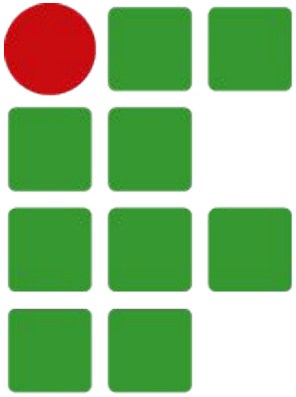


INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Posicionamento e Layout

Desenvolvimento Web

Prof. Diego Stiehl

INSTITUTO FEDERAL

Propriedade `display`

- Especifica o tipo de caixa de renderização de cada elemento
- Define a forma como os elementos fluem e são mostrados dentro da página
- Todo elemento possui um valor padrão para a propriedade `display`
  - Varia para cada elemento

```
.display {
  display: block;
}
```

INSTITUTO FEDERAL

Propriedade `display`

- Valores aceitos:
  - block
    - Elemento domina a linha dentro de seu container
    - Podemos manipular as dimensões do elemento
  - inline
    - Elemento se comporta como um caractere
    - Aceita elementos ao lado
    - Ao preencher a linha, o fluxo segue na próxima
    - Não podemos manipular as dimensões e margens do elemento

Documentação: https://developer.mozilla.org/pt-BR/docs/Web/CSS/display

INSTITUTO FEDERAL

Propriedade `display`

- Valores aceitos:
  - inline-block
    - Elemento se comporta como inline, mas podemos manipular suas dimensões
  - none
    - O elemento não é renderizado na página
    - Página se comporta como se o elemento não existisse

Documentação: https://developer.mozilla.org/pt-BR/docs/Web/CSS/display

INSTITUTO FEDERAL

Propriedade `display`

- Valores aceitos:
  - flex (e inline-flex)
    - Flexible Box Module (Flexbox)
  - grid (e inline-grid)
    - Grid Layout
  - list-item
    - Para itens de listas `<ul>` e `<ol>`
  - Relacionados a tabelas (diversos)

Documentação: https://developer.mozilla.org/pt-BR/docs/Web/CSS/display

INSTITUTO FEDERAL

Propriedade `position`

- Especifica o método de posicionamento de um elemento
- Pode assumir um destes valores:
  - static
  - relative
  - fixed
  - absolute
  - sticky

INSTITUTO FEDERAL

position: **static**;

- Por padrão, todo elemento é estático
- Significa: “mantenha o elemento no seu fluxo padrão”
  - Não importa qual seu `display`
  - Ignora movimentações

```
.static {
  position: static;
}
```

INSTITUTO FEDERAL

position: **relative**;

- Semelhante ao `static`
- Mas podemos movimentar elemento no seu <u>próprio eixo</u>
  - Propriedades: `top`, `bottom`, `right` e `left`

```css
.relative1 {
  display: block;
  position: relative;
  top: 10px;
  left: 20px;
}
```

```css
.relative2 {
  display: inline;
  position: relative;
  top: -5px;
  right: 30px;
}
```

INSTITUTO FEDERAL

position: fixed;

- Prende o elemento ao navegador (viewport)
- Ignora a propriedade `display`
- Podemos fixar o elemento em:
  - `top`, `bottom`, `right` e `left`

```
.fixed1 {
  position: fixed;
  top: 5px;
  left: 20px;
  right: 30px;
}
```

```
.fixed2 {
  position: fixed;
  bottom: 5px;
  right: 30px;
}
```

INSTITUTO FEDERAL

position: **absolute**;

- Prende o elemento ao ancestral não-estático mais próximo
- Ignora a propriedade `display`

```html
<div class="relative">
 <div class="absolute">
   Conteúdo
 </div>
</div>
```

```css
.relative {
  position: relative;
  width: 200px;
  height: 150px;
  background-color: #ff0;
}
```

```css
.absolute {
  position: absolute;
  background-color: #f00;
  bottom: 10px;
  right: 20px;
  left: 30px;
}
```

INSTITUTO FEDERAL

position: sticky;

- Baseado na rolagem da página
  - É um `relative` quando está na tela
  - Vira `fixed` quando sai da tela
- Precisamos definir:
  - `top: 0;`

```
.sticky {
  position: sticky;
  top: 0;
}
```

INSTITUTO FEDERAL

Propriedade z-index

- Utilizada quando elementos se sobrepõem
- Elementos com maiores z-index aparecem mais acima
- Aceita valor negativo
- Valor padrão: zero

```
.elemento2 {
  position: fixed;
  top: 5px;
  z-index: 2;
}
```

```
.elemento1 {
  position: fixed;
  top: 10px;
  z-index: 1;
}
```

INSTITUTO FEDERAL

Flexible Box Module (**Flexbox**)

- Criado para fornecer uma forma mais eficiente de construção de layouts, gerenciando alinhamento e distribuição de espaço entre itens em containers
  - Funciona bem quando a quantidade de itens e seus tamanhos são desconhecidos
- Container pode controlar os seus filhos
  - Altura, largura, alinhamento e ordem

