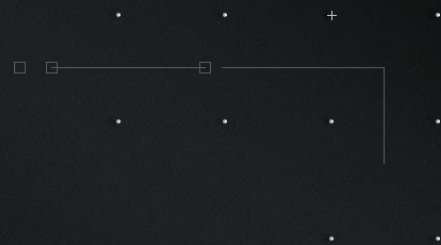




SHIFT

 FIAP





FRONT-END JOURNEY

DESENVOLVIMENTO WEB COM ANGULAR & REACT



FRONT: **PROGRAMAÇÃO**

HTML - CSS - BOOTSTRAP - SASS





ISRAEL MARQUES JÚNIOR

PROFESSOR

- Israel é pós-graduado em Engenharia Web e trabalha com educação há 27 anos. Trabalhou no desenvolvimento de sistemas para desktop e, em seguida, migrou para a criação de aplicações para a Internet.
- Na FIAP, é professor nos cursos de: Sistemas de Informação, Análise e Desenvolvimento de Sistemas, Sistemas para Internet e Jogos Digitais para as disciplinas focadas em front end.

✉ profisrael.copi@fiap.com.br

AGENDA

1

AULA 1

CONTEÚDO – Baixando o Editor + Introdução HTML + CSS

2

AULA 2

CONTEÚDO – Listas + Links + Imagens + Divs + Posicionamento

3

AULA 3

CONTEÚDO – Semântica – Background – Flexbox

4

AULA 4

CONTEÚDO – Tabelas – Formulários

5

AULA 5

CONTEÚDO – Design Responsivo + Media Queries + Mobile First

AGENDA

6

AULA 6

CONTEÚDO – BootStrap: Introdução + Containers + Utilitários

7

AULA 7

CONTEÚDO – BootStrap: Grid + NavBar + Cards + Carousel

8

AULA 8

CONTEÚDO – BootStrap: Accordion + Modal + Formulários

9

AULA 9

CONTEÚDO – Sass: Introdução + Instalação + Conceitos Iniciais

10

AULA 10

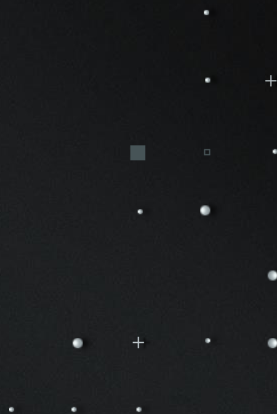
CONTEÚDO – Sass: Estilização + Variáveis + Funcionalidades

AULA 10

Sass - USANDO SUPERPODERES



VARIÁVEIS



Variáveis - criando

Da mesma forma que podemos criar variáveis em CSS, o Sass também permite a utilização desse recurso.

É bem simples definirmos uma variável, basta usarmos o símbolo de \$ para definir o nome da variável e atribuímos o respectivo valor.

Normalmente utilizamos o sinal de hífen (-) para separar os nomes compostos que poderão ser usados na criação das variáveis.

Variáveis - criando

```
<div>  
  <h2>Lorem Ipsum</h2>  
  <p>Lorem ipsum dolor</p>  
</div>
```

```
$first-color: #ccc  
$second-color: #900  
$third-color: #f60
```

```
div  
  padding: 20px  
  width: 20%  
  background-color: $first-color
```

```
h2  
  color: $second-color
```

```
p  
  color: $third-color
```

Variáveis – escopo

Diferente do que acontece no CSS, dentro do Sass as variáveis possuem escopo.

A ideia é que possamos criar variáveis que podem ser utilizada em todas as partes do código, **variáveis globais**, ou então criar variáveis que só sejam enxergadas em uma parte do código, **variáveis de bloco**.

O padrão é que sempre criemos **variáveis globais**, como foi feito no exemplo anterior.

Variáveis – escopo local

```
<div>
  <h2>Lorem Ipsum</h2>
  <p>Lorem ipsum dolor</p>
  <a href="">Link</a>
</div>
```

```
$first-color: #ccc
$second-color: #900
```

```
div
  padding: 20px
  width: 20%
  background-color: $first-color
```

```
h2
  color: $second-color
```

```
p
  $text-color: #090
  color: $text-color
```

```
a
  color: $text-color
```


Variáveis – escopo local

No exemplo anterior teremos um erro, pois a variável `text-color` foi criada no elemento `<p>` e também está sendo usada no elemento `<a>`. Como ele foi criado dentro do `<p>`, o seu escopo é local e não pode ser usada fora dele.

```
p
  $text-color: #090
  color: $text-color
a
  color: $text-color
```

Error: Undefined variable.

```
15 |           color: $text-color
    |                   ^^^^^^^^^^^^^
```

```
sass\style.sass 15:16  root stylesheet
```

Variáveis – shadowing

Como existe o escopo de variáveis em Sass, podemos ter variáveis com o mesmo nome. Nesse caso o escopo global permanecerá valendo para todos os elementos, menos para aqueles que possuírem a variável com o escopo local.

