Curso de SASS

do Básico ao Avançado

O que é um pré-processador?

- O **pré-processador** é um software que recebe uma entrada de dados e gera uma saída;
- Geralmente **modificando os dados** de uma forma que possa ser aceita por outro software;
- O **SASS** é um pré-processador de CSS;
- Ele recebe instruções avançadas, que **não são entendidas pelo CSS**, e transforma em CSS puro;

O que é SASS?

- O **SASS** é um pré-processador de CSS;
- Muitas funcionalidades desse recurso vão aumentar nossa produtividade de código para o desenvolvimento web;
- Temos: **variáveis, mixins, nesting, herança e muito mais**;
- Vários projetos grandes utilizando SASS, como o **Bootstrap**;
- Com SASS **escrevemos menos código** e geramos um CSS de qualidade;

Benefícios/recursos do SASS

- **Pré-processamento:** geração de CSS otimizado e válido;
- **Variáveis:** reaproveitamento de estilos em todo o projeto, facilidade de manutenção;
- **Nesting:** encadeamento de seletores de CSS;
- **Partials:** Divisão de responsabilidades no CSS;
- **Mixins:** espécie de função, para reaproveitamento de estilo;
- E muito mais!

SASS x SCSS x CSS

- O **SASS e o SCSS são bem parecidos**, porém o SASS possui mais recursos e tem uma sintaxe mais simples;
- Ambos têm o mesmo papel, **trazer funcionalidades novas ao CSS**;
- Podemos pré-processar SCSS com o SASS também;
- No fim, todo o código gerado vira **CSS puro** e aceito nos navegadores;
- E é possível que futuramente estas funcionalidades de SASS e SCSS, ou outros pré-processadores, estejam no CSS! :)

Instalação de dependências

- Neste curso vamos precisar de poucos recursos para criar os projetos e rodar o SASS;
- Apenas o **Node.js** e o **npm** já são o suficiente;
- O editor de código será o **VS Code**, tem uma **extensão** que facilita o código em SASS e **terminal integrado**;
- Porém não é obrigatório :)

Instalação do SASS

- Vamos instalar o **SASS** com o **Node (npm) de forma global**;
- Para isso utilizamos o seguinte comando:

  npm install -g sass
- Agora podemos executar o SASS em qualquer projeto;
- Faça o teste com o comando **sass -v**
- Isso resulta na **versão do SASS** instalada na sua máquina;

Extensão SASS no VS Code

- Para melhorar nossa experiência de programação em SASS vamos instalar uma **extensão**;
- O nome dela é **Sass** e está disponível gratuitamente no marketplace;
- Teremos funcionalidades como: auto complete, highlight de código, formatação e muito mais;
- Agora será muito mais fácil programar SASS =)

Primeiro projeto com SASS

- Vamos criar nosso primeiro projeto!
- Abra uma pasta sem arquivos;
- Vamos criar a pasta **sass** e também a pasta **css**;
- E também vamos precisar de um arquivo **index.html** que vai consumir o CSS gerado pelo SASS;

Debug de SASS

- O debug de SASS é **muito simples**!
- Se erramos algo no código, o **CSS não vai compilar corretamente**;
- O **SASS vai imprimir um texto no HTML** com informações de onde erramos;
- Assim identificamos o problema, corrigimos e compilamos novamente;

Como seguir o curso

- **Faça todo o código do curso** no seu PC também;
- **Crie exemplos diferentes** com as funcionalidades ensinadas na aula;
- **Crie projetos** com os conhecimentos do curso;
- **Se a aula estiver difícil:** Ouça primeiro e depois veja novamente copiando e executando o código;

Introdução

Conclusão da seção

Conceitos Fundamentais

Introdução da seção

Sintaxe teoria

- O SASS aceita sintaxe de **SCSS** e também a chamada **identada**;
- A sintaxe do SCSS é muito parecida com a do CSS normal, já a identada suprime o uso de **chaves e ponto e vírgula**;
- O SCSS utiliza arquivos **.scss**;
- A identada utiliza arquivos **.sass**;

Criando e compilando .scss

- Vamos criar um mini projeto utilizando o SCSS!

