SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Aula 13 | Módulo: HTML + CSS / SASS (continuação)

- ➔ Propriedades de layout: display, position, flexbox, grid (introdução)
- ➔ Media queries e responsividade básica

Abrindo editor de código

Vamos agora abrir o VSCode e criar o index.html

- Com o programa aberto, clique em File > Open Folder... (Arquivo > Abrir Pasta...).
- Escolha um local para criar a sua pasta, crie uma nova pasta e dê o nome de **seunome_aula_13**. Depois dê dois clique nessa pasta criada e clique em **Selecionar pasta**. O VSCode reabrirá dentro dessa pasta que foi criada.
- Agora vamos criar o arquivo HTML e CSS:
- Dê o nome de **index.html** , **script.js** e **style.css**

Lógica de Programação com JavaScript (continuação)

index.html

index.html > ...

```
1   <!DOCTYPE html>
2   <html lang="pt-BR">
3   <head>
4       <meta charset="UTF-8">
5       <link rel="stylesheet" href="style.css">
6       <title>Aula 13</title>
7   </head>
8   <body>
9       <h1>Meu site!</h1>
10      <script src="script.js"></script>
11  </body>
12  </html>
```

HTML + CSS / SASS (continuação)

Propriedades de Layout e Responsividade

- Agora vamos iniciar o aprendizado da organização dos elementos na tela.
- Veremos como usar **display**, **position**, **flexbox** e uma introdução ao **grid**.
- Também veremos **media queries** para adaptar sites a diferentes tamanhos de tela.

Quando você abre um site no celular, o que acontece com o layout? Ele se adapta ou fica difícil de usar?

HTML + CSS / SASS (continuação)

O que é display?

- Todo elemento HTML tem uma forma padrão de aparecer na página.
- Essa forma é controlada pela propriedade display.
- É como se disséssemos: “Como esse elemento vai se comportar no espaço da tela?”.

HTML + CSS / SASS (continuação)

Valores mais comuns do display

- **block** (bloco): Imagine uma caixa grande que empurra tudo para baixo.
  - O elemento ocupa toda a largura disponível da tela, mesmo que o conteúdo seja pequeno.
  - Sempre começa em uma nova linha.
  - , <p>, <h1> já são block por padrão.

Valores mais comuns do display

- **inline** (em linha): Imagine palavras dentro de uma frase, lado a lado.
  - O elemento ocupa somente o espaço do conteúdo.
  - Fica na mesma linha que outros elementos, se couber.
  - , <a>, já são inline por padrão.

Valores mais comuns do display

- **inline-block** (misto): É como ter caixinhas pequenas alinhadas uma ao lado da outra.
  - É parecido com o inline, mas permite definir largura e altura.
  - Fica lado a lado, mas com controle visual maior.

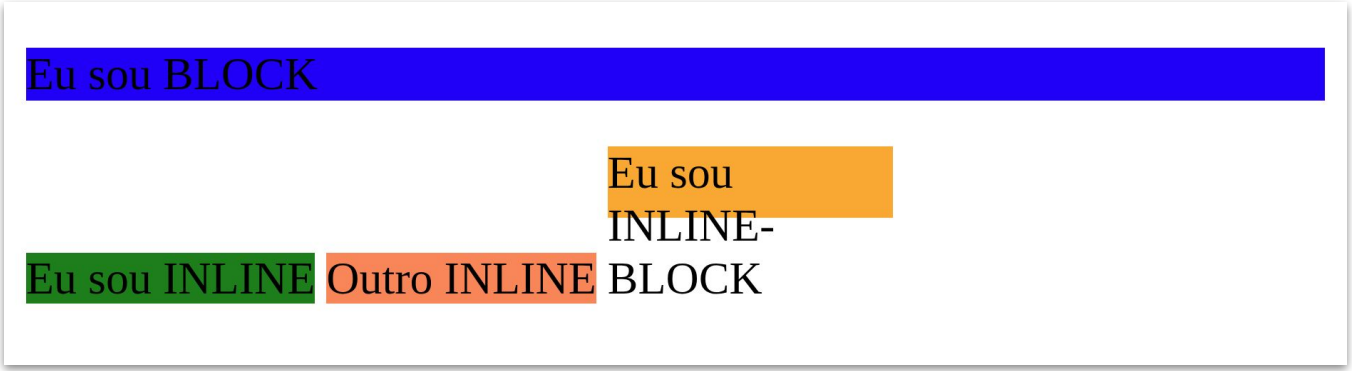
Valores mais comuns do display

- **none** (oculto): É como se o elemento não existisse na página.
  - O elemento não aparece na tela e não ocupa espaço.
  - Diferente de visibility:hidden (que deixa invisível mas ainda ocupa espaço).

