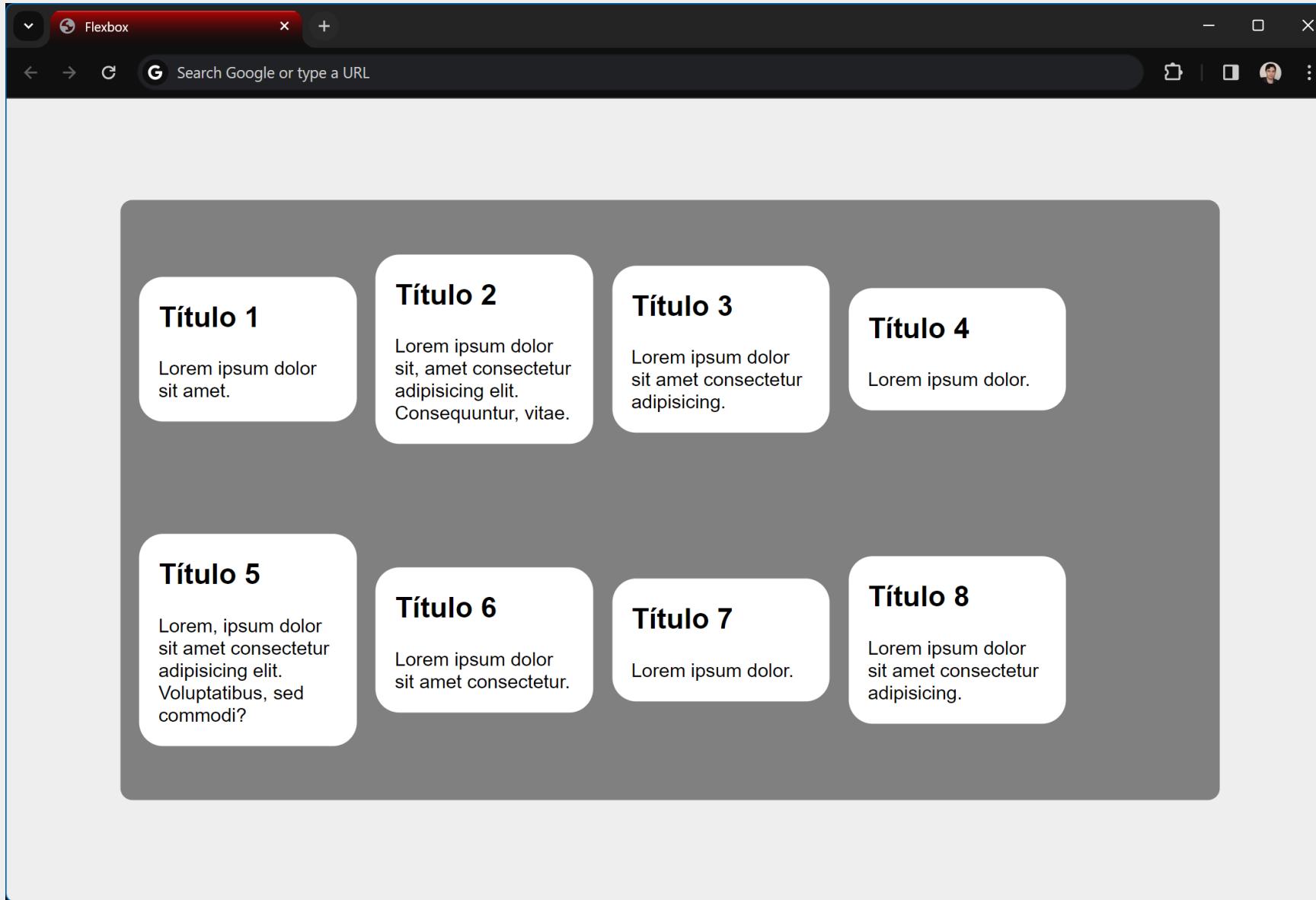
# flex-wrap: wrap com align-items: flex-start



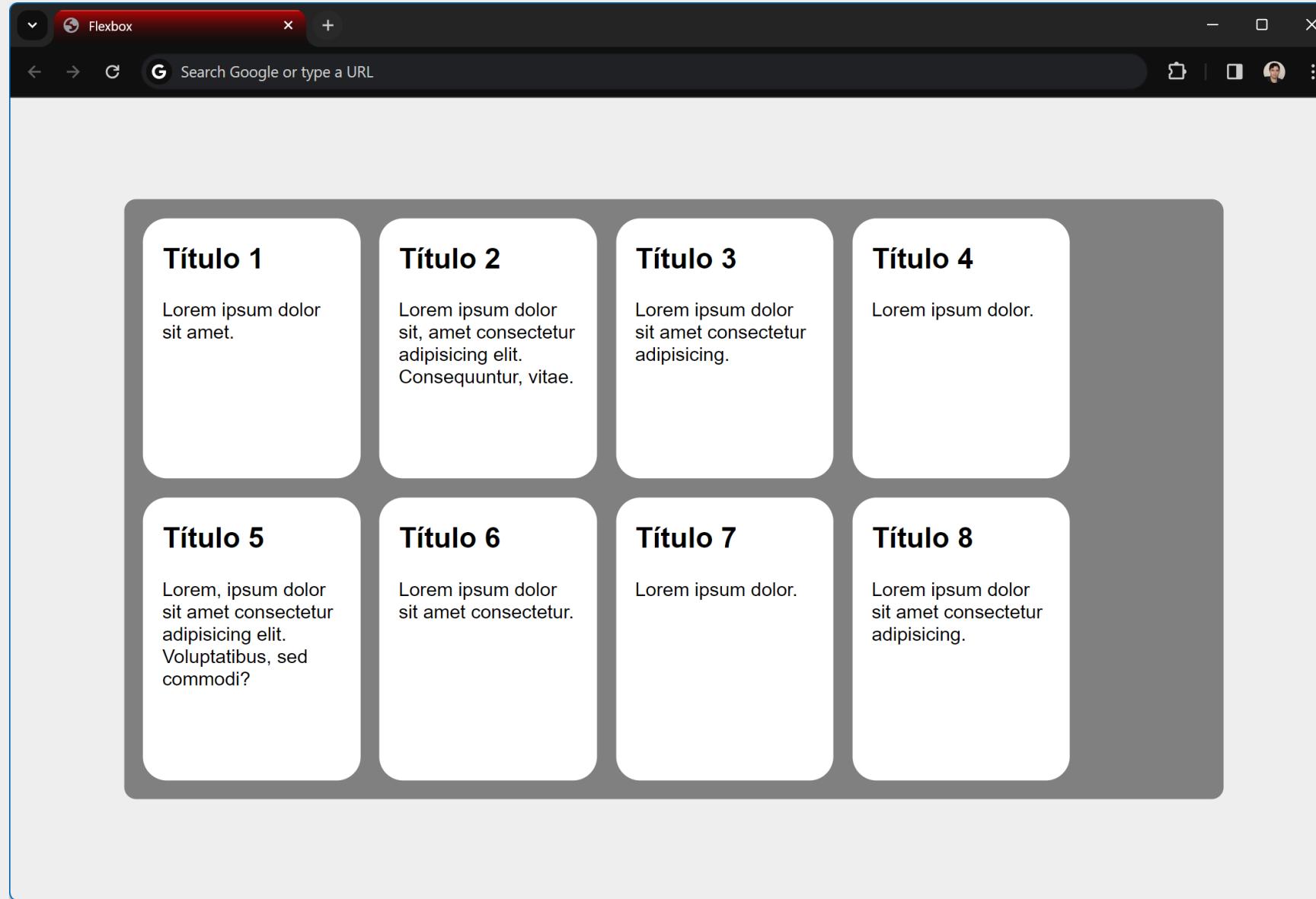
# flex-wrap: wrap com align-items: flex-end



