SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Aula 14 | Módulo: HTML + CSS / SASS (continuação)

- ➔ Introdução ao pré-processador SASS: variáveis, aninhamento e mixins
- ➔ Organização de estilos com SASS (parciais e importações)

Abrindo editor de código

Vamos agora abrir o VSCode e criar os arquivos

- 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_14**. Depois dê dois clique nessa pasta criada e clique em **Selecionar pasta**. O VSCode reabrirá dentro dessa pasta que foi criada.
- Agora vamos criar os arquivos HTML, JS e SCSS:
- Dê o nome de **index.html** , **script.js** e **style.scss**

HTML + CSS / SASS (continuação)

index.html > ...

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Aula 14</title>
</head>
<body>
    <h1>Estamos na aula 14!</h1>
    <script src="script.js"></script>
</body>
</html>
```

HTML + CSS / SASS (continuação)

O que é um pré-processador de CSS?

- CSS é a linguagem padrão de estilos, mas possui limitações.
- Um pré-processador adiciona recursos extras que o CSS puro não tem.
- O SASS é o pré-processador mais usado atualmente.
- Ele gera CSS comum no final (o navegador continua entendendo apenas CSS).

HTML + CSS / SASS (continuação)

O que é um pré-processador de CSS?

- CSS puro: funciona bem para páginas pequenas, mas em projetos grandes fica difícil de manter.
    - Exemplo: repetir sempre o mesmo código de cor ou fonte.
- Pré-processador: é uma camada a mais entre o que você escreve e o que o navegador entende.
- Você escreve em .scss > compila > vira um arquivo .css
- É como usar uma “versão turbinada” do CSS, com variáveis, funções, herança e organização de arquivos.
- O código final continua sendo CSS normal, por isso não há problema de compatibilidade.

HTML + CSS / SASS (continuação)

Exemplo Visual

O navegador só recebe o CSS final, mas para o desenvolvedor o SASS é mais limpo e organizado.

style.css > ...

```css
button.primary {
    background-color: #3498db;
    color: white;
}

button.secondary {
    background-color: #3498db;
    border: 2px solid #3498db;
    color: black;
}
```

style.scss > ...

```scss
$cor-principal: #3498db;

button {
    &.primary {
        background-color: $cor-principal;
        color: white;
    }

    &.secondary {
        background-color: $cor-principal;
        border: 2px solid $cor-principal;
        color: black;
    }
}
```

HTML + CSS / SASS (continuação)

Como instalar e configurar o SASS no seu projeto

- É necessário compilar SASS para CSS.
- Instalação via Node.js (npm).
  - Verifique se o node está instalado (abra o terminal e digite): **node -v**
  - Verifique se o npm está instalado (abra o terminal e digite): **npm -v**
  - Se não tiver instalado, baixe aqui: **https://www.nodejs.tech/pt-br/download**
- Dois modos de instalação:
  - Global: disponível em qualquer projeto: **npm install -g sass**
  - Local: instalado só dentro do projeto: **npm install sass --save-dev**
- Verificando se está instalado e a versão: **npx sass --version** ou **sass --version**
- Se ainda não funcionar, pode ser que o npm global não esteja no seu PATH. Nesse caso, basta adicionar o caminho das instalações globais do npm no Windows. O caminho que retorna deve ser adicionado no PATH do Windows: **npm config get prefix**

HTML + CSS / SASS (continuação)

Como instalar e configurar o SASS no seu projeto

- Compilando:
  - Esse comando pega o arquivo .scss e gera o .css
  - Comando: **npx sass style.scss style.css**
  - Ou seja, o SASS é como um tradutor automático.
- Modo observador (watch):
  - O compilador fica de olho nas mudanças e recompila automaticamente.
  - Muito útil durante o desenvolvimento.
  - Comando para abrir o observador: **npx sass --watch style.scss style.css**

HTML + CSS / SASS (continuação)

Trabalhando com Variáveis no SASS

- Guardam valores reutilizáveis (cores, tamanhos, fontes etc.).
- Facilitam manutenção e padronização do estilo.
- Declaração com o prefixo $.
- Se mudar a variável > muda em todo o projeto.

Trabalhando com Variáveis no SASS

- Problema no CSS puro:
  - Se você usa a mesma cor em 50 lugares e precisar trocar, terá que editar 50 vezes.
  - Isso aumenta chance de erro e inconsciência visual.
- Como o SASS resolve:
  - Você cria uma variável com a cor, fonte ou tamanho.
  - Usa essa variável em todos os lugares.
  - Se precisar trocar, basta alterar uma única vez.
- Boas práticas:
  - Criar variáveis para paleta de cores.
  - Criar variáveis para fontes e espaçamentos padrão.
  - Guardar essas variáveis em um arquivo separado (_variables.scss).

HTML + CSS / SASS (continuação)

Trabalhando com Variáveis no SASS

```html
<body>
    <h1>Título</h1>
    <button>Botão</button>
</body>
```

Depois que fizer, rode o código para compilar:

npx sass --watch style.scss style.css

style.scss > ...