HTML + CSS / SASS (continuação)

Valores mais comuns do display

```
<body>
    <p style="display:block; background: blue;">Eu sou BLOCK</p>
    <span style="display:inline; width:700px; height:700px; background: green;">Eu sou INLINE</span>
    <span style="display:inline; width:700px; height:700px; background: coral;">Outro INLINE</span>
    <div style="display:inline-block; width:100px; height:25px; background: orange;">Eu sou INLINE-BLOCK</div>
    <p style="display:none;">Ninguém vai me ver (NONE)</p>
</body>
```



HTML + CSS / SASS (continuação)

Exercício 1

- Crie, no seu arquivo index.html, 3 divs:

```
<body>
    <div class="caixa1">Sou BLOCK</div>
    <div class="caixa2">Sou INLINE</div>
    <div class="caixa3">Sou INLINE-BLOCK</div>
</body>
```

- No seu arquivo style.css, adicione os estilos (block, inline e inline-block) para diferenciar cada caixa, respectivamente.

HTML + CSS / SASS (continuação)

Exercício 2

- Crie um menu horizontal usando **inline-block** em várias caixas, simulando botões lado a lado.

```
<body>
    <div class="menu">
        <div class="botao">Início</div>
        <div class="botao">Sobre</div>
        <div class="botao">Serviços</div>
        <div class="botao">Contato</div>
    </div>
</body>
```

Para facilitar, deixo a estrutura HTML, façam agora o estilo no style.css de vocês

HTML + CSS / SASS (continuação)

O que é position?

- É a propriedade do CSS que controla onde um elemento aparece dentro da página.
- Define se o elemento vai seguir o fluxo normal do HTML ou se vai ser colocado em uma posição específica.
- Pense nisso como dar coordenadas (top, right, bottom, left) para um objeto.

Valores mais comuns do position

- **static** (padrão):
  - Todo elemento já começa como static.
  - Ele aparece no fluxo normal, sem deslocamento manual.
  - Não reage a top, left, etc.
- Exemplo: Um parágrafo normal no meio da página.

Valores mais comuns do position

- **relative** (relativo a si mesmo):
  - O elemento ainda ocupa seu espaço normal.
  - Mas pode ser movido em relação à posição original (top, right, bottom, left).
  - O espaço que ele ocupava continua reservado.
- Exemplo: empurrar uma caixa 20px para a direita, mas o espaço “vazio” dela continua lá.

Valores mais comuns do position

- **absolute** (posição absoluta):
  - O elemento é retirado do fluxo normal da página.
  - Fica posicionado em relação ao primeiro ancestral com position diferente de static.
  - Se nenhum ancestral for posicionado, usa o body como referência.
- Exemplo: Uma etiqueta grudada no canto de uma imagem.

Valores mais comuns do position

- **fixed** (fixo na tela):
  - O elemento fica sempre fixo em relação à janela do navegador.
  - Não importa se você rola a página, ele continua no mesmo lugar.
- Exemplo: aquele botão de “voltar ao topo” ou o cabeçalho fixo no topo.

Valores mais comuns do position

- **sticky** (colante):
  - Mistura relative e fixed.
  - O elemento fica no fluxo normal, mas quando chega em uma posição definida, ele “cola” na tela.
  - Muito usado em menus que acompanham a rolagem.
- Exemplo: título de seção que gruda no topo quando você rola para baixo.