# flex-wrap: wrap com align-items: center



# flex-wrap: wrap com align-items: stretch



# Propriedade align-content

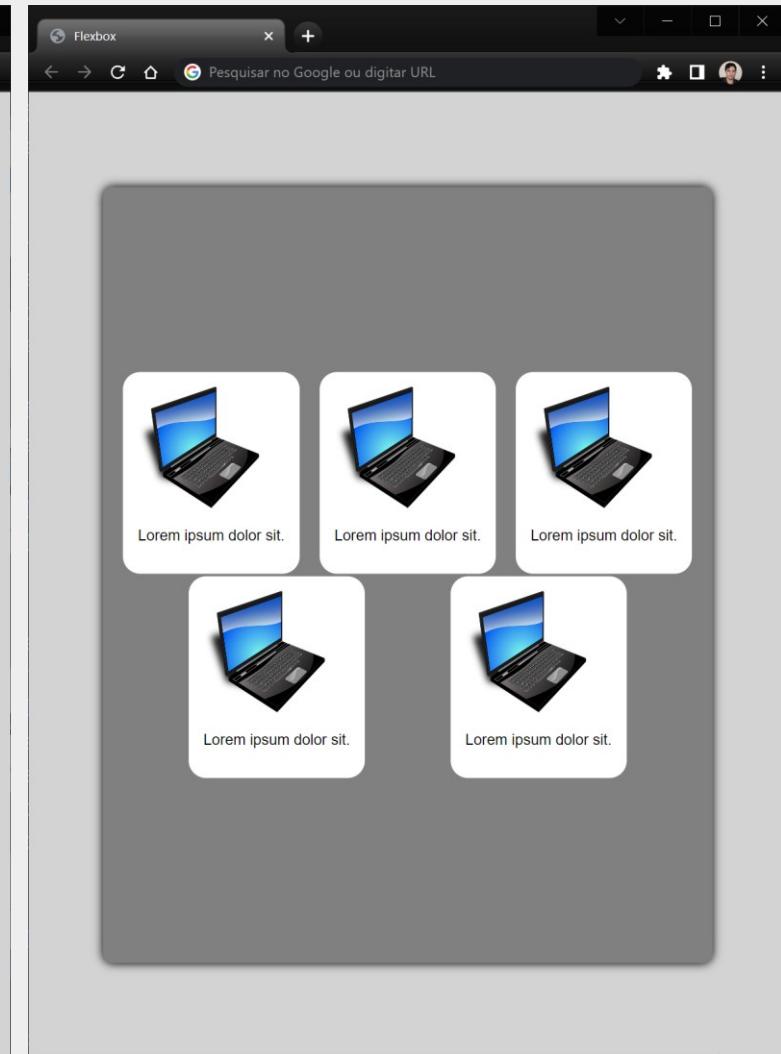
- Quando `flex-wrap` é definida para `wrap` é possível utilizar a propriedade `align-content` para ajustar o alinhamento do conjunto de itens no eixo transversal.
- Com `align-content` é possível ajustar o alinhamento de múltiplas linhas de uma forma que não é possível com `align-items` (veja exemplos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

# align-items vs align-content

Com `align-items: center`, os **itens** são alinhados verticalmente ao centro de sua **linha** de conteúdo.



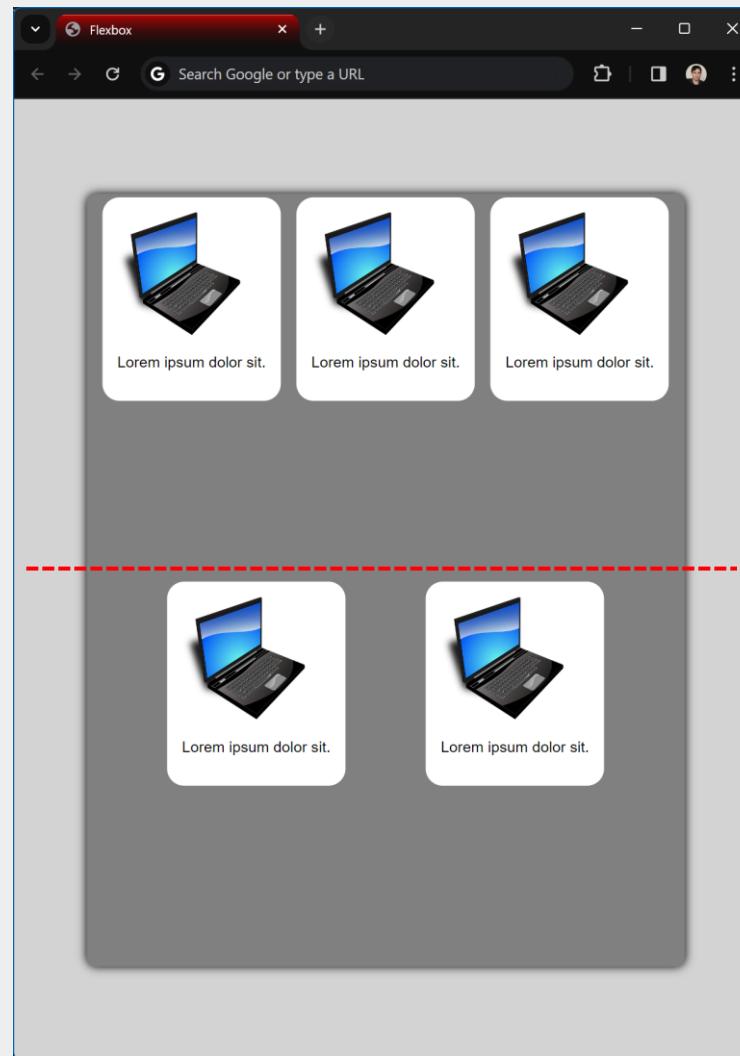
`align-items: center`



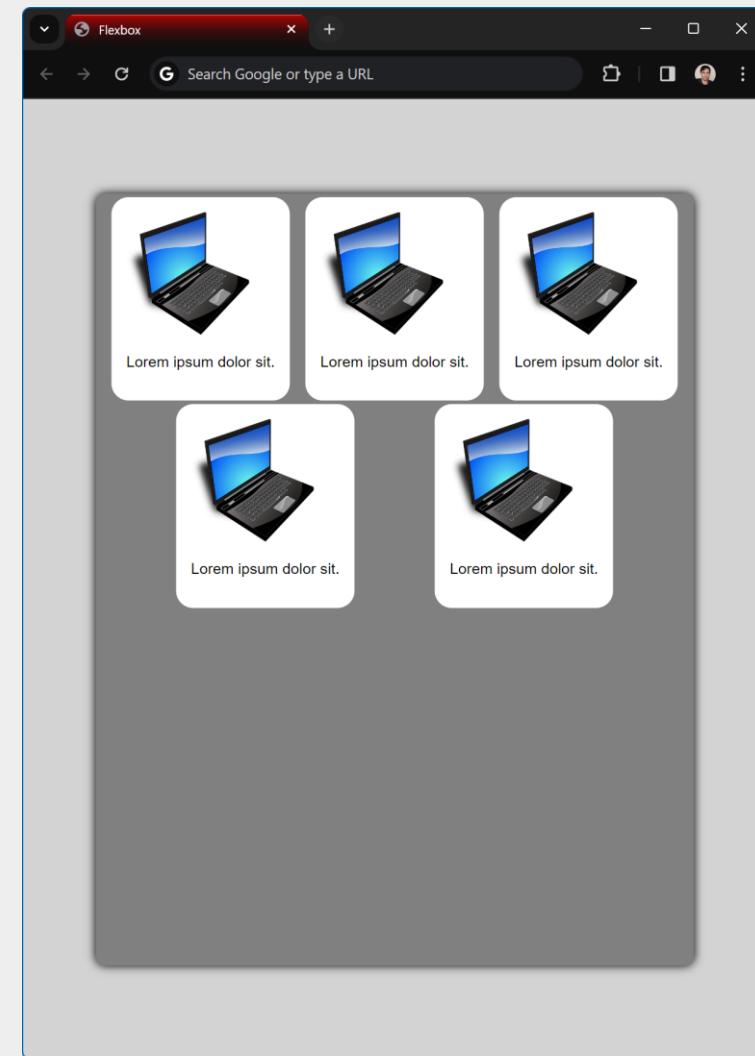
`align-content: center`

Com `align-content: center`, o **conjunto** de itens é alinhado verticalmente ao centro.