```scss
$cor-principal: #3498db;
$fonte-principal: 'Arial', sans-serif;

h1 {
    color: $cor-principal;
    font-family: $fonte-principal;
}

button {
    background: $cor-principal;
    border: 2px solid $cor-principal;
}
```

Aninhamento de Seletores no SASS

- Em CSS puro, precisamos repetir seletores várias vezes.
- O SASS permite aninhar seletores dentro de outros.
- Isso deixa o código mais parecido com a estrutura do HTML.
- Torna o CSS mais limpo e organizado.

Aninhamento de Seletores no SASS

- Problema no CSS puro:
  - Para estilizar um menu de navegação (nav ul li a), precisamos escrever o seletor completo toda vez.
- Isso gera repetição e dificulta leitura.
- Como o SASS resolve:
  - Permite aninhamento de regras dentro do seletor pai.
  - O código fica hierárquico, refletindo a estrutura do HTML.
- Atenção:
  - Evite aninhamentos muito profundos (mais de 3 níveis).
  - Pode gerar CSS longo e difícil de manter.

HTML + CSS / SASS (continuação)

Exemplo Comparativo

style.css > ...

```css
nav { background: #333; }
nav ul { list-style: none; }
nav ul li { display: inline-block; }
nav ul li a { color: white; text-decoration: none; }
```

style.scss > ...

```scss
nav {
  background: #333;
  ul {
    list-style: none;
    li {
      display: inline-block;
      a {
        color: white;
        text-decoration: none;
      }
    }
  }
}
```

O CSS final gerado é parecido com do exemplo puro, mas o código fica mais limpo e fácil de entender.

HTML + CSS / SASS (continuação)

Exemplo com o & (referência ao seletor pai)

```html
<body>
  <button>Passe o mouse</button>
  <button class="ativo">Ativo</button>
</body>
```

style.scss

```scss
button {
    color: white;

    &:hover {
        background: #3498db;
    }

    &.ativo {
        background: green;
    }
}
```

style.css

```css
button {
  color: white;
}
button:hover {
  background: #3498db;
}
button.ativo {
  background: green;
}
```

HTML + CSS / SASS (continuação)

Reutilizando Código com Mixins

- Mixins = funções do CSS (reutilização de código).
- Podem receber parâmetros (valores dinâmicos).
- Usados para aplicar estilos repetidos em diferentes elementos.
- Chamados com @include.

HTML + CSS / SASS (continuação)

Reutilizando Código com Mixins

- Problema no CSS puro:
  - Muitas vezes repetimos blocos de estilos (ex: border-radius, box-shadow, gradientes).
- Isso gera código duplicado e difícil de manter.
- Solução no SASS:
  - Criar um @mixin (bloco de código que pode ser chamado em vários lugares).
  - Usar @include para aplicar esse mixin.
  - Pode ser fixo (sempre aplica o mesmo estilo) ou parametrizado (aceita valores diferentes).

HTML + CSS / SASS (continuação)

Exemplo prático - Mixin simples

```html
<body>
    <div class="container">
        <h1>TESTE</h1>
    </div>
</body>
```

style.scss > ...

```scss
@mixin centralizar {
    display: flex;
    justify-content: center;
    align-items: center;
}

.container {
    @include centralizar;
    height: 200px;
    background: gray;
}
```

Resultado: qualquer elemento que usar **@include centralizar** terá conteúdo centralizado.

HTML + CSS / SASS (continuação)

Exemplo prático - Mixin com parâmetro

```html
<body>
    <button>BOTÃO</button>
    <div class="card">
        <h1>CARTÃO</h1>
    </div>
</body>
```

style.scss > ...

```scss
@mixin borda-redonda($raio) {
    border-radius: $raio;
}

button {
    @include borda-redonda(10px);
}

.card {
    @include borda-redonda(20px);
    background-color: gray;
}
```

Isso gera botões e cards com cantos arredondados diferentes, sem duplicar código.

Reaproveitando Estilos com @extend

- @extend permite que uma classe herde os estilos de outra.
- Evita duplicação de código.
- Bom para elementos que compartilham a mesma base, mas precisam de pequenas diferenças.
- Diferença em relação a mixins:
  - Mixins copiam o código.
  - @extend une seletores no CSS final.

HTML + CSS / SASS (continuação)

Reaproveitando Estilos com @extend

- Problema comum no CSS:
  - Criamos várias classes com estilos muito parecidos (ex: alertas de erro, aviso e sucesso).
  - Acabamos copiando e colando o mesmo bloco várias vezes.
- Como o @extend resolve:
  - Criamos uma classe base com os estilos em comum.
  - Outras classes herdam esses estilos com @extend.
  - Cada classe adiciona apenas o que é diferente.
- Diferença prática:
  - Mixins = copiar/colar código.
  - @extend = mesmo seletor com várias classes.

HTML + CSS / SASS (continuação)

Exemplo prático - Mensagens

style.scss > ...