INSTITUTO FEDERAL

Antes do Flexbox

- Técnicas antigas para criação de Layouts
  - Junção de `display`, `position` e `float`
- Acabam sendo pouco flexíveis para a construção de páginas mais complexas
- Novas necessidades:
  - Responsividade (flexibilidade)
    - Mudança de orientação
    - Encolher / espichar
    - Diferentes ordens

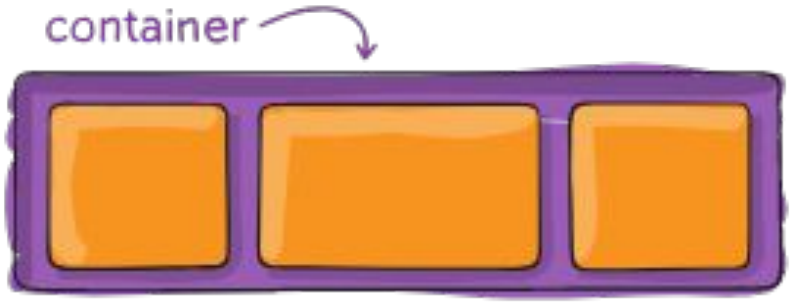
INSTITUTO FEDERAL

Flexbox

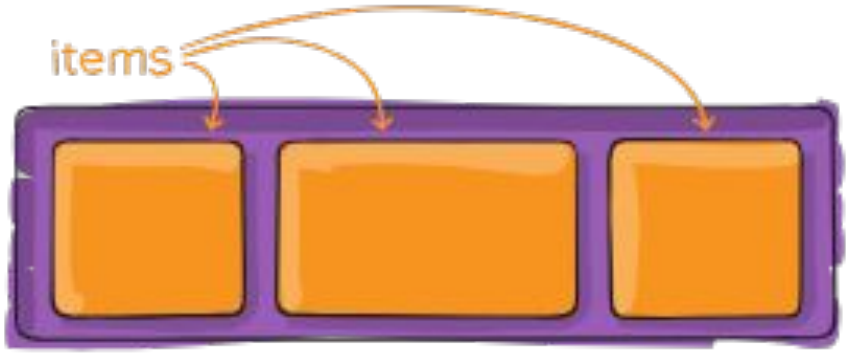
- O **Flexbox** é mais apropriado para componentes de uma página
  - Pequenos recursos
- Bom também para layouts de páginas pouco complexos
  - Foco: unidimensional
- Para a construção de páginas mais robustas, o **Grid Layout** é indicado

INSTITUTO FEDERAL

Container e Itens



Flex Container



Flex Items

Imagens honestamente furtadas do excelente **A Complete Guide to Flexbox | CSS-Tricks**
https://css-tricks.com/snippets/css/a-guide-to-flexbox

INSTITUTO FEDERAL

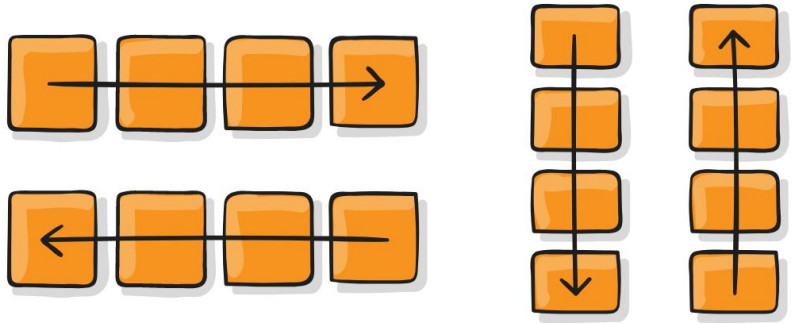
Definindo o Container

- Para definir um container Flex, usamos a propriedade `display`

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

Direção

- O container sempre tem uma direção
  - row (padrão)
  - row-reverse
  - column
  - column-reverse
- A direção gera:
  - Eixo principal
  - Eixo cruzado

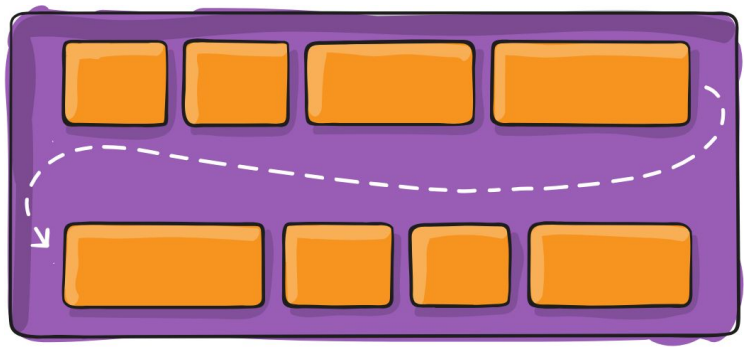


```
.container {
  display: flex;
  flex-direction: row;
}
```

Flex Wrap

- O Flexbox é unidimensional
  - Os elementos são distribuídos no eixo principal
  - Foco: apenas uma linha (ou coluna)
- Mas podemos “fluir” os elementos para mais de uma linha
  - `flex-wrap: nowrap, wrap ou wrap-reverse`

```
.container {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}
```