# align-items vs align-content

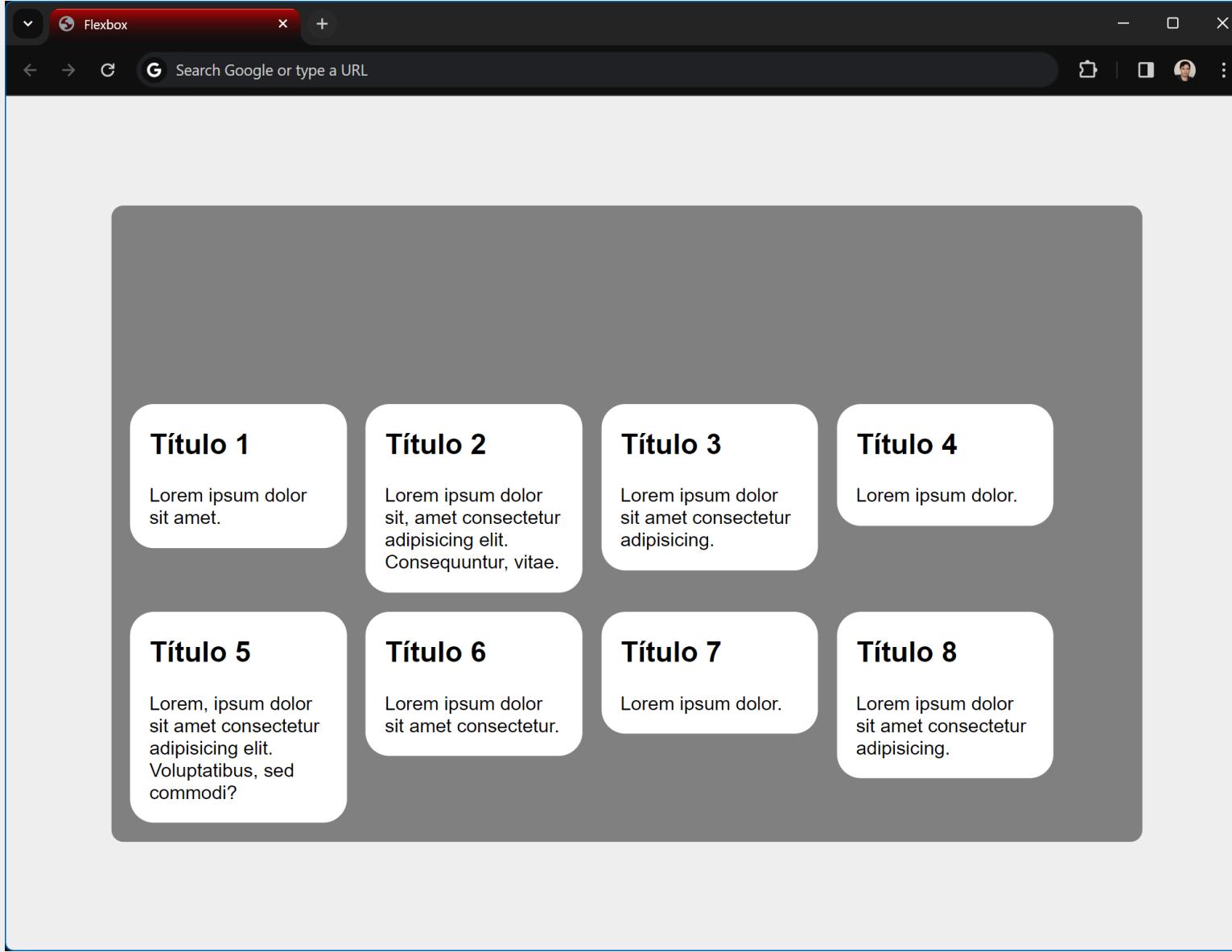


align-items: **flex-start**

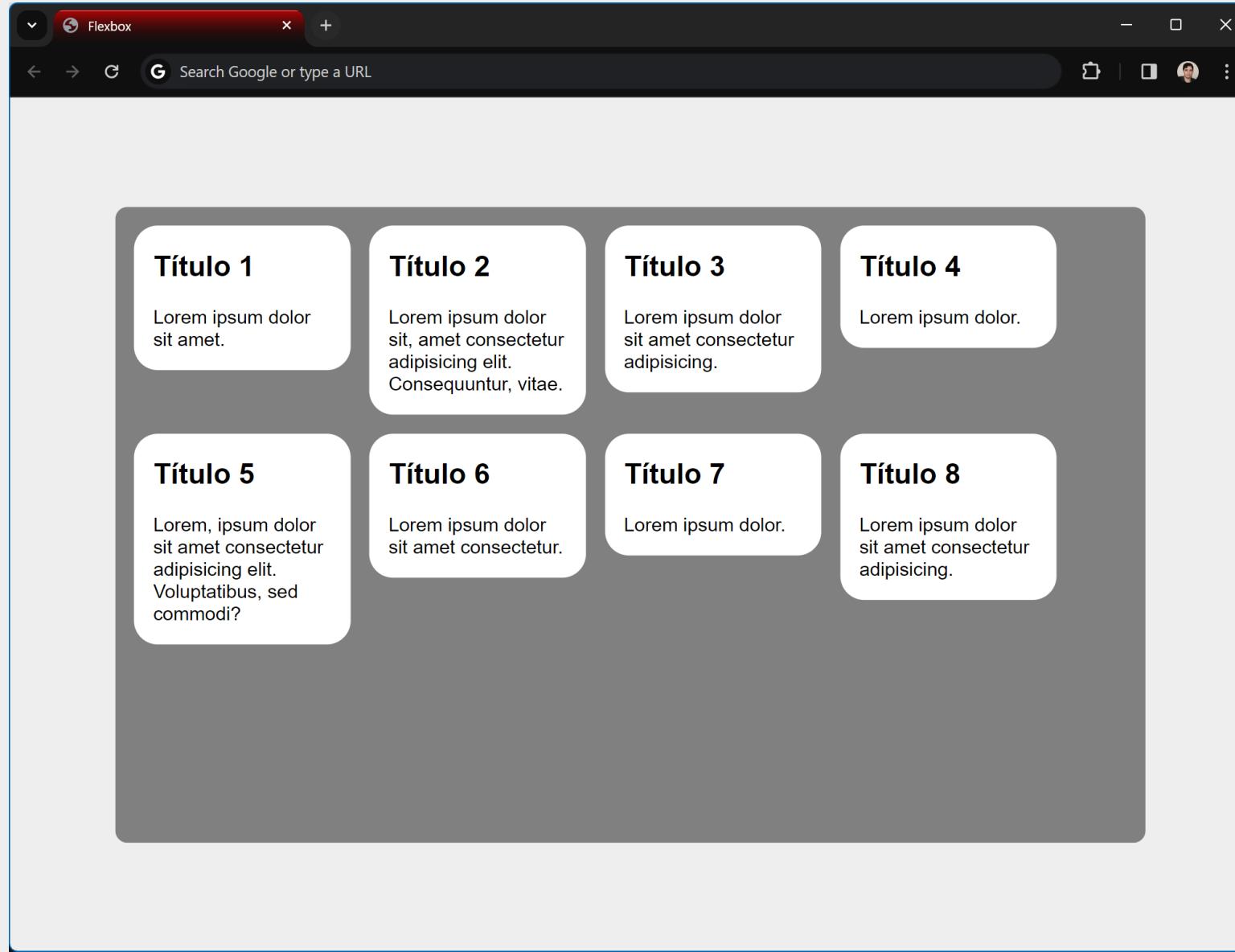


align-content: **flex-start**