Variáveis – shadowing

```
<div>  
  <p>Lorem ipsum dolor</p>  
</div>
```

```
$bgcolor: #dc143c
```

```
div
```

```
width: 20%
```

```
height: 100px
```

```
background-color: $bgcolor
```

```
padding: 10px
```

```
p
```

```
$bgcolor: #3498dd
```

```
width: 80%
```

```
height: 80px
```

```
background-color: $bgcolor
```

Variáveis – arquivos separados

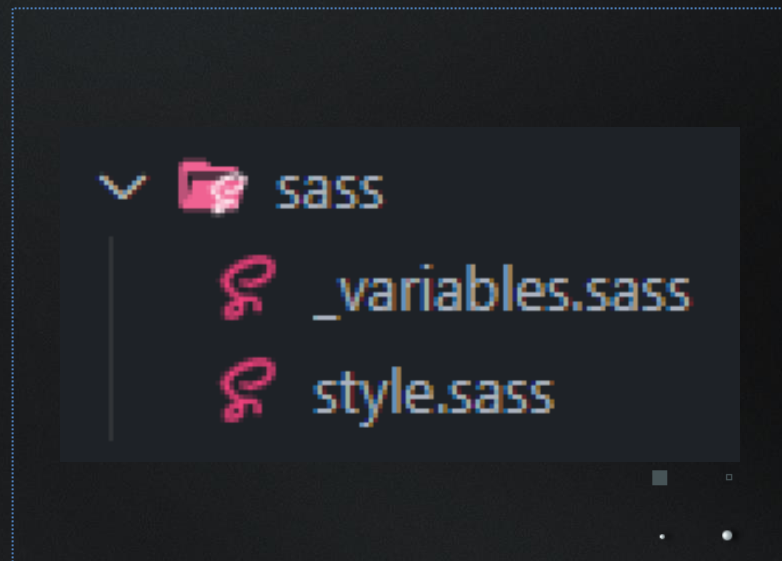
Com o Sass podemos declarar variáveis em um arquivo separado, dessa forma temos uma organização melhor do projeto que está sendo desenvolvido. Essa ideia é bem legal, assim podemos modularizar os arquivos Sass.

Para usarmos essas variáveis, basta utilizar o comando **@import**, ele fará a importação do arquivo de variáveis para o arquivo de estilos que está sendo criado.

Variáveis – criando arquivos separados

Separar arquivos no Sass é muito comum, isso é chamado de **partials**, um arquivo parcial da aplicação.

Por padrão o nome desses arquivos deve iniciar com o sinal de **underline**, seguido do nome desejado e a extensão da sintaxe que está sendo utilizada.



Variáveis – criando arquivos separados

Dentro do arquivo variables.sass, faremos a declaração de todas as variáveis que iremos utilizar em nosso código. O nome do arquivo é apenas um sugestão, você pode usar o nome que quiser, apenas lembre-se de iniciar com underline.

```
$color-text: #dc143c  
$color-title: #791025  
$color-link: #25090e  
$border-card: 1px solid #1d1012
```

Variáveis – importando arquivos separados

Para podermos utilizar as variáveis que foram criadas, precisamos fazer uso do `@import`. Para isso no início do arquivo use: `@import 'nome do arquivo'`.

Não é necessário colocar o underline nem a extensão

```
@import 'variables'
```

```
.card
```

```
border: $border-card
```

```
.title
```

```
color: $color-title
```

Variáveis – interpolação

Podemos usar a interpolação para fazer algum tipo de cálculo, inclusive utilizando variáveis. Para isso usamos: `#{cálculo desejado}`.

```
<div class="card">
  <h3 class="title">Lorem ipsum</h3>
</div>
```

```
$font-size: 16px

.title
  font-size: #{3 * $font-size}
```

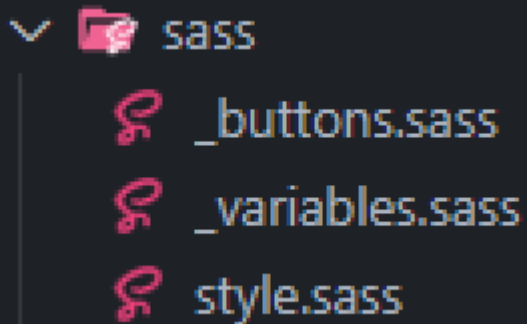
FUNCIONALIDADES

Funcionalidades - partials

Embora já tenhamos criado um partials para separar as variáveis, vamos abordar um pouco mais desse recurso fantástico. Podemos usar os partials para criar componentes que serão utilizados na aplicação, elementos que são bem comuns na página você poderá ter um arquivo específico que montará esse elemento em toda a sua aplicação.

