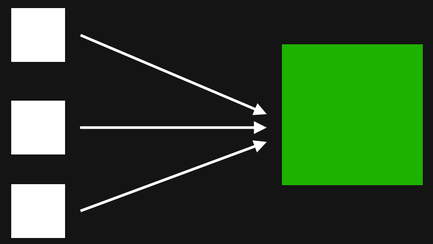
SASS



styles.css

Partials

- _typography.scss
- _colors.scss
- _grid.scss

Output

styles.css

Partials

- _typography.scss
- _colors.scss
- _grid.scss

Input

styles.scss

Output

styles.css

Partials

- _typography.scss
- _colors.scss
- _grid.scss

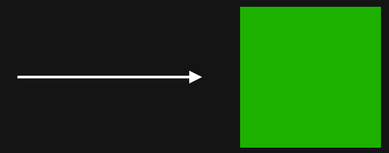
Input

styles.scss

Processo de compilação

Output

styles.css



Como se escreve SCSS / SASS?

Como se escreve SCSS / SASS?

Exactamente igual ao css, mas com
alguns detalhes

Exemplo de scss

```
@mixin button-base() {
  @include typography(button);
  @include ripple-surface;
  @include ripple-radius-bounded;

  display: inline-flex;
  position: relative;
  height: $button-height;
  border: none;
  vertical-align: middle;

  &:hover {
    cursor: pointer;
  }

  &:disabled {
    color: $mdc-button-disabled-ink-color;
    cursor: default;
    pointer-events: none;
  }
}
```

Estrutura de ficheiro

SCSS é baseado em statements. Declarações feitas de forma utilizar certas ferramentas.

Universal Statements
$var - variável. key : value
@if e @each - Fluxos de Condição.
@warn, @error, @debug - at-rules

CSS Statements
h1 {…} - regras normais de CSS
@media, @font-face - @-rules

Top Level Statements
@use - Loading de Módulos
@imports - importar ficheiros
@mixin - receitas mágicas
@functions - Blocos de execução.

$variaveis

Variáveis são locais onde podemos apontar valores para depois os utilizar noutras localizações. Podem ser globais ou locais

```scss
$color: red;

.item {
    background: $color;
}

```

```scss
.item {
  background: red;
}
```

$variaveis

As variáveis podem ser alteradas no scope local do selector, desta forma o valor global permanece sem alterações, ou até mesmo se podem declarar novas variáveis localmente.

```scss
$background: blue;

.item {
    $background: red;
    background: $background
}

```

```scss
.item {
  background: red;
}
```

Nesting

Nesting é criamos relações entre parents e childs diretamente no scss.

```scss
.item {
    .sub-item {
        background: red;
    }
}

```

```scss

.item .sub-item {
  background: red;
}

```

Parent Selector

A utilização do & juntamente com o selector do pai pode afinar o nesting de forma a poupar declarações e a organização conteúdo.

```
.item {
  &:hover {
    background: red;
  }
}
```

```
.item:hover {
    background: red;
}
```

Parent Selector

Podemos também adicionar sufixos com base no parent de forma a organizar o nosso conteúdo.

Útil quando a metodologia de CSS é BEM.

```
.item {
    &--list {
    background: red;
  }
 }
```

```
.item--list {
    background: red;
 }
```

Modules

A vantagem do scss é a possibilidade de escrever o conteúdo em ficheiros separados e conseguir reutilizar noutros locais

```scss
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```scss
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

Mixins

Mixins é o que se pode chamar de receitas. Um conjunto de propriedades, às quais acrescentamos ingredientes (parametros) e conseguimos obter declarações reutilizáveis.

```
@mixin color($color) {
    background: $color;
}
.item {
    @include color(red);
}

```

```
.item {
  background: red;
}
```

Mixins com condições

No SCSS também temos a possibilidade de criar condições, com @if and @else.

É um padrão que podemos utilizar em qualquer lugar do SCSS para conseguir, condicionalmente, representar várias propriedades

```scss
@mixin conditional-color($color) {
    @if $color == red {
        background: red;
    } @else {
        background: blue;
    }
}

.conditional {
    @include conditional-color(red);
}
```

```css
.item {
  background: red;
}
```

Extend and Inheritance

O padrão herança e imutabilidade é algo muito comum na programação. O facto de conseguirmos criar um template que não seja alterado e seja a base para algo é um factor importante e útil no dia-a-dia.

Imagem um botão, que é sempre um botão. Mas o mesmo botão pode ser , noutro local, azul. Não deixa de ser um botão, é só modificado de certa forma.