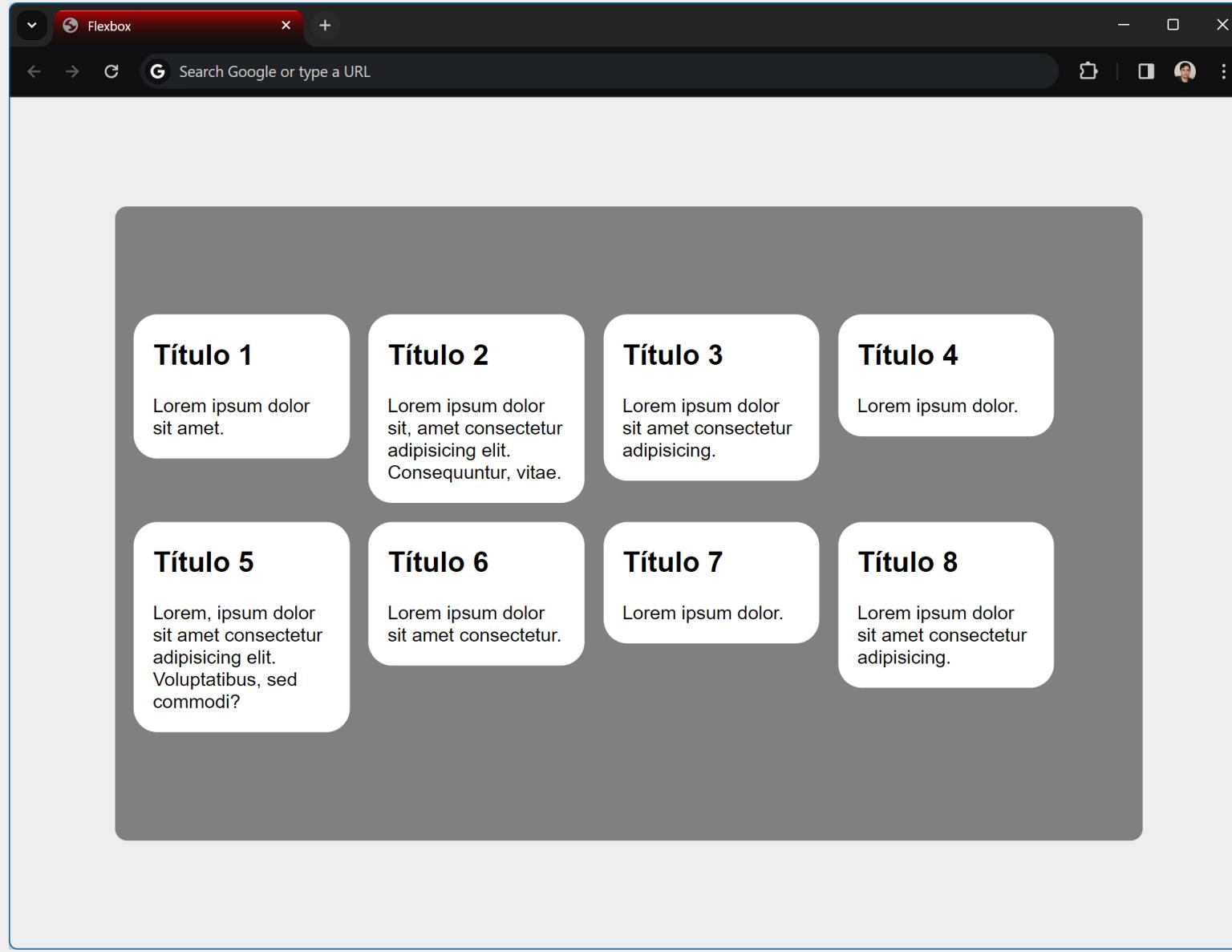
# align-content: flex-end com align-items: flex-start



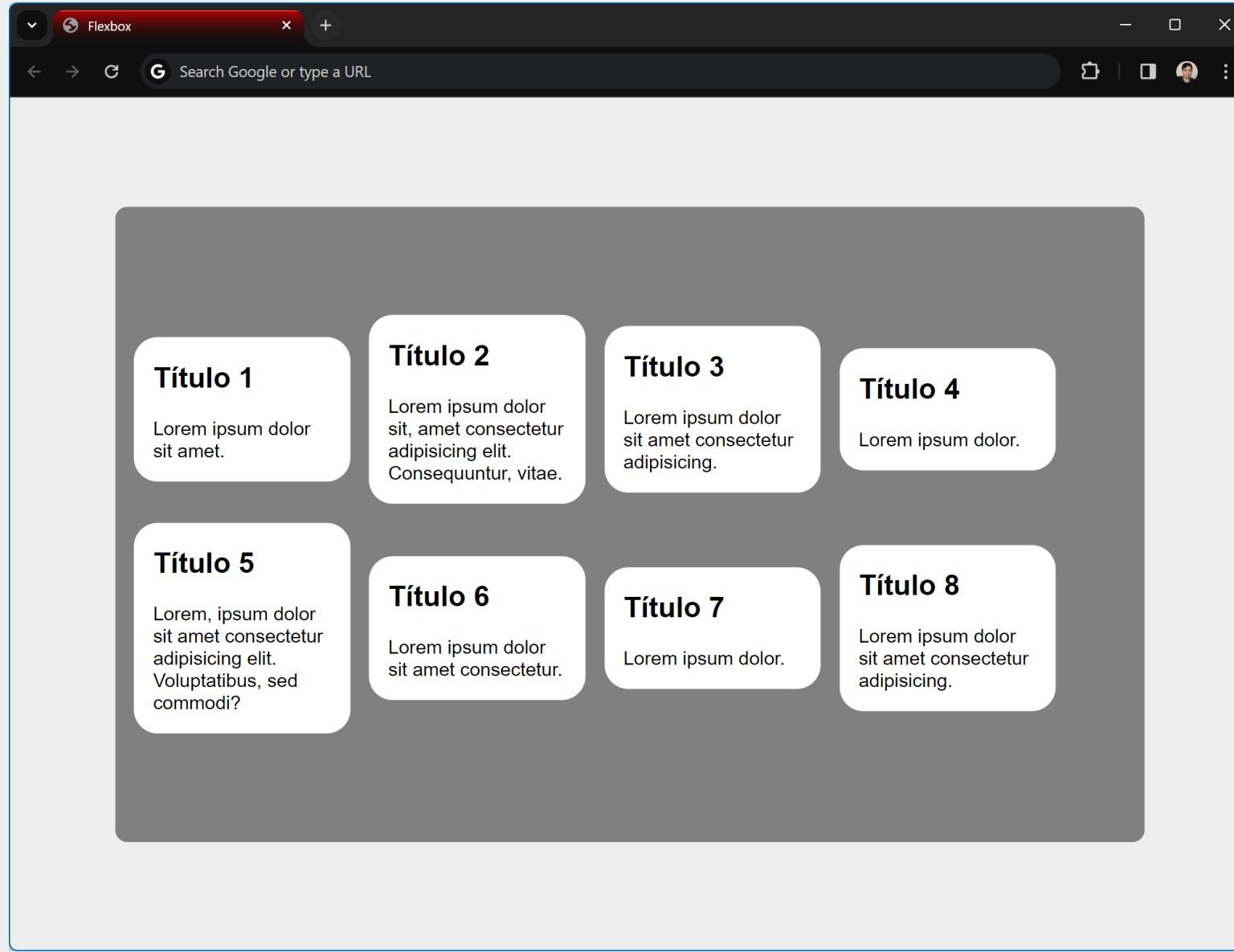
# align-content: flex-start com align-items: flex-start