HTML + CSS / SASS (continuação)

Exemplo de uso de position (html):

```
<body>
    <div class="caixa static">Static</div>
    <div class="caixa relative">Relative</div>
    <div class="caixa absolute">Absolute</div>
    <div class="caixa fixed">Fixed</div>
    <div class="caixa sticky">Sticky</div>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br
    ><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><b
    r><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
    br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br
    ><br><br><br><br>
</body>
```

HTML + CSS / SASS (continuação)

Exemplo de uso de position (css):

style.css > ...

```css
.caixa {
  width: 100px;
  height: 50px;
  margin: 10px;
  color: white;
  text-align: center;
  line-height: 50px;
}

.static { background: gray; position: static; }
.relative { background: blue; position: relative; top: 20px; left: 20px; }
.absolute { background: red; position: absolute; top: 0; right: 0; }
.fixed { background: green; position: fixed; bottom: 0; right: 0; }
.sticky { background: orange; position: sticky; top: 0; }
```

Exercício 3

- Faça um cabeçalho fixo (fixed) no topo da página com um fundo colorido e veja como ele acompanha a rolagem.

```
<body>
    <header class="cabecalho">
        <h1>Meu Site</h1>
        <nav>
            <a href="#">Início</a>
            <a href="#">Sobre</a>
            <a href="#">Serviços</a>
            <a href="#">Contato</a>
        </nav>
    </header>

    <main>
        <p>Aqui é um parágrafo qualquer</p>
        <p>Role a página para ver o efeito do cabeçalho fixo.</p>
        <p>Multiplique esses parágrafos abaixo várias vezes para criar muitas linhas</p>
        <p>Bla bla bla blaaa...</p>
    </main>
</body>
```

O que é Flexbox?

- O Flexbox é um modelo de layout do CSS criado para facilitar a organização de elementos.
- Antes dele, usávamos tabelas ou floats (muito mais trabalhoso).
- Com Flexbox conseguimos:
  - Alinhar elementos horizontal e verticalmente com poucas linhas de código.
  - Distribuir espaços de forma automática.
  - Fazer layouts que se adaptam a diferentes tamanhos de tela.

HTML + CSS / SASS (continuação)

Conceito principal de Flexbox

- O Flexbox trabalha com eixos:
  - Eixo principal (main axis):
    - Pode ser horizontal (linha) ou vertical (coluna).
    - Controlado por flex-direction.
  - Eixo cruzado (cross axis):
    - É sempre perpendicular ao eixo principal.
    - Usado para alinhamentos verticais ou horizontais complementares.

➔ Imagine uma caixa maior (container) que segura várias caixinhas menores (itens).
➔ O Flexbox controla como essas caixinhas vão se organizar dentro da caixa maior.

HTML + CSS / SASS (continuação)

Exemplo básico

```html
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
    </div>
</body>
```

style.css > ...

```css
.container {
    display: flex; /* ativa o flexbox no container */
    background: gray;
}

.item {
    background: blue;
    color: white;
    margin: 10px;
    padding: 20px;
}
```

Só com **display: flex** os itens já ficam lado a lado, ocupando o espaço horizontal.

HTML + CSS / SASS (continuação)

Conceito principal de Flexbox

- Por padrão, quando usar **display: flex**, o **flex-direction** já é configurado como **row**. Experimente adicionar **flex-direction: column** para visualizar o que acontece.

style.css > ...

```css
.container {
    display: flex; /* ativa o flexbox no container */
    flex-direction: column;
    background: gray;
}

.item {
    background: blue;
    color: white;
    margin: 10px;
    padding: 20px;
}
```

Propriedades do Container Flex

- Quando colocamos display: flex no elemento pai (container):
  - Todos os filhos diretos (itens) passam a ser controlados por esse sistema.
  - A partir daí, podemos dizer em qual direção os itens vão, como eles se alinham e como ocupam o espaço disponível.

Principais propriedades do Container Flex

- **flex-direction**: Define a direção dos itens
  - **row** (padrão): em linha, da esquerda para direita.
  - **row-reverse**: em linha, mas da direita para esquerda.
  - **column**: em coluna, de cima para baixo.
  - **column-reverse**: em coluna, mas de baixo para cima.

Principais propriedades do Container Flex

- **justify-content**: Alinha os itens no eixo principal
  - **flex-start** (padrão): alinhados ao início.
  - **flex-end**: alinhados ao final.
  - **center**: alinhados ao centro.
  - **space-between**: espaço igual entre os itens.
  - **space-around**: espaço ao redor dos itens.
  - **space-evenly**: espaço igual em volta e entre os itens.

Principais propriedades do Container Flex

- **align-items**: Alinha os itens no eixo cruzado
  - **stretch** (padrão): esticam para ocupar altura/largura.
  - **flex-start**: grudados no início do eixo cruzado.
  - **flex-end**: grudados no final.
  - **center**: alinhados no centro.