```scss
.mensagem {
    padding: 10px;
    border: 1px solid #ccc;
    font-weight: bold;
}

.alerta {
    @extend .mensagem;
    background: red;
    color: white;
}

.sucesso {
    @extend .mensagem;
    background: green;
    color: white;
}
```

```html
<body>
    <div class="mensagem">Mensagem genérica</div>
    <div class="alerta">Alerta de erro!</div>
    <div class="sucesso">Operação concluída com sucesso!</div>
</body>
```

Mensagem genérica

Alerta de erro!

Operação concluída com sucesso!

HTML + CSS / SASS (continuação)

Organizando Estilos com Parciais e Importações

- Em projetos grandes, não é prático ter tudo em um único arquivo CSS.
- O SASS permite dividir o código em vários arquivos.
- Arquivos auxiliares = Parciais (começam com _).
- Usamos @use ou @import para juntar tudo em um arquivo principal.
- Benefício: organização, reaproveitamento e manutenção facilitada.

HTML + CSS / SASS (continuação)

Organizando Estilos com Parciais e Importações

- Problema no CSS puro:
  - Normalmente acabamos com um style.css gigantesco (milhares de linhas).
  - Difícil de localizar e alterar algo.
- Solução no SASS:
  - Criar arquivos separados para cada parte do estilo:
    - Cores, Mixins, Componentes, Layout
  - Esses arquivos são chamados de parciais.
  - No final, um único main.scss importa todos e gera o style.css.
- Sobre os parciais:
  - Nome sempre começa com _ (ex: _cores.scss, _mixins.scss).
  - Eles não geram CSS sozinhos, só são incluídos dentro de outro.
- Formas de importar:
  - @import: ainda funciona, mas está em desuso.
  - @use (recomendado): mais moderno, evita conflitos de nomes.

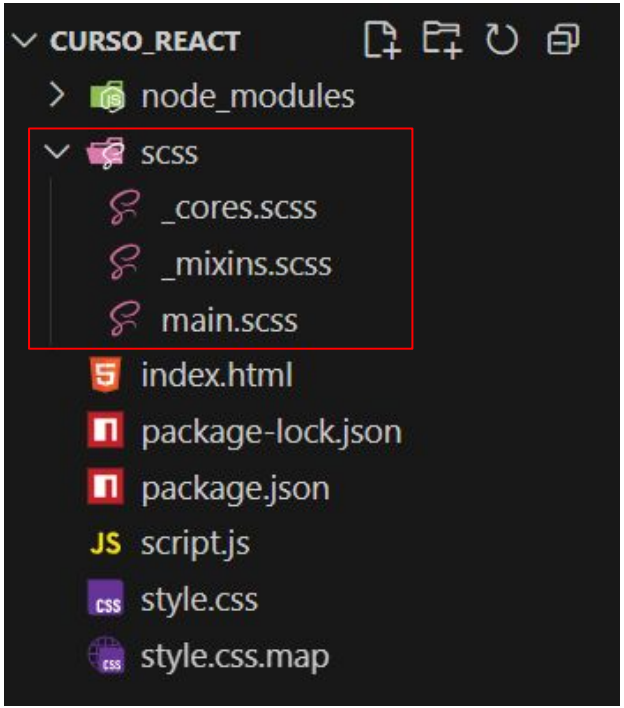
HTML + CSS / SASS (continuação)

Organizando Estilos com Parciais e Importações

- Em projetos grandes, não é prático ter tudo em um único arquivo CSS.
- O SASS permite dividir o código em vários arquivos.
- Arquivos auxiliares = Parciais (começam com _).
- Usamos @use ou @import para juntar tudo em um arquivo principal.
- Benefício: organização, reaproveitamento e manutenção facilitada.

HTML + CSS / SASS (continuação)

Estrutura



Com essa estrutura, rode o comando abaixo do diretório principal do projeto para compilar:

npx sass scss/main.scss style.css

HTML + CSS / SASS (continuação)

Exemplo de Código

```html
<body>
    <h1>Meu site</h1>
</body>
```

scss > _cores.scss > ...

```scss
$cor-primaria: #3498db;
$cor-secundaria: #2ecc71;
```

scss > _mixins.scss > ...

```scss
@mixin centralizar {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

scss > main.scss > ...

```scss
@use 'cores';
@use 'mixins';

body {
    background: cores.$cor-primaria;
    @include mixins.centralizar;
}
```

ATÉ A PRÓXIMA AULA!

Front-end - Design. Integração. Experiência.

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