Criando e compilando .sass

- Vamos criar um mini projeto utilizando o modo identado!

Comentários no SASS

- Os comentários no SASS tem algumas funcionalidades a mais que no CSS;
- **//** -> Não é incluso no CSS;
- **/*** -> É incluso no CSS;
- **/*!** -> Incluso no CSS, até no modo de compressão;
- Podemos **interpolar valores** nos comentários também, ex: #{1 + 3}

Minificando SASS

- Podemos gerar **CSS minificado** facilmente com o interpretador;
- Na saída do comando de watch, vamos mudar a extensão:

  De: styles.css

  Para: styles.**min.**css

Conceitos Fundamentais

Conclusão da seção

Regras de estilo

Introdução da seção

Nesting

- O **nesting** é a possibilidade de aninhar seletores;
- Deixando mais lógica a estilização de elementos, pois **segue o padrão de como o HTML está desenhado** pelas tags;
- Tome **cuidado com o uso excessivo** de nesting, pode confundir em vez de ajudar;

Listas com nesting

- Podemos separar os elementos com **vírgula** no nesting;
- O SASS vai entender que queremos selecionar mais de um elemento seguindo o padrão de aninhamento;
- Tanto para os **níveis superiores** quanto para os **níveis inferiores**;

Combinators com nesting

- Os símbolos de combinators (**+, >, ~**) também podem ser utilizados com nesting;
- O SASS consegue interpretar corretamente o estilo que cada elemento vai receber por meio das combinações;

Parent selector

- Um seletor especial que serve para se referir ao **elemento externo/elemento pai**;
- Utilizamos o símbolo **&**;
- Podemos criar **hovers** facilmente na mesma regra, por exemplo, sem precisar criar uma nova para o efeito;

Adicionando sufixos

- Podemos utilizar o **parent selector** para adicionar sufixos também;
- Desta forma criamos **classes variantes** e mais complexas do elemento alvo;
- Por exemplo **botões diferentes**:

```
.button
    &-danger
    &-success
```

Seletor de placeholder

- Estes seletores **não criam código CSS**;
- Podem ser utilizados para **estender alguma outra classe**;
- O símbolo para o placeholder é o **%;**
- Ou seja, quando atribuímos a um elemento um extend de um placeholder, estamos **transferindo os estilos para este elemento**;

Regras de estilo

Conclusão da seção

Variáveis

Introdução da seção

O que são variáveis?

- Um recurso para **salvar um valor na memória do computador** e utilizá-lo posteriormente em qualquer lugar do código;
- No SASS utilizamos o símbolo **$** para variáveis;
- Normalmente as variáveis em SASS são declaradas com um **hífen**;

$primary-color: #DDD;

Escopo

- As variáveis declaradas fora de blocos **podem ser acessadas em qualquer bloco**;
- Já as variáveis declaradas em um bloco só podem ser acessadas pelo bloco e seus elementos filhos;
- Normalmente declaramos **variáveis globais**;

Shadowing

- É um recurso que permite ter **duas variáveis com mesmo nome**;
- A variável com **escopo global** tem o seu valor mantido para os demais elementos;
- Já a **variável local** tem o valor válido apenas para o seu bloco;

Módulo de variáveis

- Podemos **externalizar as variáveis em um arquivo**;
- Então ele será responsável pela **declaração de variáveis**;
- Adicionando uma organização maior ao nosso projeto;
- Vamos chamar o arquivo de variáveis com **@import**;

Interpolação

- Interpolação é o recurso de **substituir valores de forma dinâmica** no código;
- Utilizamos a sintaxe **#{valor}** em SASS;
- Podemos interpolar variáveis, por exemplo;
- Desta forma conseguimos deixar nosso código ainda mais dinâmico;

Variáveis

Conclusão da seção

Funcionalidades do SASS

Introdução da seção

Partials

- Podemos criar um arquivo e invocar o mesmo em um outro arquivo de SASS;
- Esta técnica é chamada de **partials**;
- Por convenção os partials começam com underline, exemplo: **_button.sass**
- Podemos invocar um partial com a função **@import**;
- Em @import podemos omitir o underline;