INSTITUTO FEDERAL

Flex Flow

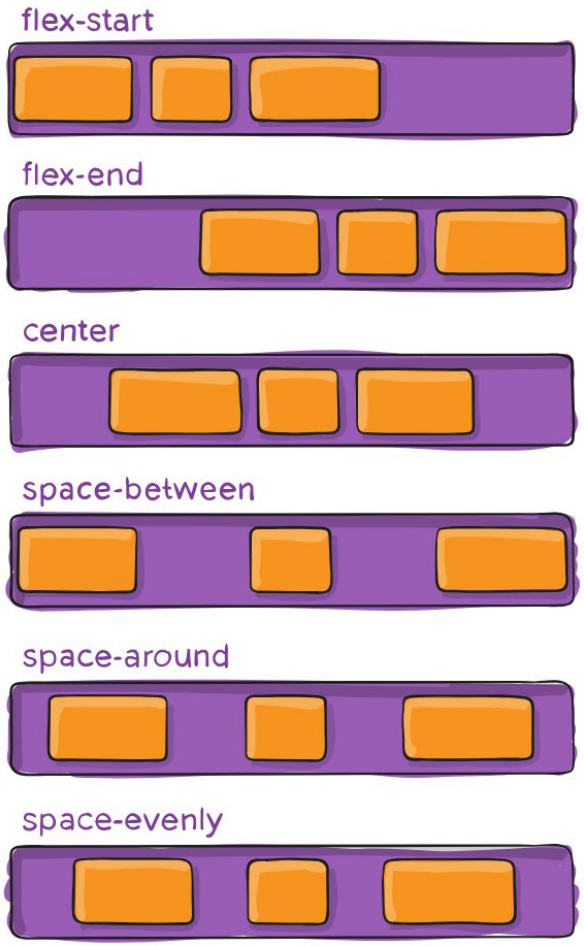
- Podemos abreviar as propriedades `flex-direction` e `flex-wrap`
  - `flex-flow`
- Exemplos:

```
flex-flow: column wrap;
flex-flow: row nowrap;
flex-flow: row wrap-reverse;
flex-flow: row-reverse wrap-reverse;
```

Distribuição

- Podemos distribuir os filhos dentro do eixo principal
- Espaçamentos automáticos
- Propriedade
  - `justify-content`

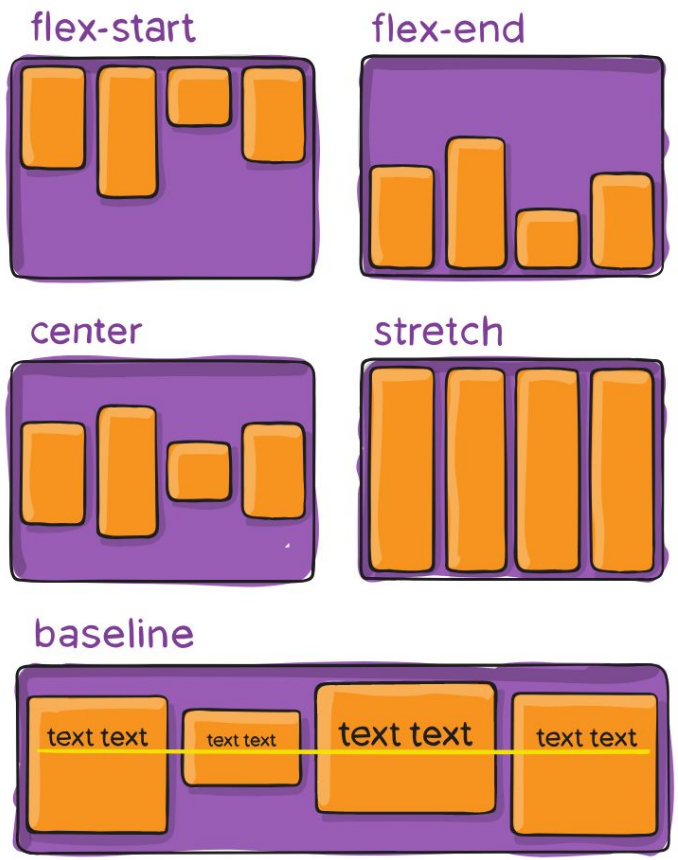
```
.container {
  display: flex;
  justify-content: valores da imagem;
}
```