Imagine que podemos ter componentes para botões, containers, áreas mais específicas como cards, etc.

Funcionalidades - partials



A screenshot of a file explorer interface. At the top, there is a folder icon and the text 'sass'. Below it, three files are listed, each with a red icon: '_buttons.sass', '_variables.sass', and 'style.sass'.

```
<button class="btn btn-primary">Botão</button>
```

```
<button class="btn btn-danger">Botão</button>
```

```
<button class="btn btn-success">Botão</button>
```

Funcionalidades - partials

```
$btn-primary-color: #0d6efd
```

```
$btn-danger-color: #dc3545
```

```
$btn-success-color: #198754
```

```
$btn-primary-border: 1px solid #0d6efd
```

```
$btn-danger-border: 1px solid #dc3545
```

```
$btn-success-border: 1px solid #198754
```

```
.btn
```

```
width: 120px
```

```
margin: 10px
```

```
padding: 10px
```

```
border-radius: 5px
```

```
color: #fff
```

```
text-align: center
```

```
&-primary
```

```
background-color: $btn-primary-color
```

```
border: $btn-primary-border
```

```
&-danger
```

```
background-color: $btn-danger-color
```

```
border: $btn-danger-border
```

```
&-success
```

```
background-color: $btn-success-color
```

```
border: $btn-success-border
```

Variáveis – importando arquivos separados

Agora no arquivo de estilização, vamos fazer a chamada dos partials. Eles devem ser importados em ordem, ou seja, o arquivo com as variáveis deverá vir em primeiro lugar.

Pronto, agora temos um componente para botões.

```
@import 'variables'  
@import 'buttons'
```

Funcionalidades - mixins

Mixins são arquivos que têm o mesmo funcionamento de uma função. A ideia é criar um código de estilização que poderá ser reutilizado pela aplicação quantas vezes forem necessárias. Podemos inclusive criar um partial apenas contendo mixins.

Criando um mixin

```
@mixin nome desejado  
  regras a serem aplicadas
```

Incluindo um mixin

```
@include nome do mixin
```


Funcionalidades - mixins

```
<div class="card"></div>
<div class="card"></div>
<h2>Promoção</h2>
```

```
@mixin promotion
```

```
background-color: #f60
```

```
padding: 10px
```

```
font-size: 20px
```

```
text-transform: uppercase
```

```
@import 'mixins'
```

```
.card
```

```
width: 200px
```

```
height: 200px
```

```
border: 1px solid #000
```

```
@include promotion
```

```
h2
```

```
@include promotion
```

Funcionalidades – mixins com argumentos

Como os mixins funcionam como funções, podemos também fazer a passagem de argumentos para eles, dessa forma podemos deixar ainda mais dinâmica essa funcionalidade.

Criando um mixin com argumentos

```
@mixin nome desejado(argumento1, argumento2, ...)  
  regras a serem aplicadas
```

Funcionalidades – mixins com argumentos

```
<div class="card-small"></div>  
<div class="card-big"></div>
```

```
@mixin size($w, $h)
```

```
width: $w
```

```
height: $h
```

```
.card-small
```

```
@include size(150px, 150px)
```

```
border: 1px solid #369
```

```
.card-big
```

```
@include size(300px, 300px)
```

```
border: 1px solid #f60
```

Funcionalidades – mixins com argumentos e valores iniciais

Os argumentos que serão usados nos mixins poderão ser definidos com um valor inicial. Esse valor será substituído caso o mixin seja chamado e receber um novo valor.

Criando um mixin com argumentos

```
@mixin nome desejado(arg1: valor1, arg2: valor2, ...)  
  regras a serem aplicadas
```

Funcionalidades – mixins com argumentos e valores iniciais

```
<div class="card-small"></div>  
<div class="card-big"></div>
```

```
@mixin size($w:100px, $h:100px)  
  width: $w  
  height: $h
```

.card-small

```
@include size
```

```
border: 1px solid #369
```

.card-big

```
@include size(200px, 200px)
```

```
border: 1px solid #f60
```


Funcionalidades – mixins com argumentos pelos nomes

Podemos também passar valores para os argumentos utilizando o nome, isso facilitará na hora de você definir novos valores para os argumentos .

A forma de criação do mixin será a mesma, o que irá mudar é que você pode passar um novo valor usando o nome do argumento. Caso nada seja declarado, o valor padrão definido no mixin será usado.