Mixins

- **Mixins** são como funções, podemos definir um código que poderá ser reutilizado ao longo do projeto;
- A sintaxe é:

  @mixin nome

  regras
- Podemos criar um arquivo apenas com mixins, e importar ele no projeto principal;

Mixins com argumentos

- Os mixins podem receber **argumentos**, deixando nosso código ainda mais dinâmico;
- Os argumentos devem ficar **depois do nome e entre parênteses**;
- Esta é a sintaxe:

  @mixin calc($a, $b)

Argumentos opcionais

- Os mixins podem receber **argumentos opcionais**, ou seja, que já tem um valor pré-definido;
- Se não passarmos um valor, ele vai utilizar o já definido;
- Estes argumentos devem ficar **depois dos obrigatórios**;
- Veja:

  @mixin optional($a, $b: 10)

Argumentos pelo nome

- Podemos utilizar um mixin e passar os **argumentos pelo seu nome**;
- Esta técnica é útil **quando há vários argumentos opcionais**;
- Utilizamos desta forma:

  @include calc(100px, $color: red)

Herança

- Podemos criar uma herança com **@extend**;
- É interessante utilizar em conjunto dos **placeholder selectors (%)**;
- Veja um exemplo:

```
%red-button
        regras
.button
        @extend %red-button
```

Operações complexas

- Podemos realizar **operações matemáticas complexas** em uma regra de SASS;
- O resultado final será calculado e **transferido para CSS**;
- Veja um exemplo:

  width: 300px / 10px + 5%

Funções

- Podemos também criar **funções** no SASS;
- Definida pela sintaxe **@function nome(args)**;
- Possui a possibilidade de retornar valores com **@return**;
- As funções parecem com mixins, porém elas tem o papel de realizar operações como cálculos por exemplo;

Funções: argumentos opcionais

- Como nos mixins as funções também podem ter **argumentos opcionais**;
- Onde colocamos o valor inicial nos mesmos;
- Eles devem ficar **à direita dos obrigatórios**;
- Veja a sintaxe:

  @function test($color, $px: 10px)

Funções: argumentos pelo nome

- Podemos utilizar uma função e passar os **argumentos pelo seu nome**;
- Esta técnica é útil **quando há vários argumentos opcionais**;
- Utilizamos desta forma:

  calc($height: 100px, $color: red)

Ativação de erros

- Podemos **gerar um erro com o SASS**;
- Uma situação interessante é: quando criamos uma função e ela precisa de alguns valores mínimos para um argumento;
- Então caso o programador não insira corretamente, disparamos um erro com **@error**;
- Podemos **interpolar valores** na mensagem também;

Ativação de avisos

- Podemos **gerar um aviso com o SASS**;
- A situação pode ser a mesma que a do error, mas este método é menos invasivo, ele não para a aplicação, **mostra apenas no console**;
- Então caso o programador não insira corretamente, disparamos um erro com **@warn**;
- Podemos **interpolar valores** na mensagem também;

Ativação de debug

- Podemos **ativar um bugger no SASS**;
- A chamada do recurso é por **@debug**;
- Vamos imprimir uma mensagem no console, com **valores interpolados** ou não;
- Parecido com o warning, mas serve para **depuração de código**;

if e else no SASS

- Podemos criar uma **estrutura de condição** no SASS também;
- Utilizando o **@if** criamos a condição;
- E o **@else** quando a condição do if for falsa;
- Temos também a opção de **@else if**;
- Estas regras funcionam como em outras linguagens de programação;

each no SASS

- Podemos criar uma **estrutura de repetição em listas** no SASS também;
- Utilizando o **@each** criamos a estrutura;
- Podemos repetir uma **criação de classe n vezes**, por exemplo;
- Veja a sintaxe (onde $sizes é a lista de valores):

  @each $size in $sizes

for no SASS

- Podemos criar uma **estrutura de repetição** no SASS também;
- Utilizando o **@for** criamos a estrutura, que se repete quantas vezes for necessário, por meio de uma condição;
- Veja a sintaxe;