```
%btn {
  border: 1px solid #ccc;
  padding: 10px;
  background: gray;
  color: #333;
}
```

```
.button--primary {
  %extend btn;
  background: blue;
}
```

```
.button--primary {
  border: 1px solid #ccc;
  padding: 10px;
  background: gray;
  color: #333;
}

```

Interpolação

A **interpolação** é uma forma de **inserir valores dinâmicos** (como variáveis ou expressões) dentro de seletores, nomes de propriedades ou valores — onde normalmente só podias pôr texto estático em CSS.

É um espaço seguro para computar determinados cálculos.

Usa-se
#{parametro | calculo | variável}

Interpolação em SCSS = usar #{$variavel} para misturar código dinâmico dentro de strings de CSS (seja em nomes, propriedades ou valores).

```scss
$tema: dark;

.body-#{$tema} {
  background: #111;
}

$direcao: left;

.div {
  border-#{$direcao}: 1px solid black;
}
```

@for ou @each

É uma maneira de repetir um bloco de código SCSS várias vezes, mudando valores a cada repetição.

Um **loop** é uma estrutura de repetição que te permite gerar CSS de forma dinâmica e automatizada — como se estivesses a "programar" os estilos.

O que é um loop? (@each)

Imaginem que temos uma lista de 4 elementos.

Que queremos representar um de cada vez.

Um loop é quando repetimos a mesma ação por um determinado número de vezes de forma controlada.

```
$colors: red, blue, green, gray
```

O que é um loop? (@each)

Imaginem que temos uma lista de 4 elementos.

Que queremos representar um de cada vez.

Um loop é quando repetimos a mesma ação por um determinado número de vezes de forma controlada.

O que é um loop? (@each)

Imaginem que temos uma lista de 4 elementos.

Que queremos representar um de cada vez.

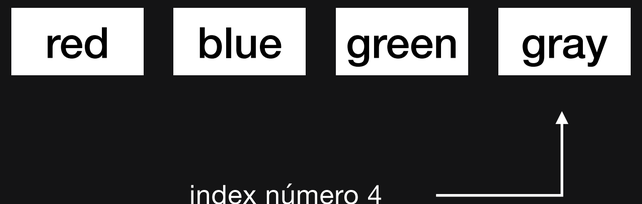
Um loop é quando repetimos a mesma ação por um determinado número de vezes de forma controlada.

O que é um loop? (@each)

Imaginem que temos uma lista de 4 elementos.

Que queremos representar um de cada vez.

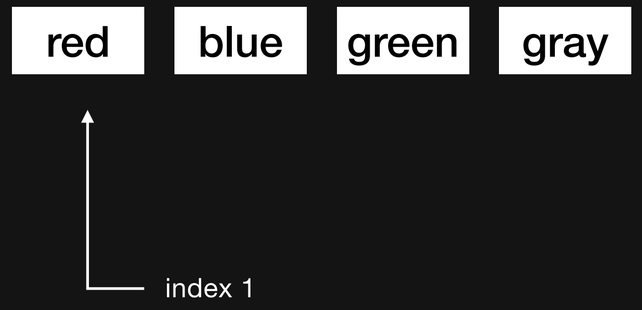
Um loop é quando repetimos a mesma ação por um determinado número de vezes de forma controlada.



O que é um loop? (@each)

Aqui utilizamos um @each, que corre cada index da variável $colors. Percorre todos os elementos da variável, utilizando cada um até que não há mais nenhum disponível.

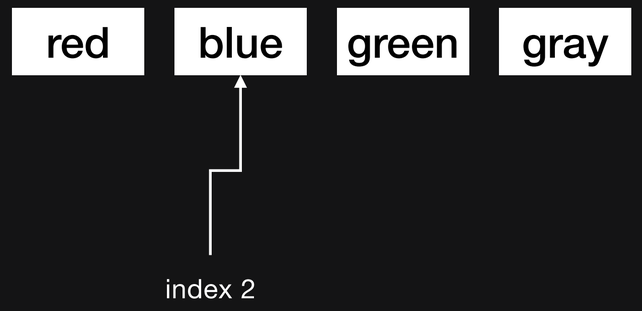
```scss
$colors: red, blue, green, gray

