1 Flexbox

O flexbox é uma tecnologia moderna para estruturar layouts complexos, que anteriormente só poderiam ser construídos por meio de floats e position. Po rém, essas duas tecnologias possuiam algumas limitações em certos aspec tos, que são:

- Centralizar um bloco de conteúdo verticalmente dentro de seu pai;
- Fazer com que os filhos de um container ocupe uma quantidade igual de largura/altura disponível, independente da quantidade de largura/al tura disponível;
- Fazer todas as colunas de um layout com múltiplas colunas adotem a mesma altura, mesmo que contenham uma quantidade diferente de conteúdo.

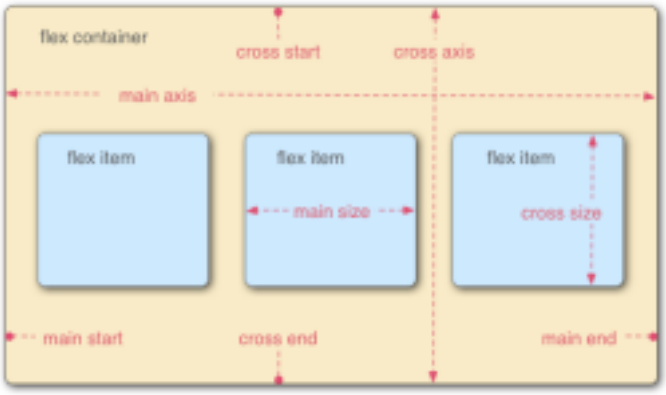
1.1 Partes do Flexbox

O Flexbox pode ser dividido em dois elementos muito importantes: **flex con tainer** e **flex items**. Uma vez que teremos um elemento superior que irá agru par elementos filhos que formaram o layout desejado.

O flexbox também possui duas direções possível: row e column. A direção row indica que os flex items irão se organizar em linha distribuindo-se **ho rizontalmente**, na direção column os flex items irão se organizar em coluna distribuindo-se **verticalmente**.

Cada configuração do flexbox altera a forma como algumas propriedades irão funcionar, por isso é importante entender bem como funciona esses con ceitos do flexbox.

Figura 1: Termos do Flexbox



Fonte: MDN Web Docs.

- O **main axis** é o eixo que corre na direção em que os flex items estão dispostos. O início e o fim do eixo é chamado main start e main end;
- O **cross axis** é o eixo perpendicular que corre na direção em que os flex items são dispostos. O início e o fim deste eixo são chamados de cross start e cross end;
- Quando estamos em uma **row** o main axis é na **horizontal**, e o cross axis é na **vertical**.
- Quando estamos em uma **column** o main axis é na **vertical**, e o cross axis é na **horizontal**.

1.2 Configurando Flexbox

Primeiramente para nosso estudo iremos utilizar um exemplo de caixas sim ples para entender a aplicação de cada propriedade disponível. A ideia é obter o seguinte resultado:

Figura 2: Resultado desejado

Fonte: os autores.

Segue o código estruturado em HTML, sendo a div com class "boxes"o flex container e os flex items sendo as divs internas com class "box-*".

Listagem 1: HTML base

```
<div class="boxes">
<div class="box-1">
<h3>#1</h3>
Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, dolorum possimus illo ab libero iste aliquam temporibus dolore hic dolor.
</div>
<div class="box-2">
<h3>#2</h3>
Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, dolorum possimus illo ab libero iste aliquam temporibus dolore hic dolor. Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, dolorum possimus illo ab libero iste aliquam temporibus dolore hic dolor.
</div>
<div class="box-3">
<h3>#3</h3>
Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, dolorum possimus illo ab libero iste aliquam temporibus dolore hic dolor.
```

```
</div>
<div class="box-4">
<h3>#4</h3>
Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, dolorum
possimus illo ab libero iste aliquam temporibus dolore hic dolor.
</div>
</div>
```

Por padrão cada div ficaria uma acima da outra, seguindo o fluxo normal do HTML, porém podemos definir *display: flex* dentro de *.boxes* para mudar essa forma de exibição interna dos elementos.