Principais propriedades do Container Flex

- **flex-wrap**: Permite quebra de linha dos itens
  - **nowrap** (padrão): tudo na mesma linha.
  - **wrap**: itens quebram linha quando não couberem.
  - **wrap-reverse**: quebram linha, mas de baixo para cima.

HTML + CSS / SASS (continuação)

Principais propriedades do Container Flex

```html
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
    </div>
</body>
```

style.css > ...

```css
.container {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items: center;
    flex-wrap: wrap;
    height: 200px;
    background: gray;
}

.item {
    background: blue;
    color: white;
    margin: 10px;
    padding: 20px;
}
```

HTML + CSS / SASS (continuação)

Principais propriedades do Container Flex

```html
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
    </div>
</body>
```

style.css > ...

```css
.container {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items: center;
    flex-wrap: wrap;
    height: 200px;
    background: gray;
}

.item {
    background: blue;
    color: white;
    margin: 10px;
    padding: 20px;
}
```

1. Testem cada valor de **justify-content** e observarem a diferença.
2. Depois mudem para **flex-direction: column** e veja como muda o eixo principal.

Propriedades dos Itens (Filhos) no Flexbox

- Os itens dentro de um container flex não precisam se comportar todos iguais.
- Com algumas propriedades, conseguimos dizer qual item cresce mais, qual encolhe mais e até sobrescrever o alinhamento só dele.

Principais propriedades dos Itens (Filhos) no Flexbox

- **flex-grow**: Quanto o item pode crescer
  - Valor padrão: 0 (não cresce além do conteúdo).
  - Se um item tiver **flex-grow: 2** e outro **flex-grow: 1**, o primeiro ocupa o dobro de espaço.

HTML + CSS / SASS (continuação)

Principais propriedades dos Itens (Filhos) no Flexbox

- **flex-shrink**: Quanto o item pode encolher
  - Valor padrão: 1 (todos podem encolher).
  - Se um item tiver **flex-shrink: 0**, ele não diminui quando a tela é pequena.

HTML + CSS / SASS (continuação)

Principais propriedades dos Itens (Filhos) no Flexbox

- **flex-basis**: Define o tamanho inicial do item
  - Pode ser em **px**, **%**, **em** ou **auto**.
  - Funciona como uma “largura preferida”.

Principais propriedades dos Itens (Filhos) no Flexbox

- **align-self**: Alinhamento individual no eixo cruzado
  - Pode sobrescrever o align-items do container.
  - Valores: **flex-start**, **flex-end**, **center**, **stretch**.

HTML + CSS / SASS (continuação)

Principais propriedades dos Itens (Filhos) no Flexbox

```html
<body>
    <div class="container">
        <div class="item a">Item A</div>
        <div class="item b">Item B</div>
        <div class="item c">Item C</div>
    </div>
</body>
```

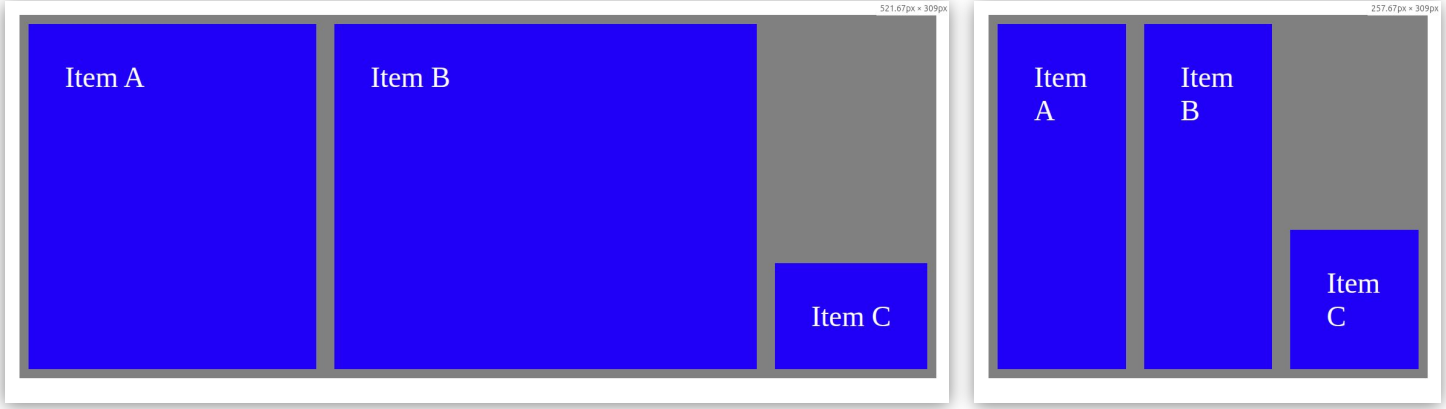
style.css > ...

```css
.container {
    display: flex;
    background: gray;
    height: 200px;
}

.item {
    background: blue;
    color: white;
    margin: 5px;
    padding: 20px;
}

.a { flex-grow: 1; } /* cresce pouco */
.b { flex-grow: 2; } /* cresce o dobro do A */
.c { align-self: flex-end; } /* desce sozinho no eixo cruzado */
```