Alinhamento

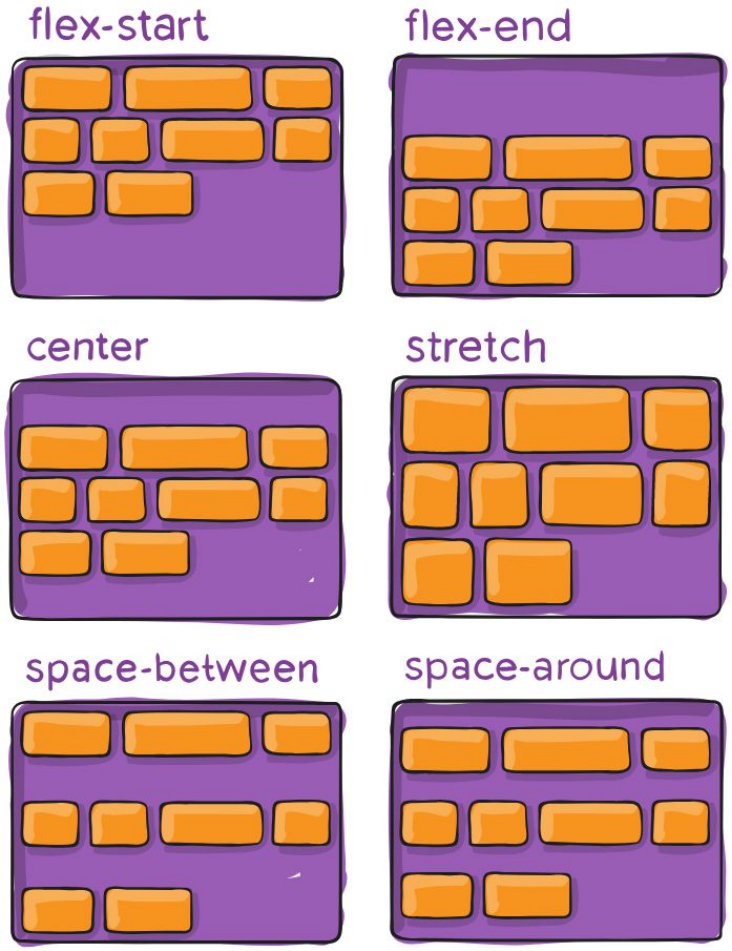
- Podemos alinhar os filhos dentro do eixo cruzado
- Espaçamentos automáticos
- Propriedade
  - `align-items`

```
.container {
  display: flex;
  align-items: valores da imagem;
}
```



Alinhamento (multilinha)

- Quando usamos wrap, podemos juntar ou separar todo o conteúdo
- Propriedade
  - `align-content`



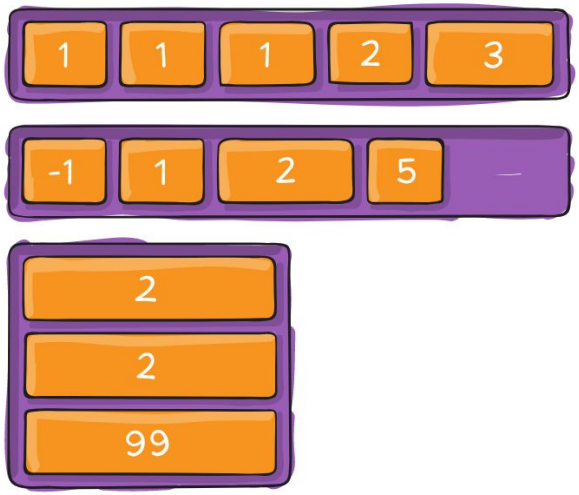

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: valores da imagem;
}
```

Ordem

- Podemos redefinir a ordem de apresentação dos itens
- Propriedade `order` nos itens
  - Menores aparecem antes
  - Valor padrão: zero

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

```
.container .item {
  order: -1;
}
.container .item {
  order: 2;
}
```



INSTITUTO FEDERAL

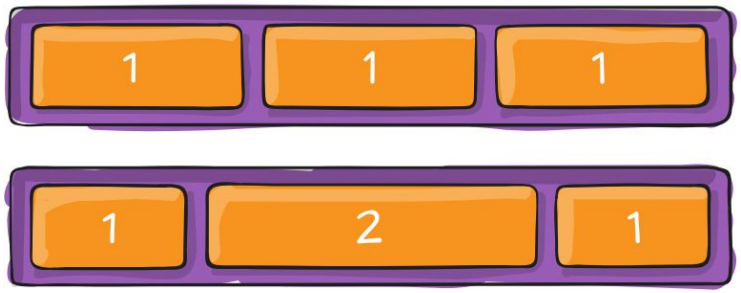
Flexibilidade

- Em vários momentos, vamos querer que o eixo principal seja todo preenchido
- Para isto, para os elementos-filho desejados, precisamos informar:
  - Seu tamanho-base
  - Sua proporção de crescimento
  - Sua proporção de encolhimento

INSTITUTO FEDERAL

Crescimento

- Podemos definir a habilidade de um elemento crescer, se necessário
- Definimos uma proporção de crescimento
- Propriedade:
  - `flex-grow` → valor padrão: `0`
- Elementos com a propriedade definida distribuem entre si o espaço restante

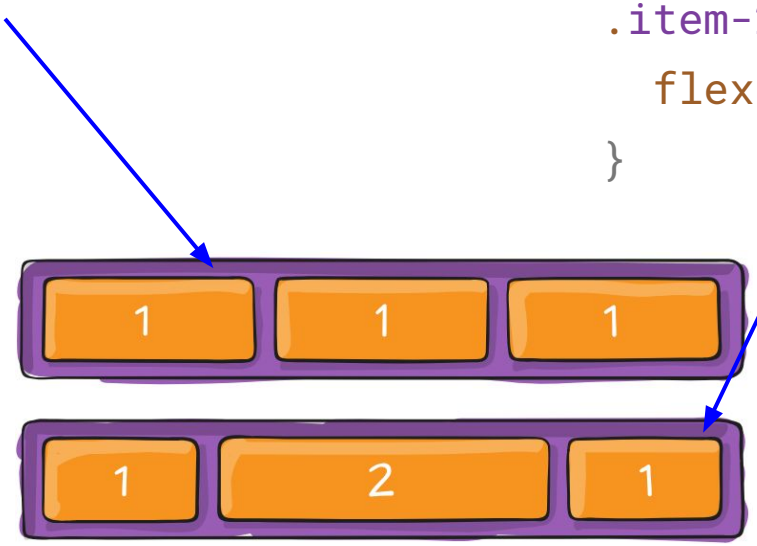


Crescimento