Para obter o resultado desejado, podemos definir o seguinte CSS:

Listagem 2: Estilos

```
* {
margin: 0;
font-family: sans-serif;
box-sizing: border-box;
}

.boxes {
background-color: cadetblue;
padding: 16px;
display: flex;
}

.box-1 {
background-color: red;
}

.box-2 {
background-color: gold;
}

.box-3 {
background-color: hotpink;
}

.box-4 {
background-color: indigo;
color: white;
}

.box-1,
.box-2,
.box-3,
.box-4 {
width: 400px;
padding: 8px;
border-radius: 8px;
}
```

Dessa forma configuramos nosso layout para utilizar o Flexbox e agora os elementos por padrão estão organizados em **rows**.

1.3 flex-direction

Com esta propriedade podemos definir como o Flexbox irá distribuir seus ele mentos: em linha ou em coluna.

Vale lembrar que isso muda os eixos main e cross e devemos estar atentos a isto. Por padrão o flex será **row**, porém para mudar isto basta utilizar: *flex direction: column* no flex container (.boxes).

Figura 3: flex-direction: column;

Fonte: os autores.

Porém, iremos trabalhar alguns conceitos de Flexbox (row e column) sepa radamente para um melhor entendimento, então para os conceitos a diante estaremos utilizando *flex-direction: row*.

1.4 flex-wrap

Voltando para *flex-direction: row*, podemos fazer o seguinte teste: reduzir a largura disponível. Sabemos que todos os quatro flex-items possuem uma largura de *400px* definida.

Isso significa que se reduzirmos a largura da tela além disso, o Flexbox vai dar um jeito, certo?

Figura 4: Flexbox na largura limitada

Fonte: os autores.

Podemos ver que os flex itens acabam sendo compactados pelo flex con tainer, o que pode ser um efeito indesejado. Por padrão, os flex itens não quebram linha (*nowrap*), a propriedade *flex-wrap* trata justamente deste com portamento.

Aplique *flex-wrap: wrap* no flex container para que as larguras sejam res peitadas e assim teremos um efeito de quebra de linha muito interessante.

Figura 5: flex-wrap: wrap

Fonte: os autores.

Dessa forma, os itens não redimensionados de maneira indesejada. Tor nando o layout mais adaptável a depender da largura disponível. Você pode definir tanto o flex-direction quanto o flex-wrap usando a propriedade *flex flow*, que é um atalho.

Por exemplo: flex-flow: row wrap.

1.5 justify-content

Essa propriedade ajuda a controlar como os flex itens irão se comportar no eixo principal. Será na **horizontal** se for **row**, e **vertical** se for **column**. Aplica mos esta propriedade em *.boxes* no flex container.

Um ponto importante é: se os itens ocuparem todo o container a propri edade não irá funcionar (o efeito não será visível). Portanto, a largura dos flex-itens vai ser ajustada de 400px para 200px. Valores possíveis:

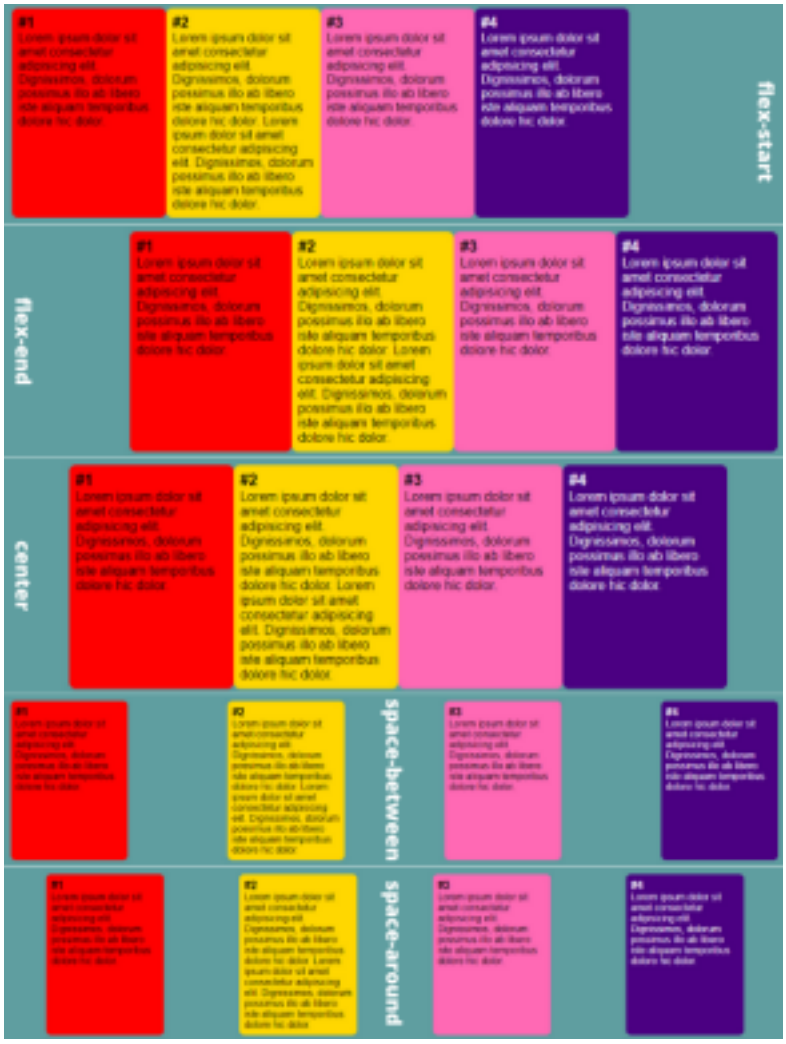
• **flex-start**: Propriedade padrão. Alinha os itens no começo do

container; • **flex-end**: Alinha os itens no final do container;

• **center**: Alinha os itens no centro do container;

- **space-between**: Gera um espaçamento igual entre os elementos. O pri meiro e último item ficam grudados no início e no final;
- **space-around**: Gera um espaçamento entre os elementos. Sendo que os espaços entre os itens são duas vezes maiores que o espaçamento no início e no final.

Figura 6: justify-content



Fonte: os autores.

1.6 align-items

Essa propriedade ajuda a controlar como os flex itens irão se comportar no eixo cross. Será na **vertical** se for **row**, será na **horizontal** se for **column**. Para testar melhor esta propriedade e as próximas, iremos definir *800px* para a altura do .boxes.

- **stretch**: Propriedade padrão. Faz os flex itens crescerem igualmente no eixo cross;
- **flex-start**: Alinha os itens no início;
- **flex-end**: Alinha os itens no final;
- **center**: Alinha os itens ao centro.

Figura 7: align-items

Fonte: os autores.

1.7 align-content

Podemos confundir esta propriedade com align-items, no entanto, align-content busca alinhar as linhas do container (o conteúdo). Caso não tenhamos uma altura definida, ou não exista quebra de linha no container, essa propriedade não funcionará.

- **stretch**: Propriedade padrão. Faz com que os itens cresçam

igualmente; • **flex-start**: Alinha as linhas no início;

- **flex-end**: Alinha as linhas no final;
- **center**: Alinha as linhas ao centro;
- **space-between**: Similar ao que acontece no justify-content. Porém se aplica às linhas.

- **space-around**: Similar ao que acontece no justify-content. Porém se aplica às linhas.

Figura 8: align-content

Fonte: os autores.

Figura 9: align-content

Fonte: os autores.

1.8 align-self

Essa propriedade pode sobrepôr o align-items definido. Ela controla um flex item específico. Por exemplo, no .box-1 podemos definir um align-self: flex start. Enquanto os demais seguem o fluxo de center, o .box-1 segue com seu próprio alinhamento.

Figura 10: align-self

Fonte: os autores.