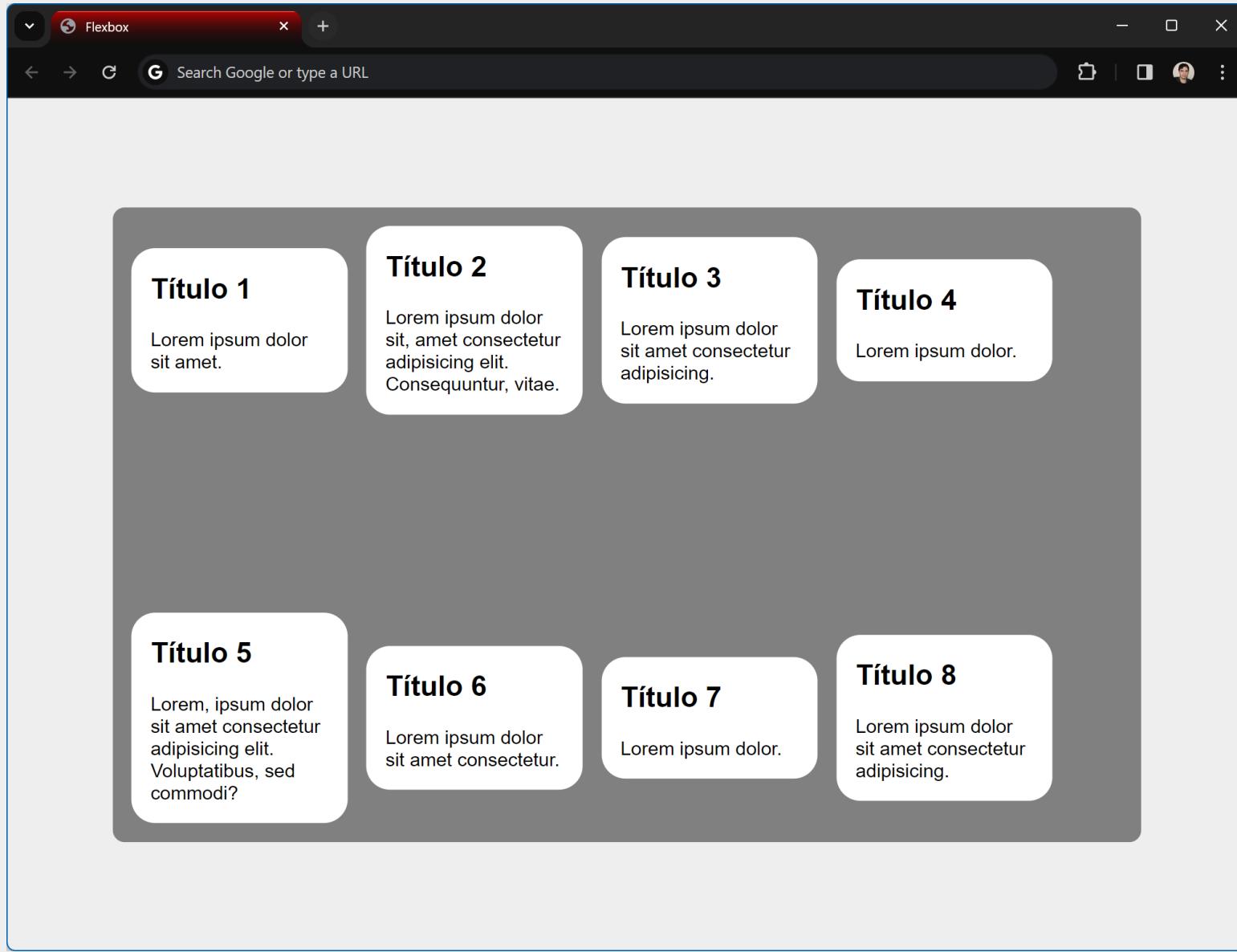
# align-content: center com align-items: flex-start



# align-content: center com align-items: center



# align-content: space-between com align-items: center



# Permitindo quebra de coluna com flex-wrap: wrap

```
.container {  
    background-color: gray;  
    padding: 1rem;  
    height: 80vh;  
    display: flex;  
    flex-direction: column;  
    flex-wrap: wrap;  
    justify-content: space-evenly;  
}
```

Quando o eixo principal é **vertical** e o container tem altura definida, então **flex-wrap: wrap** permitirá a quebra de coluna caso os itens extrapolem a altura do container.



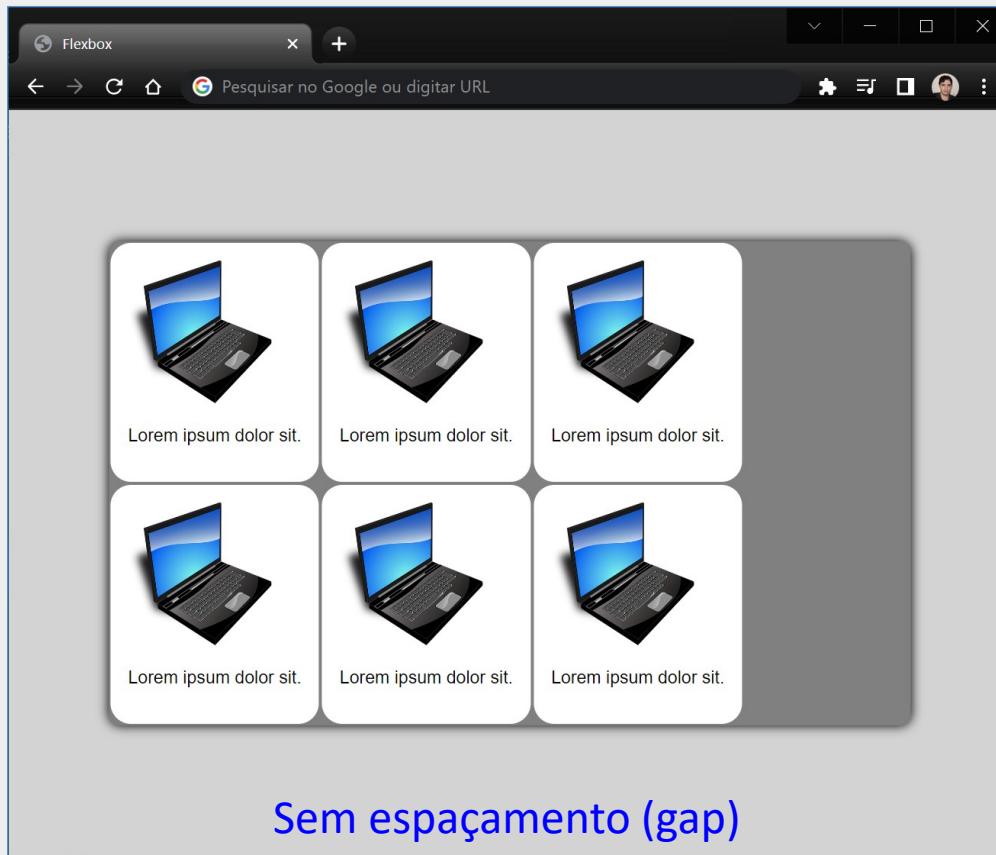
# Propriedade gap

- 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 espaçoamento nas linhas e nas colunas.
- Com dois valores (**gap: val1 val2**), o primeiro valor será utilizado como espaçoamento entre as linhas e o segundo, entre as colunas.
- **gap** é uma propriedade abreviada de **row-gap** e **column-gap**.

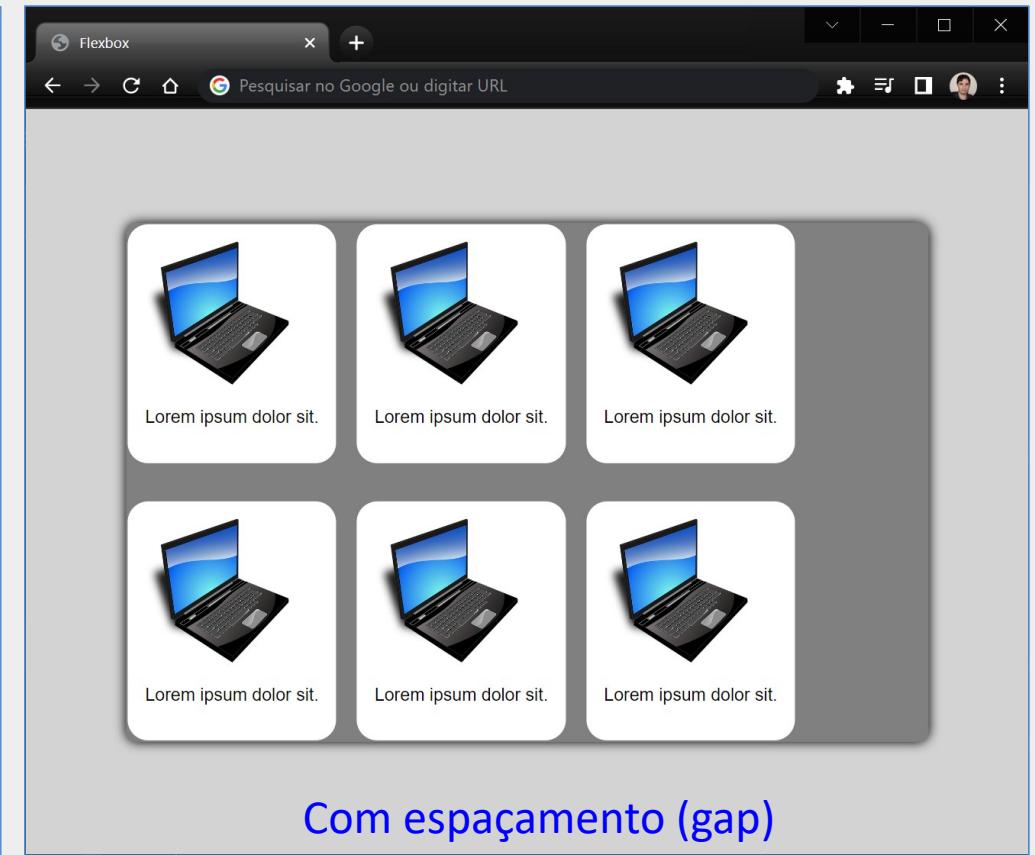
**OBS:** é importante enfatizar que a propriedade **gap** deve ser utilizada **no container flexível**, e não nos itens propriamente ditos. Além disso, a propriedade **margem**, mesmo que aplicada aos itens, não substitui a propriedade **gap** devido à questões dinâmicas associadas aos itens flexíveis, como quebras de linha etc.