```
.container {
  display: flex;
}

.container * {
  flex-grow: 1;
}
```

```
.container {
  display: flex;
}
.container * {
  flex-grow: 1;
}
.item-2 {
  flex-grow: 2;
}
```



INSTITUTO FEDERAL

Encolhimento

- Da mesma forma, podemos especificar o fator de encolhimento de um item com relação aos demais
- Propriedade:
  - `flex-grow` → valor padrão: 1
- Só faz sentido se soubermos o tamanho-base

INSTITUTO FEDERAL

Tamanho-base

- Podemos definir um tamanho-base para cada elemento
- Ele irá crescer ou encolher baseado no tamanho informado
- Podemos usar unidades de medida normais
  - `px`, `%`, `em`, ...
- Ou podemos usar o valor `auto` (padrão)
  - Puxa da largura e altura do elemento

INSTITUTO FEDERAL

Exemplo

```
.container {
  display: flex;
}

.container * {
  flex-grow: 1;
  flex-shrink: 3;
  flex-basis: 100px;
}

.item-2 {
  flex-grow: 2;
  flex-shrink: 1;
}
```

- Tamanho-base de 100px para todos elementos
- Quando “sobra” espaço, .item-2 expande duas vezes mais rápido que os demais.
- Quando “falta” espaço, os demais itens encolhem 3 vezes mais rápido que o .item-2

INSTITUTO FEDERAL

flex

- As propriedades `flex-grow`, `flex-shrink` e `flex-basis` podem ser abreviadas
- Propriedade:
  - `flex: <flex-grow> <flex-shrink> <flex-basis>`
    - Somente a primeira é obrigatória
- Exemplos:

```
.item-2 {
  flex: 1 100px;
}
```

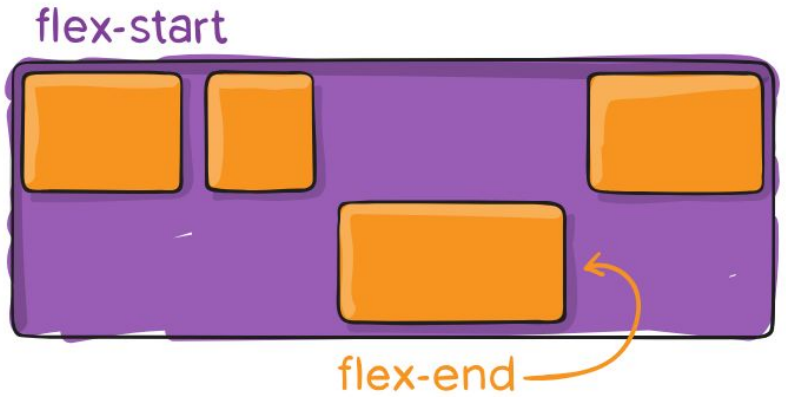
```
.item-1 {
  flex: 1 2 100px;
}
```

```
.item-3 {
  flex: 2;
}
```

Alinhamento de Item

- Um item pode sobrescrever o alinhamento dado pelo container
  - Implementar o seu próprio alinhamento
- Propriedade:
  - `align-self`
- Mesmos valores que `align-items`