  @for $i from 1 through 3 (repetir de 1 até chegar a 3)

  código

Funcionalidades do SASS

Conclusão da seção

Arquitetura SASS

Introdução da seção

@use

- A diretiva **@use** é semelhante ao **@import**, porém mais nova e performática (lançou no fim de 2019);
- Também é **indicada pela documentação do SASS** como mais indicada para importar arquivos;
- O @use deve ser sempre a primeira regra do arquivo;
- E nos dá acesso aos **módulos do SASS**, que aprenderemos futuramente;

Live Server

- **Live Server** é uma extensão do VS Code;
- Que cria um **servidor temporário** em nossa máquina;
- Facilitando o trabalho com os projetos;
- Não vamos mais utilizar o caminho relativo, e sim o **IP** que o Live Server nos dá para acessar;
- Isso vai eliminar os eventuais **erros de path** dos diferentes sistemas operacionais que podemos trabalhar;

SMACSS

- **SMACSS** é referência que vamos utilizar para arquitetar nosso SASS;
- Neste padrão o CSS é dividido em partes:
- **Base**: regras básicas de estilo, elementos gerais;
- **Layout**: Elementos que formam a página, ex: header;
- **Module**: Componentes menores;
- **State** e **Theme**: regras que sobrepõe outras e cores do tema;
- Vamos adaptar e trazer isso para a realidade do SASS;

O arquivo principal

- O arquivo principal vai **apenas realizar importações**, não vamos adicionar estilos a ele;
- Podemos chamá-lo de **main.sass** ou **app.sass**;
- As pastas de outros módulos iniciam com números, por exemplo: 0-plugins;

Estrutura de pastas

- Estas são as pastas que vamos criar e também os significados de cada uma:
- **0-plugins**: Frameworks ou códigos de terceiros;
- **1-base**: Arquivos de estilo base;
- **2-layouts**: Arquivos para elementos que formam o layout da página;
- **3-modules**: Componentes das páginas;

O arquivo principal do subdiretório

- O arquivo principal do subdiretório também **só irá importar arquivos**;
- Isso vai **diminuir os imports** no nosso arquivo principal;
- Organizando ainda mais a estrutura;
- Nomenclatura para **1-base** => **base.sass**

Estilos de base

- Vamos agora criar alguns estilos de base;
- Podemos estilizar tags como: **body, a, p**;
- **Não utilizamos** classes nem ids;
- A ideia é **pré-estilizar nosso HTML**;
- Podemos estilizar elementos em arquivos separados;

Estilos de layout

- Vamos agora criar alguns estilos de layout;
- Aqui **podemos adicionar ids e classes** as regras;
- A ideia é estilizar componentes maiores, como o cabeçalho da página (header);
- Novamente podemos **separar elementos em arquivos distintos**;

Estilos de module

- Vamos agora criar alguns estilos de module;
- Aqui **podemos adicionar ids e classes** as regras;
- Vamos colocar nossos **componentes menores**: cards, botões, algum elemento na sidebar;
- Podemos também separar cada um deles em um arquivo;

Arquitetura SASS

Conclusão da seção

Valores e operadores

Introdução da seção

Números

- Os números em SASS são compostos por duas partes: o **número** e a **unidade**;
- Exemplo: 100% - 100 número - % unidade;
- Algumas unidades são **incompatíveis**, como rem e px;
- A precisão de números vai até **10 casas**;

Strings

- String são os **textos** que escrevemos;
- As strings são inseridas **entre aspas** e também **sem aspas**;
- “Roboto” (com aspas), hidden (sem aspas);
- Podemos **interpolar** valores em string;
- Podemos **concatenar** strings com +;

Cores

- A **cor é considerada um tipo de dado** em SASS;
- Podemos inserir em **hexadecimal, nome da cor, rgb, hsl**;
- Todos estes formatos também são aceitos em CSS;

Listas

- As listas em SASS são como **arrays**;
- Podemos declarar em uma variável com **valores separados por vírgula**;
- $arr: 1px, 2px, 3px
- Utilizamos as listas para trabalhar com **estruturas de repetição** (each);