Funcionalidades – mixins com argumentos pelos nomes

```
<div class="card-small"></div>  
<div class="card-big"></div>
```

```
@mixin size($w:100px, $h:100px)  
  width: $w  
  height: $h
```

.card-small

```
@include size
```

```
border: 1px solid #369
```

.card-big

```
@include size($h:300px)
```

```
border: 1px solid #f60
```

Funcionalidades - herança

Podemos utilizar o conceito de herança para aplicar formatações a um determinado grupo de elementos. Para isso podemos utilizar placeholder selector, que são representados pelo sinal **%**. Uma grande vantagem de utilizar placeholder é que o código CSS só será criado se o seletor for atribuído a um elemento.

Criando um placeholder

% nome desejado
regras a serem aplicadas

Aplicando o placeholder

@extend nome placeholder

Funcionalidades – herança

```
<header>Lorem ipsum</header>  
<footer>Lorem ipsum</footer>
```

%box-pattern

width: 100px

height: 100px

border: 1px solid #369

border-radius: 5px

padding: 10px

text-align: center

header

@extend %box-pattern

background-color: #369

footer

@extend %box-pattern

background-color: #000

Funcionalidades - cálculos

É possível realizar qualquer tipo de cálculo utilizando o Sass, desde que estejamos utilizando medidas do mesmo tipo.

```
<header>Lorem ipsum</header>  
<footer>Lorem ipsum</footer>
```

```
$size: 50px
```

```
header  
  font-size: $size * 2
```

```
footer  
  font-size: $size - 15px
```


Funcionalidades – condicional if

O Sass possui recursos poderosos, e um deles é a possibilidade de montarmos estruturas condicionais com o uso das instruções `@if` e `@else`, como acontece em outras linguagens de programação.

Funcionalidades - condicional if

```
<header>Lorem ipsum</header>  
<footer>Lorem ipsum</footer>
```

header, footer

\$size: 200px

width: \$size

height: \$size

border: 1px solid #000

@if \$size < 200px

background-color: #f60

@else

background-color: #900

Funcionalidades – loop for

Podemos usar a instrução `@for` para criarmos uma estrutura de repetição que fará a estilização de elementos. Para isso devemos definir um valor inicial e um valor final, eles representarão a quantidade de repetições.

Sintaxe

`@for $variável de controle from início through final`
regras a serem aplicadas

Funcionalidades – loop for

```
<div class="div-1"></div>
```

```
<div class="div-2"></div>
```

```
<div class="div-3"></div>
```

```
@for $i from 1 through 3
```

```
.div-#{ $i }
```

```
width: 200px
```

```
height: 200px
```

```
border: 1px solid #000
```

```
margin: 10px
```

```
background-color: #f60
```

```
@if $i == 2
```

```
background-color: #369
```

Funcionalidades – loop each

Podemos usar a instrução `@each` para percorrer uma lista e aplicar formatações baseadas nos valores dessa lista.

A lista pode ser uma variável que receberá um conjunto de valores que poderão ser aplicados a qualquer regra.

No exemplo vamos criar três divs, uma variável que será a lista com três valores definidos e a instrução `@each` que percorrerá essa lista e atribuirá a cada div um tamanho conforme os valores existentes nela.

Funcionalidades – loop each

```
<div class="size-100px"></div>  
<div class="size-200px"></div>  
<div class="size-300px"></div>
```

```
$sizes: 100px, 200px, 300px
```

```
@each $size in $sizes
```

```
.size-#{$size}
```

```
width: $size
```

```
height: $size
```

```
background-color: #f60
```

```
padding: 10px
```

```
margin: 10px
```

Funcionalidades – function

Além do Sass possibilitar o uso de mixins, podemos também criar funções utilizando a instrução `@function`. Embora ambos sejam bem semelhantes, os mixins são mais utilizados na estruturação de elementos, já as funções **retornam** algum tipo de resultado com a instrução `@return`.

Sintaxe

```
@function nome (argumento1, argumento2, ...)
```

```
  regras a serem aplicadas
```

```
  @return .....
```

Funcionalidades – function

```
<section></section>  
<aside></aside>
```

```
@function box-color($size)
```

```
  $back-color: #fff
```

```
  @if $size < 100px
```

```
    $back-color: #f60
```

```
  @else
```

```
    $back-color: #369
```

```
  @return $back-color
```

```
section
```

```
  $size: 90px
```

```
  width: $size
```

```
  height: $size
```

```
  background-color: box-color($size)
```

```
aside
```

```
  $size: 120px
```

```
  width: $size
```

```
  height: $size
```

```
  background-color: box-color($size)
```


OBRIGADO



/icajai

FIAP

Copyright © 2021 | Professor Israel Marques Cajai Júnior

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

SHIFT

FIAP