HTML + CSS / SASS (continuação)

Principais propriedades dos Itens (Filhos) no Flexbox

➔ Redimensione a tela para visualizar como ele agora ajusta.



Experimente colocar **flex-shrink: 0** em um item e diminua a tela. Mude **flex-basis** para **200px** e veja como isso afeta o tamanho inicial. Aplique **align-self: center** em apenas um item para notar a diferença.

O que é o Grid Layout?

- O Grid é um sistema de layout bidimensional do CSS.
- Diferente do Flexbox (que organiza em uma dimensão de cada vez (linha ou coluna)), o Grid permite organizar elementos em linhas e colunas ao mesmo tempo.
- Muito útil para criar layouts de páginas inteiras ou grades complexas de conteúdo.

Conceitos básicos de Grid Layout

- Container: o elemento pai que recebe display: grid.
- Itens do grid: os filhos diretos do container.
- Linhas e colunas: criadas com grid-template-rows e grid-template-columns.
- Gap: espaço entre as linhas e colunas.

HTML + CSS / SASS (continuação)

Conceitos básicos de Grid Layout

```html
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
        <div class="item">4</div>
        <div class="item">5</div>
        <div class="item">6</div>
    </div>
</body>
```

style.css > ...

```css
.container {
    display: grid; /* ativa o grid */
    grid-template-columns: 50px 100px 150px; /* 3 colunas fixas */
    grid-template-rows: auto auto; /* 2 linhas automáticas */
    gap: 10px; /* espaçamento */
    background: gray;
    padding: 10px;
}

.item {
    background: blue;
    color: white;
    text-align: center;
    padding: 20px;
}
```

- ➔ Os itens vão se organizar automaticamente em 3 colunas e 2 linhas.
- ➔ O **gap** cria espaço entre eles.
- ➔ Se aumentar a quantidade de itens, o Grid cria novas linhas automaticamente.

HTML + CSS / SASS (continuação)

Analogia simples para Grid Layout

- Imagine uma planilha do Excel:
  - Cada célula é um “item”.
  - Cada linha e coluna pode ter tamanhos diferentes.
  - Você pode controlar exatamente onde cada item vai ficar.

Configurando linhas e colunas no Grid

- Quando usamos display: grid, precisamos definir a estrutura do grid:
  - Quantas colunas terá.
  - Quantas linhas terá.
  - Como será o espaçamento entre os itens.

HTML + CSS / SASS (continuação)

Principais propriedades do Grid

- grid-template-columns
  - Define o número e largura das colunas.
  - Pode ser em **px**, **%**, **fr** (fração do espaço disponível).
- Exemplo: **grid-template-columns: 200px 1fr 1fr;**
- A 1ª coluna fixa em 200px, as outras duas dividem igualmente o restante do espaço.

Principais propriedades do Grid

- **grid-template-rows**
  - Define a altura das linhas.
  - Exemplo: **grid-template-rows: 100px auto;**
- A primeira linha fixa em 100px, segunda linha cresce conforme o conteúdo.