# 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;  
}
```



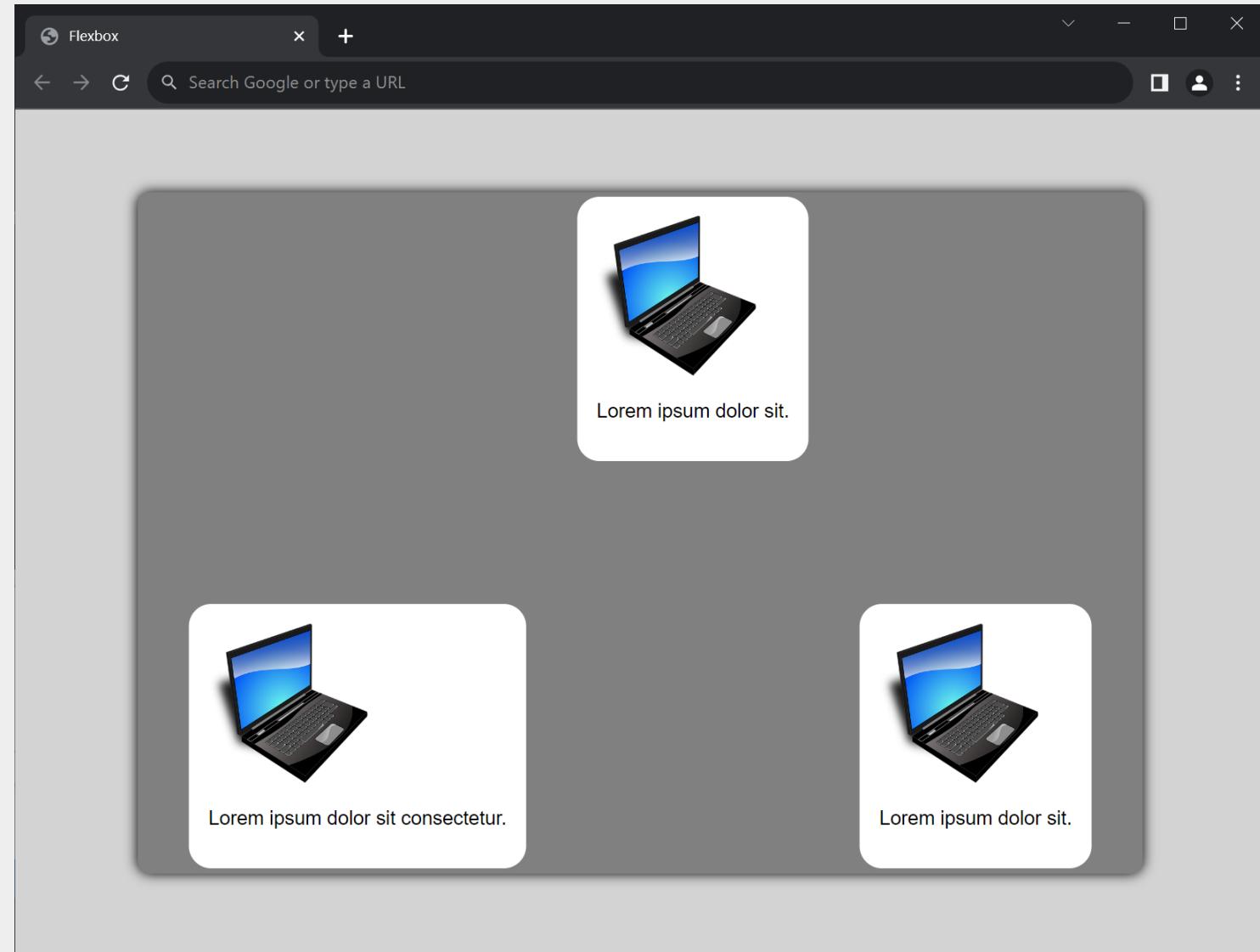
# Propriedades de Itens

- As propriedades apresentadas anteriormente são todas propriedades do **container** flexível. Portanto, devem ser utilizadas no **container**.
- Entretanto, 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 (tamanho base no eixo principal);
    - valor padrão: auto.
  - **flex**
    - propriedade abreviada que permite definir as três propriedades anteriores de uma vez.

# Propriedade align-self

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

Neste exemplo a propriedade `align-items: flex-end` é utilizada no container para alinhar os itens no **final** do eixo transversal. Entretanto, a propriedade `align-self` é utilizada apenas no segundo item para sobrepor esse valor e alinhá-lo no **início** do eixo transversal (no código HTML, o segundo item faz referência à classe CSS `.item-2`)



# flex-grow: 0 – Itens não expandindo

Título 1

Lorem ipsum dolor sit.

Título 2

Lorem ipsum dolor sit amet consectetur.

Título 3

Lorem ipsum dolor sit.

Neste exemplo, os três itens ocupam apenas a largura necessária para exibição do conteúdo, sem expandir para preencher o container, pois `flex-grow` tem seu valor padrão (`0`).

```
<div class="container">
  <div class="item">
    <h1>Título 1</h1><p>Lorem ipsum dolor
  </div>
  <div class="item">
    <h1>Título 2</h1><p>Lorem ipsum dolor
  </div>
  <div class="item">
    <h1>Título 3</h1><p>Lorem ipsum dolor
  </div>
</div>
```

```
.container {
  background-color: #gray;
  box-shadow: 0 0 10px;
  height: 70vh;
  display: flex;
  gap: 1rem;
}
```

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

# flex-grow: 1 – Itens expandindo igualmente

## Título 1

Lorem ipsum dolor sit.

## Título 2

Lorem ipsum dolor sit amet consectetur.

## Título 3

Lorem ipsum dolor sit.

Ao definir “flex-grow: 1” para todos os itens, eles passam a expandir igualmente para preencher o espaço disponível.