@each $color in $colors {
  .bg-#{$color} {
    background: $color;
  }
}

// output
.bg-red {background: red;}
```



O que é um loop? (@each)

Aqui utilizamos um @each, que corre cada index da variável $colors. Percorre todos os elementos da variável, utilizando cada um até que não há mais nenhum disponível.

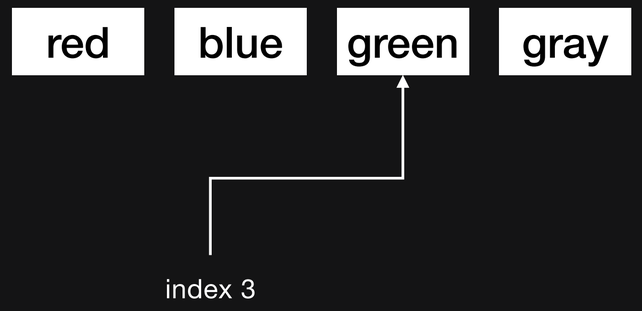
```scss
$colors: red, blue, green, gray

@each $color in $colors {
  .bg-#{$color} {
    background: $color;
  }
}

// output
.bg-red {background: red;}
.bg-blue {background: blue;}
```



O que é um loop? (@each)

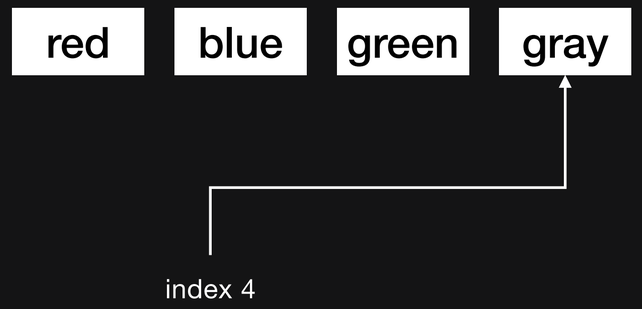
Aqui utilizamos um @each, que corre cada index da variável $colors. Percorre todos os elementos da variável, utilizando cada um até que não há mais nenhum disponível.

```scss
$colors: red, blue, green, gray

@each $color in $colors {
  .bg-#{$color} {
    background: $color;
  }
}

// output
.bg-red {background: red;}
.bg-blue {background: blue;}
.bg-green {background: green;}
```



O que é um loop? (@each)

Aqui utilizamos um @each, que corre cada index da variável $colors. Percorre todos os elementos da variável, utilizando cada um até que não há mais nenhum disponível.




```scss
$colors: red, blue, green, gray

@each $color in $colors {
  .bg-#{$color} {
    background: $color;
  }
}

// output
.bg-red {background: red;}
.bg-blue {background: blue;}
.bg-green {background: green;}
.bg-gray {background: gray;}
```

O que é um loop? @for

Um ciclo @for é similar, mas utilizar um ponto de referência.

O loop corre de forma controlada até atingir esse paramêtro de forma incremental.

```scss
@for $i from 1 through 4 {
  .size-#{$i} {
    width: calc(100% / ${i})
  }
}

// output
.size-1 { width: 100% }
```

O que é um loop? @for

Um ciclo @for é similar, mas utilizar um ponto de referência.

O loop corre de forma controlada até atingir esse paramêtro de forma incremental.

```scss
@for $i from 1 through 4 {
  .size-#{$i} {
    width: calc(100% / ${i})
  }
}

// output
.size-1 { width: 100% }
.size-2 { width: 50% }
```

O que é um loop? @for

Um ciclo @for é similar, mas utilizar um ponto de referência.

O loop corre de forma controlada até atingir esse paramêtro de forma incremental.

```scss
@for $i from 1 through 4 {
  .size-#{$i} {
    width: calc(100% / ${i})
  }
}

// output
.size-1 { width: 100% }
.size-2 { width: 50% }
.size-3 { width: 33.3% }
```

O que é um loop? @for

Um ciclo @for é similar, mas utilizar um ponto de referência.

O loop corre de forma controlada até atingir esse paramêtro de forma incremental.


```scss
@for $i from 1 through 4 {
  .size-#{$i} {
    width: calc(100% / ${i})
  }
}

// output
.size-1 { width: 100% }
.size-2 { width: 50% }
.size-3 { width: 33.3% }
.size-4 { width: 25% }
```

O que é um loop? @for

No ciclo @for, a definição da palavra **through** inclui o index número 4.

Se usamos o **to <from 1 to 4> ,** o index número 4 não é incluído.

É a definição de **<=** ou **<** na programação.


```scss
@for $i from 1 to 4 {
  .size-#{$i} {
    width: calc(100% / ${i})
  }
}

// output
.size-1 { width: 100% }
.size-2 { width: 50% }
.size-3 { width: 33.3% }
```