```
.item {
  align-self: flex-end;
}
```



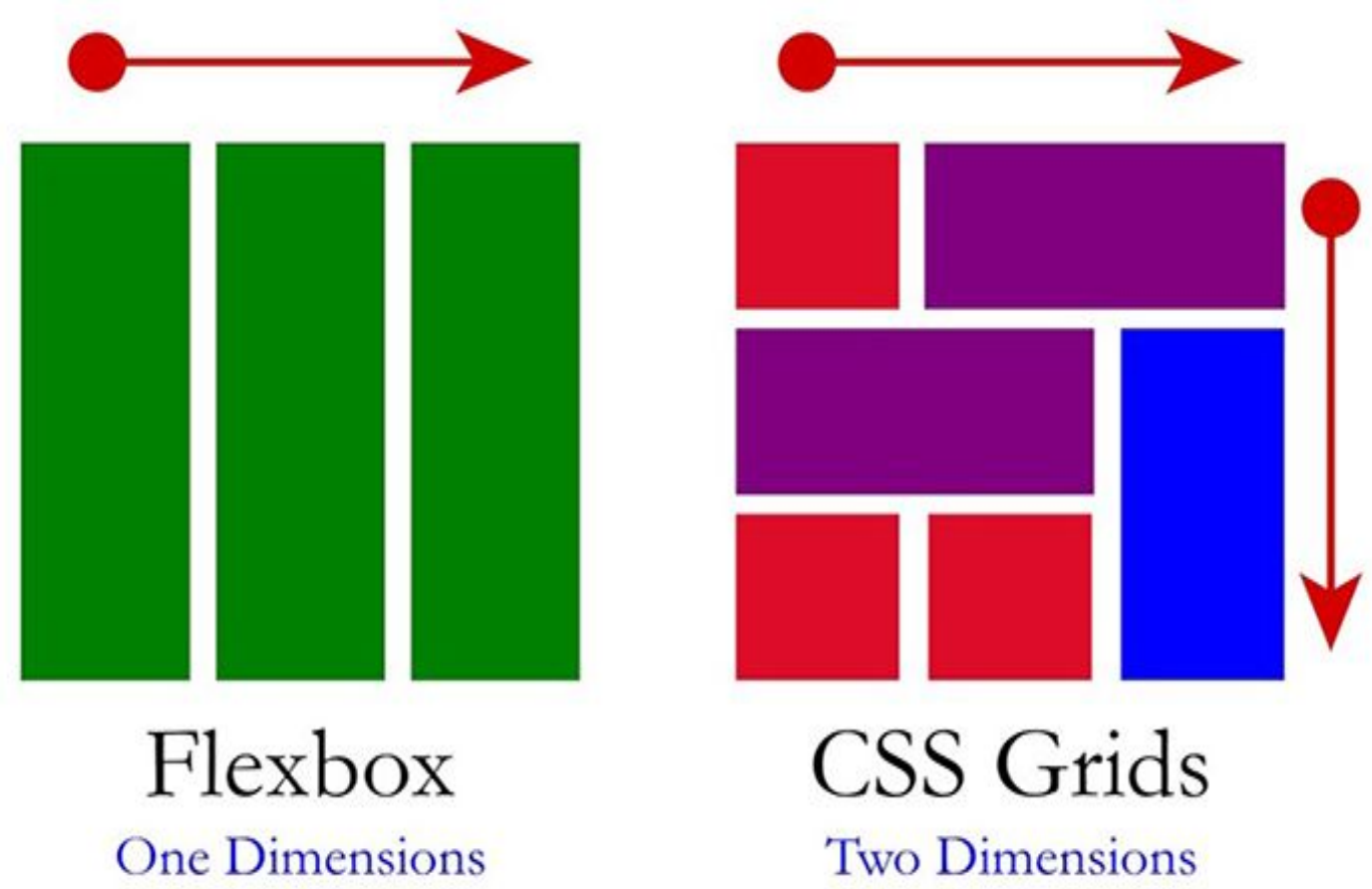
INSTITUTO FEDERAL

Grid Layout

- Grid Layout é um sistema de layout CSS baseado em duas dimensões
  - Linhas / Colunas
- Bastante recente
  - Mas já está bem suportado
- Substitui ou adiciona recursos aos métodos clássicos para criação de templates
- Pode (e deve) ser conjugado com Flexbox

INSTITUTO FEDERAL

Flex X Grid

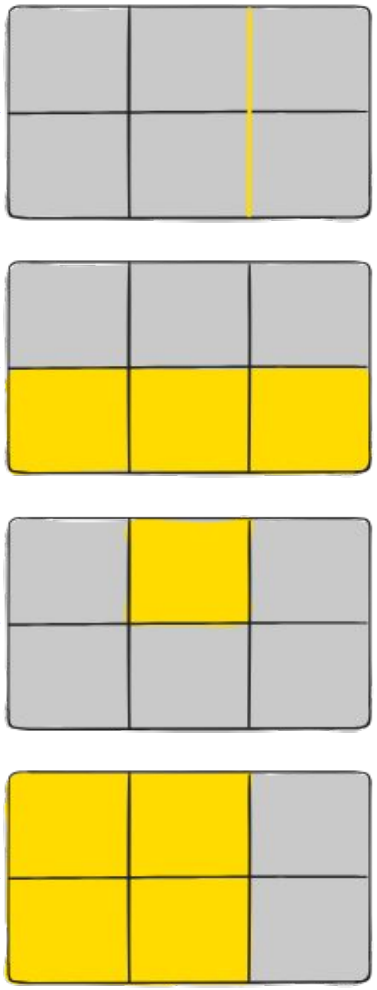


https://www.codepolitan.com/css-grid-vs-flexbox-5b4336849183d

INSTITUTO FEDERAL

Grid Layout - Conceitos

- Grid Container
  - Elemento HTML
- Grid Item
  - Elemento HTML
- Grid Line
- Grid Track
- Grid Cell
- Grid Area



Imagens honestamente furtadas do também excelente
A Complete Guide to Grid | CSS-Tricks
https://css-tricks.com/snippets/css/complete-guide-grid

INSTITUTO FEDERAL

Definindo o Container

- Para definir um container Flex, usamos a propriedade display

```
.container {
  display: grid;
}
```

Colunas

- Para definir as colunas usamos a propriedade:
  - `grid-template-columns`
- Especificamos a quantidade e o tamanho de colunas do nosso layout

```
.container {
  display: grid;
  grid-template-columns: 100px 200px 10% auto;
}
```

4 colunas

Primeira: 100px de largura
Segunda: 200px de largura
Terceira: 10% da largura do container
Quarta: Distribui espaço restante

INSTITUTO FEDERAL

Linhas

- Para definir as linhas usamos a propriedade:
  - `grid-template-rows`
- Especificamos a quantidade e o tamanho de linhas do nosso layout