Principais propriedades do Grid

- **gap**
  - Espaço entre linhas e colunas.
  - Substitui **grid-row-gap** e **grid-column-gap**.
- Exemplo: **gap: 15px;**

HTML + CSS / SASS (continuação)

Outro exemplo prático de Grid

```html
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
        <div class="item">4</div>
        <div class="item">5</div>
        <div class="item">6</div>
    </div>
</body>
```

style.css > ...

```css
.container {
    display: grid;
    grid-template-columns: 200px 1fr 2fr;
    grid-template-rows: 100px 100px;
    gap: 10px;
    background: gray;
    padding: 10px;
}

.item {
    background: blue;
    color: white;
    text-align: center;
    padding: 20px;
}
```

- A primeira coluna sempre tem 200px fixos.
- A segunda ocupa 1 fração do espaço restante.
- A terceira ocupa 2 frações, ou seja, o dobro da segunda.
- O gap deixa os itens separados, sem precisar de margin.

HTML + CSS / SASS (continuação)

O que são Media Queries?

- Media Queries são instruções no CSS que dizem:
  - “Se a tela tiver um tamanho específico (ou condição), use este estilo.”
- São fundamentais para criar sites responsivos, que funcionam bem no celular, tablet e computador.
- Permitem alterar:
  - cores
  - tamanhos de fontes
  - organização de colunas
  - exibir/esconder elementos

HTML + CSS / SASS (continuação)

O que são Media Queries?

```html
<body>
    <p>TEXTO!</p>
</body>
```

style.css > ...

```css
/* Estilo padrão (desktop) */
body {
    background: blue;
    font-size: 18px;
}

/* Se a largura da tela for até 300px */
@media (max-width: 300px) {
    body {
        background: green;
        font-size: 14px;
    }
}
```

Reduza e aumente o tamanho da tela e veja o que acontece.

O que são Media Queries?

- **Outras condições comuns:**
  - (min-width: 768px): a partir de tablets.
  - (orientation: portrait): se o dispositivo estiver em modo retrato.
  - (orientation: landscape): se estiver em modo paisagem.

O que é Responsividade?

- Responsividade é a capacidade de um site se adaptar automaticamente a diferentes tamanhos de tela: celular, tablet, notebook, desktop.
- O layout muda para que o conteúdo continue legível e fácil de usar, sem precisar dar zoom ou rolar para os lados.
- É um dos pilares do desenvolvimento web moderno.

HTML + CSS / SASS (continuação)

Por que responsividade é importante?

- Mais de 70% dos acessos à web vêm de dispositivos móveis.
- Usuários abandonam sites que não funcionam bem no celular.
- Motores de busca (Google, Bing) dão prioridade para sites responsivos no ranking.

HTML + CSS / SASS (continuação)

Como conseguimos responsividade?

- Usando unidades relativas em vez de fixas:
  - em, rem, %, fr, vh, vw.
  - Evitar sempre **px** fixos em tudo.
- Combinando Flexbox e Grid: layouts fluidos.
- Usando Media Queries: para mudar o layout em pontos de quebra.

HTML + CSS / SASS (continuação)

Exercício 1 - Gabarito

style.css > ...

```css
.caixa1 {
    display: block;
    background: blue;
    padding: 10px;
    margin: 5px;
}

.caixa2 {
    display: inline;
    background: green;
    padding: 10px;
    margin: 5px;
}

.caixa3 {
    display: inline-block;
    background: coral;
    width: 120px;
    height: 50px;
    margin: 5px;
}
```

HTML + CSS / SASS (continuação)

Exercício 2 - Gabarito

style.css > ...

```css
.menu {
    background: #ec5151;
    padding: 10px;
}

.menu .botao {
    display: inline-block;
    background: blue;
    color: white;
    padding: 10px 20px;
    margin-right: 5px;
    border-radius: 5px;
}
```

HTML + CSS / SASS (continuação)

Exercício 3 - Gabarito

style.css > ...

```css
body { margin: 0; } /* remove margens padrão do navegador */
.cabecalho {
    position: fixed;
    width: 100%; /* ocupa toda a largura da tela */
    background: blue;
    color: white;
    padding: 15px;
    z-index: 1000; /* fica por cima de outros elementos */
}
.cabecalho h1 { display: inline-block; }
.cabecalho nav { display: inline-block; margin-left: 30px; }
.cabecalho nav a { color: white; margin: 0 10px; }
main { padding: 120px 20px 20px; } /* espaço extra para ñ esconder conteúdo atrás do header */

```

ATÉ A PRÓXIMA AULA!

Front-end - Design. Integração. Experiência.

Professor: Hygor Rasec
https://www.linkedin.com/in/hygorrasec
https://github.com/hygorrasec