```
<div class="container">
  <div class="item">
    <h1>Título 1</h1><p>Lorem ipsum dolor
  </div>
  <div class="item">
    <h1>Título 2</h1><p>Lorem ipsum dolor
  </div>
  <div class="item">
    <h1>Título 3</h1><p>Lorem ipsum dolor
  </div>
</div>
```

```
.container {
  background-color: #gray;
  box-shadow: 0 0 10px;
  height: 70vh;
  display: flex;
  gap: 1rem;
}
```

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

# flex-grow – Item central expandindo mais que os laterais



```
<div class="container">
  <div class="item-left">...</div>
  <div class="item-center">...</div>
  <div class="item-right">...</div>
</div>
```

```
.item-left { flex-grow: 1; }
.item-center { flex-grow: 5; }
.item-right { flex-grow: 1; }
```

Neste exemplo, o item da esquerda e o item da direita recebem `flex-grow: 1`, enquanto o item central recebe `flex-grow: 5`. Isso permite que o item central expanda pelo espaço disponível ocupando uma fatia desse espaço 5x maior do que aquela ocupada pelos itens laterais.

# flex-shrink: 1 – Itens encolhendo igualmente



```
.container {  
    background-color: #gray;  
    box-shadow: 0 0 10px;  
    height: 70vh;  
    display: flex;  
    gap: 1rem;  
}
```

```
.item {  
    padding: 1rem;  
    border-radius: 20px;  
    background-color: #eee;  
    flex-shrink: 1;  
}
```

Neste exemplo, os três itens tem o valor padrão `flex-shrink: 1` e a largura da janela foi reduzida ao mínimo. Repare que os itens encolheram o máximo possível e ficaram com largura limitada à palavra “Título”

# flex-shrink com valores individuais



```
.item-left { flex-shrink: 0; }
.item-center { flex-shrink: 3; }
.item-right { flex-shrink: 1; }
```

Neste exemplo, ao reduzir a largura da janela, o item da esquerda não encolhe (`flex-shrink: 0`), o item central encolhe mais rapidamente (`flex-shrink: 3`) e o item da direita encolhe mais lentamente (`flex-shrink: 1`).

# Propriedade flex-basis

- **flex-basis** define o **tamanho principal inicial** do item.
- Quando o eixo principal é horizontal, esse tamanho é a **largura (width)**.
- Quando o eixo principal é vertical, esse tamanho é a **altura (height)**.
- A expansão dos itens ocorre (conforme **flex-grow**) quando o tamanho do item se torna maior que **flex-basis**.
- O encolhimento ocorre (conforme **flex-shrink**) quando o tamanho do item se torna inferior a **flex-basis**.
- Ao definir **flex-basis** para um valor inferior a **min-content**, então o tamanho do item será ajustado para o tamanho mínimo de seu conteúdo.
- **width** poderá ser ignorada se definida em conjunto com **flex-basis**, uma vez que o tamanho do item é calculado dinamicamente.

# flex-basis: 0 com flex-grow: 0

## Programação

Lorem ipsum dolor sit consectetur.

Lorem ipsum dolor sit consectetur.

Lorem ipsum dolor sit consectetur.



Lorem ipsum dolor sit.

```
.item {  
background-color: white;  
padding: 1rem;  
border-radius: 20px;  
flex-basis: 0;  
}
```

Neste exemplo os itens foram definidos com `flex-basis: 0`, enquanto a propriedade `flex-grow` é mantida em seu valor padrão (0, não expandir). Repare que eles aparecem com o menor tamanho possível, equivalente à mínima largura do conteúdo (`min-content`).

# flex-basis: 0 com flex-grow: 1

## Programação

Lorem ipsum dolor sit consectetur.

Lorem ipsum dolor sit consectetur.

Lorem ipsum dolor sit consectetur.



```
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
    flex-basis: 0;  
    flex-grow: 1;  
}
```

Neste exemplo os itens foram definidos com **flex-basis: 0** e **flex-grow: 1**. Como há espaço suficiente no container, essa combinação permite que os itens expandam e ocupem o **mesmo tamanho** no eixo principal.

# flex-basis: 100%

## Programação

Lorem ipsum dolor sit consectetur.

Lorem ipsum dolor sit consectetur.

Lorem ipsum dolor sit consectetur.



```
.item {  
    background-color: white;  
    padding: 1rem;  
    border-radius: 20px;  
    flex-basis: 100%;  
}
```

Outra forma de obter um resultado similar ao do exemplo anterior é definindo **flex-basis: 100%** em todos os itens.

# Propriedade abreviada flex

- **flex** é uma propriedade abreviada de **flex-grow**, **flex-shrink** e **flex-basis**.
- Isto significa que com **flex** é possível definir, de uma só vez, os valores das três propriedades constituintes.
- 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 faz com que eles sejam definidos automaticamente (neste caso, **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 tamanhos iguais no container.

**OBS:** não confunda a propriedade **flex** com o valor **flex** da propriedade **display**. A propriedade **flex** deve ser utilizada nos **itens flexíveis**. A propriedade **display** com o valor **flex** deve ser utilizada no **container** para torná-lo flexível.

# flex: 1

## Programação

  Lorem ipsum dolor sit consectetur.

  Lorem ipsum dolor sit consectetur.

  Lorem ipsum dolor sit consectetur.



  Lorem ipsum dolor sit.



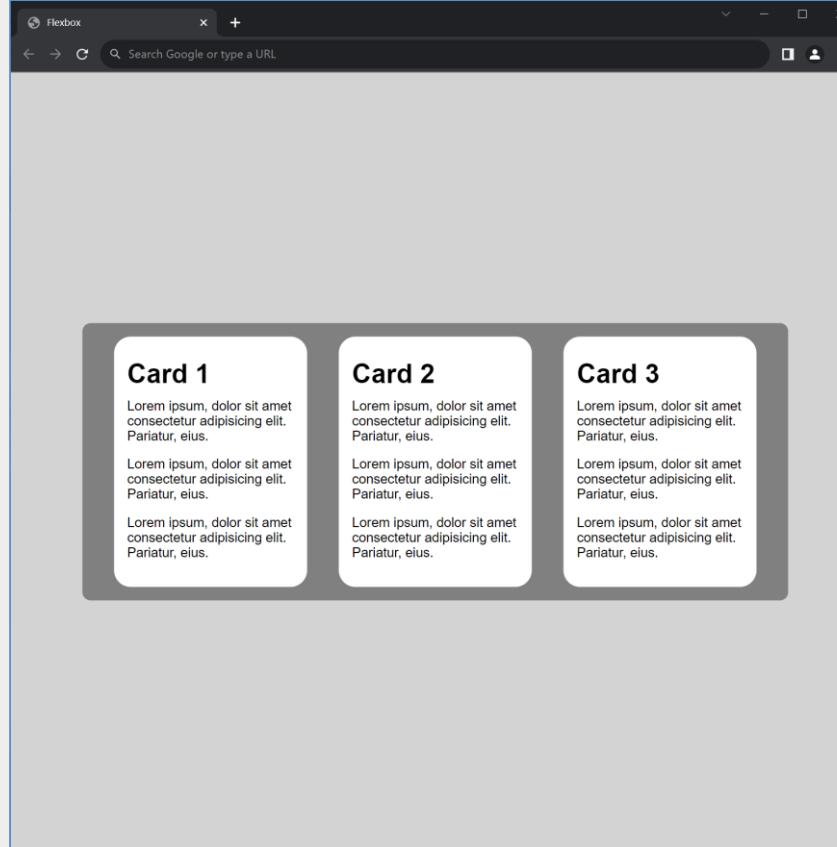
  Lorem ipsum dolor sit.

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

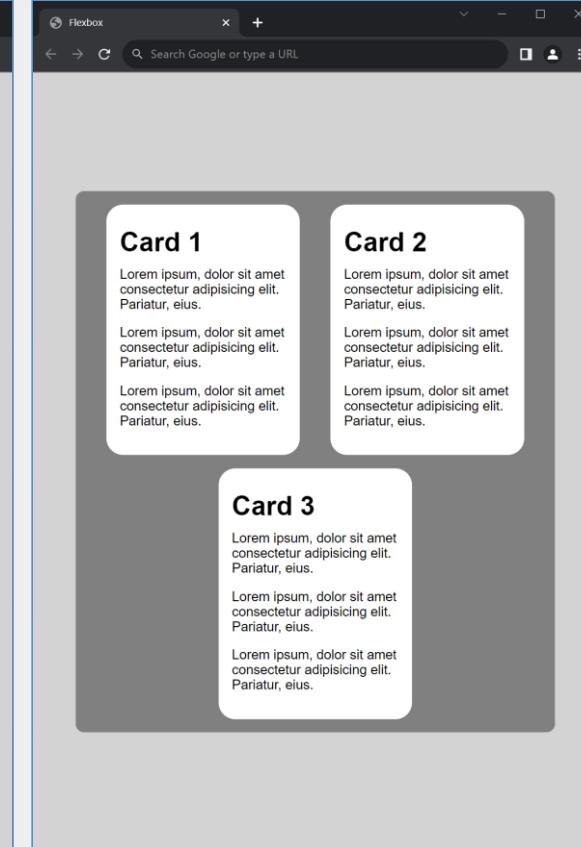
flex: 1 tem o mesmo efeito que “flex-grow: 1, flex-basis: 0”, permitindo aos itens se expandirem e ocuparem espaços iguais no container flexível (quando há espaço suficiente)

# Exemplo de Design Flexível – Itens sem expandir

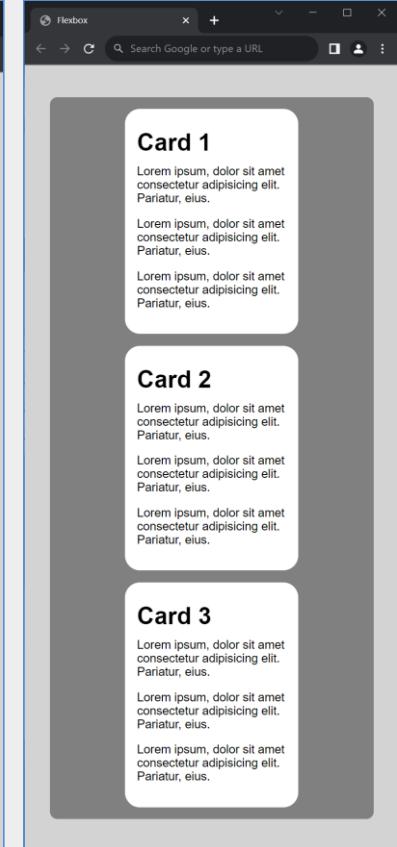
```
.container {  
    background-color: gray;  
    padding: 1rem;  
    display: flex;  
    flex-wrap: wrap;  
    justify-content: space-evenly;  
    gap: 1rem;  
}  
  