Para obter este efeito o flex container está com as seguintes propriedades definidas:

Listagem 3: .boxes

```
1 .boxes {
2 background-color: cadetblue;
3 padding: 16px;
4 display: flex;
5 flex-wrap: wrap;
6 justify-content: space-around;
7 height: 800px;
8 align-items: center;
9 }
```


1.9 Diferenças na column

Particularmente não há grandes diferenças quando utilizamos *flex-direction: column*. No entanto, é importante entender que;

- A altura precisa estar definida para uma boa parte das propriedades funcionarem. Diferente da largura dos elementos com flex que sempre é definida, mesmo que implicitamente;

- **justify-content**: Agora alinha os elementos no eixo vertical. Lembrando que é necessário definir uma altura caso queira usar esta propriedade de fato. Pois sem uma altura definida o flex box não consegue calcular o posicionamento dos elementos;
- **flex-wrap**: O mesmo acontece com flex-wrap, caso não haja uma altura definida não ocorrerá a quebra, nesse caso uma quebra em colunas;
- **align-items**: No caso de align-items, os elementos são distribuídos ho rizontalmente, podemos controlar isso;
- **align-content**: Segue a mesma ideia, porém o alinhamento ocorre com as colunas. Lembrando a necessidade de serem geradas colunas para esta propriedade funcionar;
- **align-self**: Irá afetar o alinhamento horizontal do elemento.

Dica: Faça testes diversos com a propriedade *flex-direction: column* ativada, assim você irá aprendendo como utilizá-la corretamente.

1.10 Itens flexíveis

Agora iremos testar uma capacidade muito interessante do Flexbox: os itens flexíveis. Podemos definir proporções para os flex itens de acordo com o es paço disponível. Observe o HTML a seguir:

Listagem 4: HTML base

```
<div class="boxes-2">
<div class="box-flex-1">#1</div>
<div class="box-flex-2">#2</div>
<div class="box-flex-3">#3</div>
</div>
```

Agora temos aqui o CSS que descreve os itens flexíveis e o container:

Listagem 5: CSS base

```
.boxes-2 {
background-color: orange;
height: 200px;
display: flex;
flex-flow: row nowrap;
}
```



```

.box-flex-1 {
background-color: red;
flex: 1;
}

.box-flex-2 {
background-color: green;
```

```
15 flex: 2;
16 }
17
18 .box-flex-3 {
19 background-color: hotpink;
20 width: 100px;
21 }
```

A propriedade *flex* define uma proporção para o elemento de acordo com o espaço disponível. Neste caso, .box-flex-1 ocupa 1/3 do espaço deixado pelo .box-flex-3 que possui largura definida (100px). Já o .box-flex-2 ocupa 2/3 do espaço deixado pelo .box-flex-3.

O cálculo sempre é feito em cima dos flex definidos, some todos e terá as proporções de cada um, neste caso 2 elementos estão definindo flex: 1 + 2 = 3. Dessa forma, temos 1/3 para .box-flex-1 e 2/3 para .box-flex-2.

Caso .box-flex-3, tivesse *flex: 1*, as proporções mudariam, pois agora temos 1 + 2 + 1 = 4. Vale ressaltar que ao usar flex para os elementos, propriedades como wrap não funcionarão corretamente, pois os itens vão se ajustando di namicamente, a não ser que aja elementos com largura definida.

Figura 11: flex

Fonte: os autores.

2 Desafio: Barra de Navegação

Crie uma barra de navegação seguindo os moldes da imagem a seguir:

Figura 12: Navbar

Fonte: os autores.

Observe atentamente os elementos Logo e Nav, estão dentro da área da barra de navegação separadas por um grande espaçamento que empurra o Logo para o início e a Nav para o final. Dentro da Nav, vemos os elementos de Link organizados um ao lado do outro, o que também pode ser um Flex box. Parte importante de construir layouts complexos é buscar enxergar como aplicar o Flexbox e outras técnicas de layout com CSS.

3 Outras técnicas de layout

O Flexbox é uma das mais modernas formas e mais utilizada nos websites modernos, tanto pela sua facilidade de aplicar como de entendimento. Porém existem outras técnicas para montar o seu layout, o seu uso dependerá do contexto, recomendamos a leitura dos seguintes materiais:

- Floats;
- Positioning;
- Grids;