```
.container {
  display: grid;
  grid-template-columns: 100px 200px 10% auto;
  grid-template-rows: 50px 300px;
}
```

2 linhas

Primeira: 50px de altura

Segunda: 300px de altura

INSTITUTO FEDERAL

Linhas Extra

- Não tem problema se nosso HTML gerar mais linhas dos que o previsto no Grid Layout
  - Elas são renderizadas com o tamanho mínimo
- Para especificar as demais, usar propriedade:
  - `grid-auto-rows`

```
.container {
  display: grid;
  grid-template-columns: 100px 200px 10% auto;
  grid-template-rows: 100px 100px;
  grid-auto-rows: 200px;
}
```

INSTITUTO FEDERAL

Unidade Fracionária

- Quando precisamos uma largura proporcional ao container, usar a unidade:
  - **fr** → Fração
- O uso de percentual (%) é desencorajado
- Distribui a largura proporcionalmente entre os elementos com unidade fr

```
.container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr 3fr;
}
```

INSTITUTO FEDERAL

Exemplo

```
.container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr 3fr;
}
```

4 colunas

Primeira: 100px de largura
Segunda: 1/6 do espaço restante
Terceira: 2/6 do espaço restante
Quarta: 3/6 do espaço restante

INSTITUTO FEDERAL

Repeat

- Se tivermos muitas colunas ou linhas com o mesmo tamanho, podemos compactar
  - Usar: `repeat(quantidade, valor)`

```
.container {
  display: grid;
  grid-template-columns: 50px repeat(5, 1fr) 80px;
  grid-template-rows: 100px repeat(30, 70px);
}
```

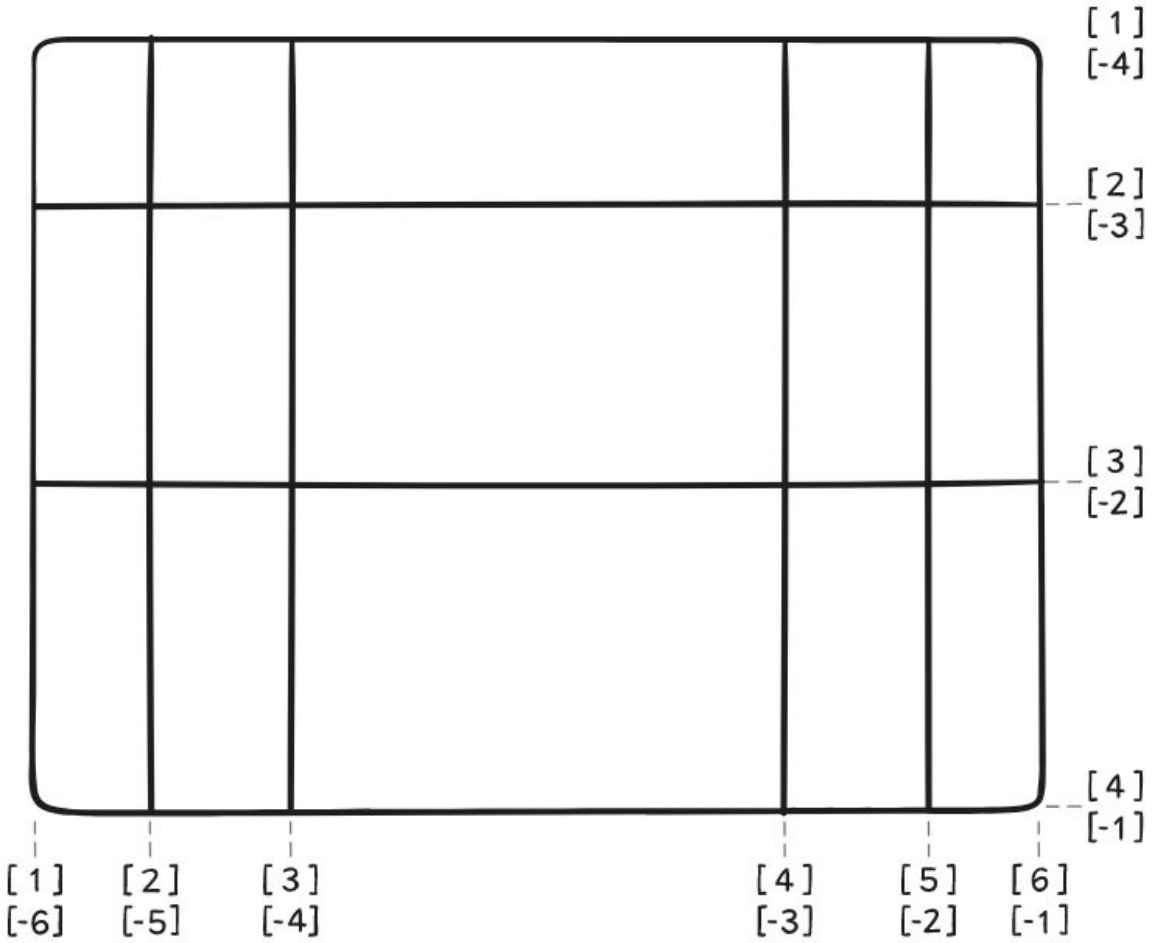
INSTITUTO FEDERAL

Distribuição das Células

- Nem sempre um Item de Grid vai ocupar apenas uma Célula
  - Podemos expandi-los em linhas e colunas
  - Semelhante ao “Mesclar Células” do Excel
- Propriedades:
  - grid-column-start
  - grid-column-end
  - grid-row-start
  - grid-row-end

Grid Lines

- A expansão de células é baseada nas Grid Lines
- Cada célula estará entre duas Grid Lines
  - Em cada uma das duas dimensões



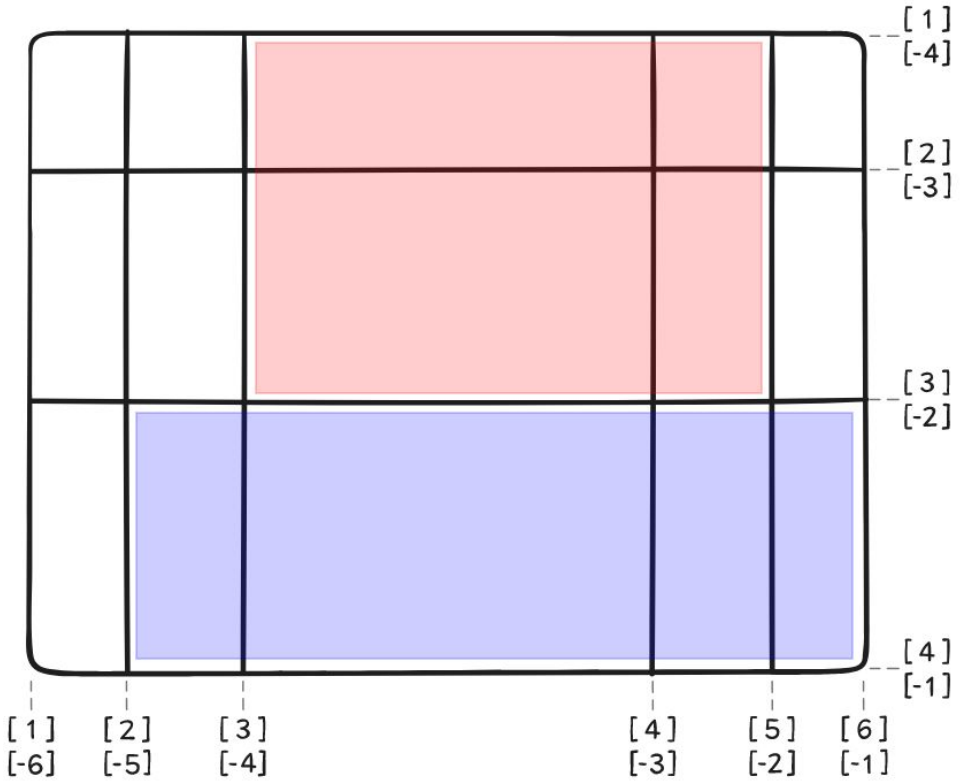
INSTITUTO FEDERAL

Exemplo