.card {  
    background-color: white;  
    border-radius: 20px;  
    padding: 1rem;  
    flex: 0 1 200px;  
}
```



Janela com largura "normal"



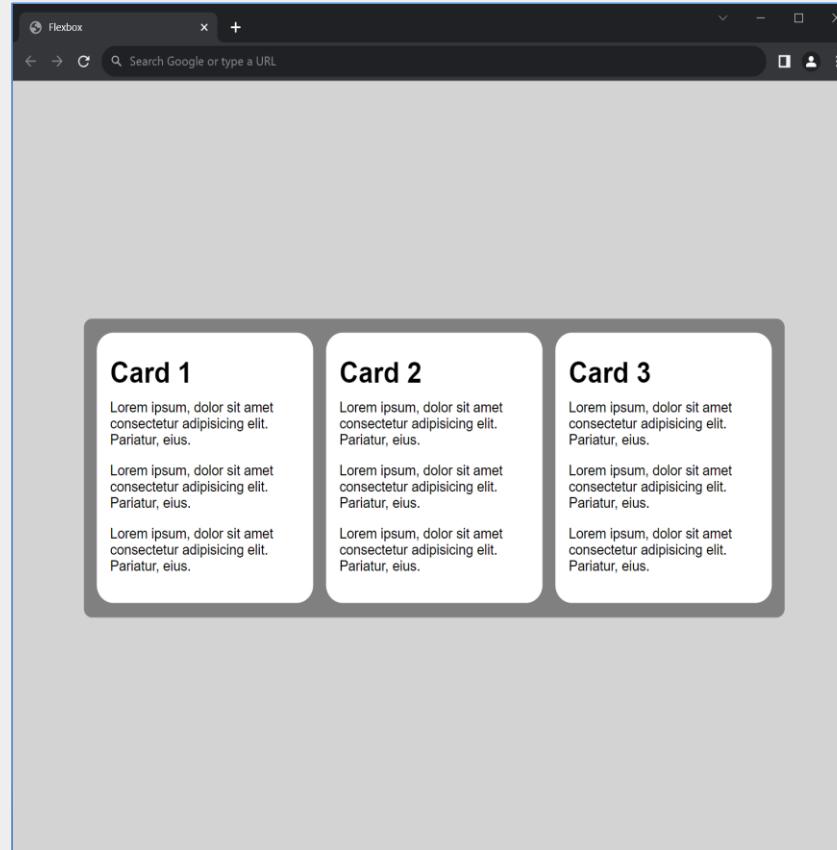
Janela com largura reduzida



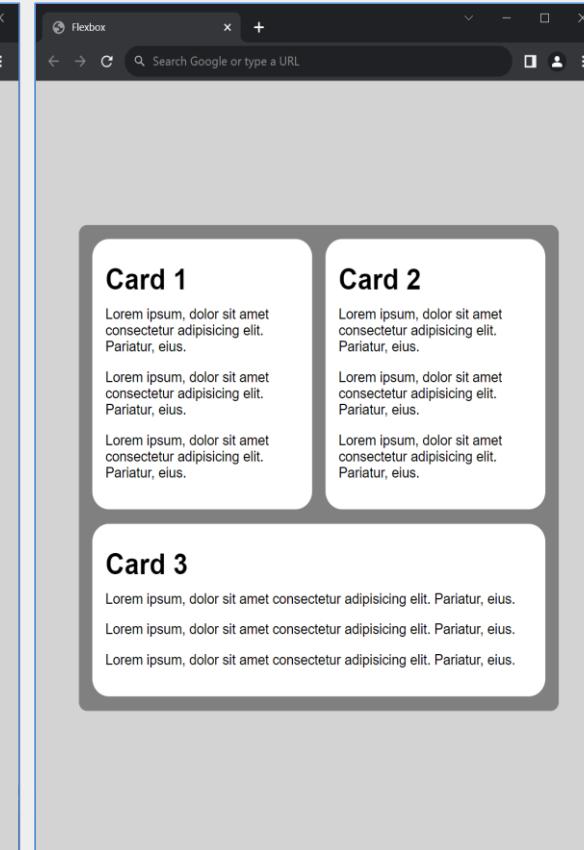
Janela com largura mínima

# Exemplo de Design Flexível – Itens expandindo

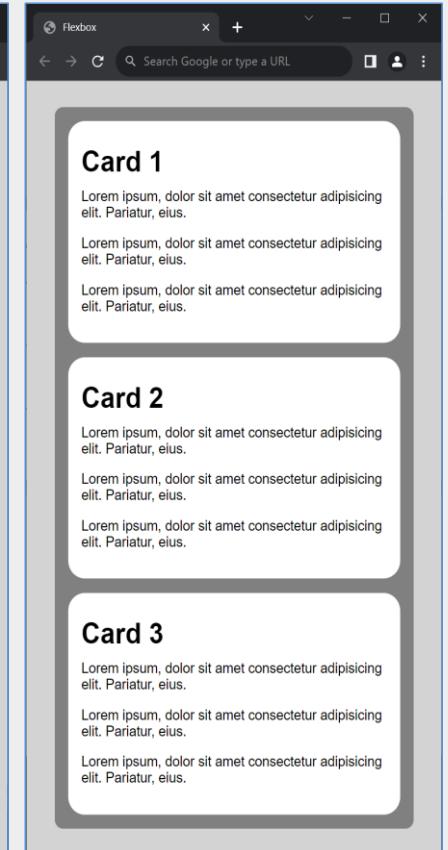
```
.container {  
    background-color: gray;  
    padding: 1rem;  
    display: flex;  
    flex-wrap: wrap;  
    justify-content: space-evenly;  
    gap: 1rem;  
}  
  
.card {  
    background-color: white;  
    border-radius: 20px;  
    padding: 1rem;  
    flex: 1 1 200px;  
}
```



Janela com largura "normal"



Janela com largura reduzida



Janela com largura mínima

# Referências

- [www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)
- [developer.mozilla.org/en-US/docs/Web/CSS](http://developer.mozilla.org/en-US/docs/Web/CSS)
- [www.w3.org/Style/CSS/learning](http://www.w3.org/Style/CSS/learning)
- **HTML and CSS: Design and Build Websites**, Jon Duckett.