Maps

- Os maps em SASS são como **objetos**;
- Declaramos entre parênteses, e cada item precisa de uma **chave e valor**;
- Exemplo: $map: (“chave”: 1, “teste”: 50)
- Podemos também utilizar maps com **estruturas de repetição**;

Booleanos

- Os booleanos são representados por **true e false**;
- Toda **expressão lógica** retorna um verdadeiro ou false;
- Por exemplo: 10 > 2 (retorna true)
- Em **if's** no SASS podemos observar a utilização dos booleanos;

Igualdade e diferença

- Caso haja a necessidade de comparar valores, se são **iguais ou diferentes**, podemos utilizar **==** ou **!=**;
- Veja um exemplo:

  @debug 100px != 10rem // true

  @debug “Helvetica” == Helvetica // true
- Os resultados de operações são sempre **booleanos**;

Maior ou menor

- Caso haja a necessidade de comparar valores, se são **maiores ou menores**, podemos utilizar **<** ou **>**;
- Veja um exemplo:

  @debug 100 > 20 // true
- Os resultados de operações são sempre **booleanos**;
- Podemos adicionar um **igual** para também verificar a igualdade, ex: >=

Operadores numéricos

- Os **operadores numéricos** do SASS são:
- +, -, *, /, %
- Podemos realizar cálculos com os mesmos;
- Tome cuidado com a **compatibilidade de valores**;
- Contas entre parênteses também funcionam em SASS;

Valores e operadores

Conclusão da seção

SASS Modules

Introdução da seção

Módulo color

- O módulo color deve ser importado com **@use ‘sass:color’**;
- Teremos acessos a funções que trabalham com cores;
- **color.adjust**: Aumenta ou diminui alguma propriedade da cor;
- **color.green**: Retorna o quanto de verde tem na cor;
- **color.scale**: Pode deixar mais clara ou mais escura uma cor;
- **color.invert**: Inverte uma cor;

Módulo color parte 2

- Mais alguns exemplos de funções que trabalham com cores:
- **color.mix**: Mistura cores;
- **color.desaturate**: Remove uma % de saturação;
- **color.lighten**: Deixa uma cor mais clara;

Módulo list

- O módulo list deve ser importado com **@use ‘sass:list**;
- Teremos acessos a funções que trabalham com listas / múltiplos valores;
- **list.append**: Adiciona um elemento a uma lista;
- **list.index**: Retorna o índice de um elemento;

Módulo list parte 2

- Mais alguns exemplos do módulo de listas:
- **list.join**: Une duas listas;
- **list.length**: Retorna o tamanho de uma lista;
- **list.nth**: Retorna o enésimo elemento de uma lista;

Módulo map

- O módulo map deve ser importado com **@use ‘sass:map**;
- Teremos acessos a funções que trabalham com maps (objetos);
- **map.merge**: Une dois maps;
- **map.get**: Retorna o valor de uma determinada chave;
- **map.has-key**: Verifica se um map tem uma determinada key;

Módulo math

- O módulo math deve ser importado com **@use ‘sass:math**;
- Teremos acessos a funções que trabalham com operações matemáticas;
- **math.$pi**: valor do PI;
- **math.ceil / floor**: arredonda um número para cima / baixo;
- **math.max / min**: maior /menor número de uma lista;
- **math.random**: Retorna um número aleatório de 0 a 1;

Módulo meta

- O módulo meta deve ser importado com **@use ‘sass:meta**;
- Este módulo trabalha com algumas questões específicas do projeto;
- **meta.global-variable-exists**: verifica se variável existe;
- **meta.inspect**: Inspeciona um valor passado a função;
- **meta.mixin-exists**: verifica se mixin existe;
- **meta.type-of**: verifica o tipo de dado passado como parâmetro;

Módulo string

- O módulo string deve ser importado com **@use ‘sass:string**;
- Este módulo trabalha com textos;
- **string.index**: retorna o índice do elemento passado como parâmetro;
- **string.length**: retorna a qtd de caracteres de uma string;
- **string.to-upper/lower-case**: muda a string para letras maiúsculas / minúsculas;

SASS Modules

Conclusão da seção