```
.container {
  display: grid;
  grid-template-columns: 100px 100px auto 100px 100px;
  grid-template-rows: 100px 200px 300px;
  grid-gap: 5px;
}

.item-3 {
  grid-column-start: 3;
  grid-column-end: 5;
  grid-row-start: 1;
  grid-row-end: 3;
}

.item-9 {
  grid-column-start: 2;
  grid-column-end: -1;
  grid-row-start: 3;
  grid-row-end: -1;
}
```



INSTITUTO FEDERAL

Shorthands

- Abreviando

```
.item-9 {
  grid-column: 2 / -1;
  grid-row: 3 / -1;
}
```

- Abreviando mais

```
.item-9 {
  grid-area: 3 / 2 / -1 / -1;
  /* row-start / column-start / row-end / column-end */
}
```

INSTITUTO FEDERAL

Definindo uma Área

- Podemos definir um Grid Container em áreas
- Usamos strings para definir linhas e colunas
- Propriedade:

```
.site {
  grid-template-areas:
    "area1 area1 area2 area2"
    "area3 area3 area2 area2"
    "area3 area3 . ."
    "area4 area4 area4 area4";
}
```

```
.item {
  grid-area: area1;
}

.item {
  grid-area: area2;
}
```

INSTITUTO FEDERAL

Exemplo

```
<div class="site">
  <div class="cabecalho">Cabeçalho</div>
  <div class="corpo">Corpo</div>
  <div class="menu">Menu</div>
  <div class="rodape">Rodapé</div>
</div>
```

INSTITUTO FEDERAL

Exemplo

```
.site {
  margin-top: 50px;
  display: grid;
  grid-template-columns:
          repeat(4, 1fr);
  grid-template-rows: auto;
  grid-template-areas:
    "cab cab cab cab"
    "corpo corpo . menu"
    "rod rod rod rod";
  grid-gap: 5px;
}
```

```
.site .cabecalho {
  grid-area: cab;
}
.site .corpo {
  grid-area: corpo;
}
.site .menu {
  grid-area: menu;
}
.site .rodape {
  grid-area: rod;
}
```