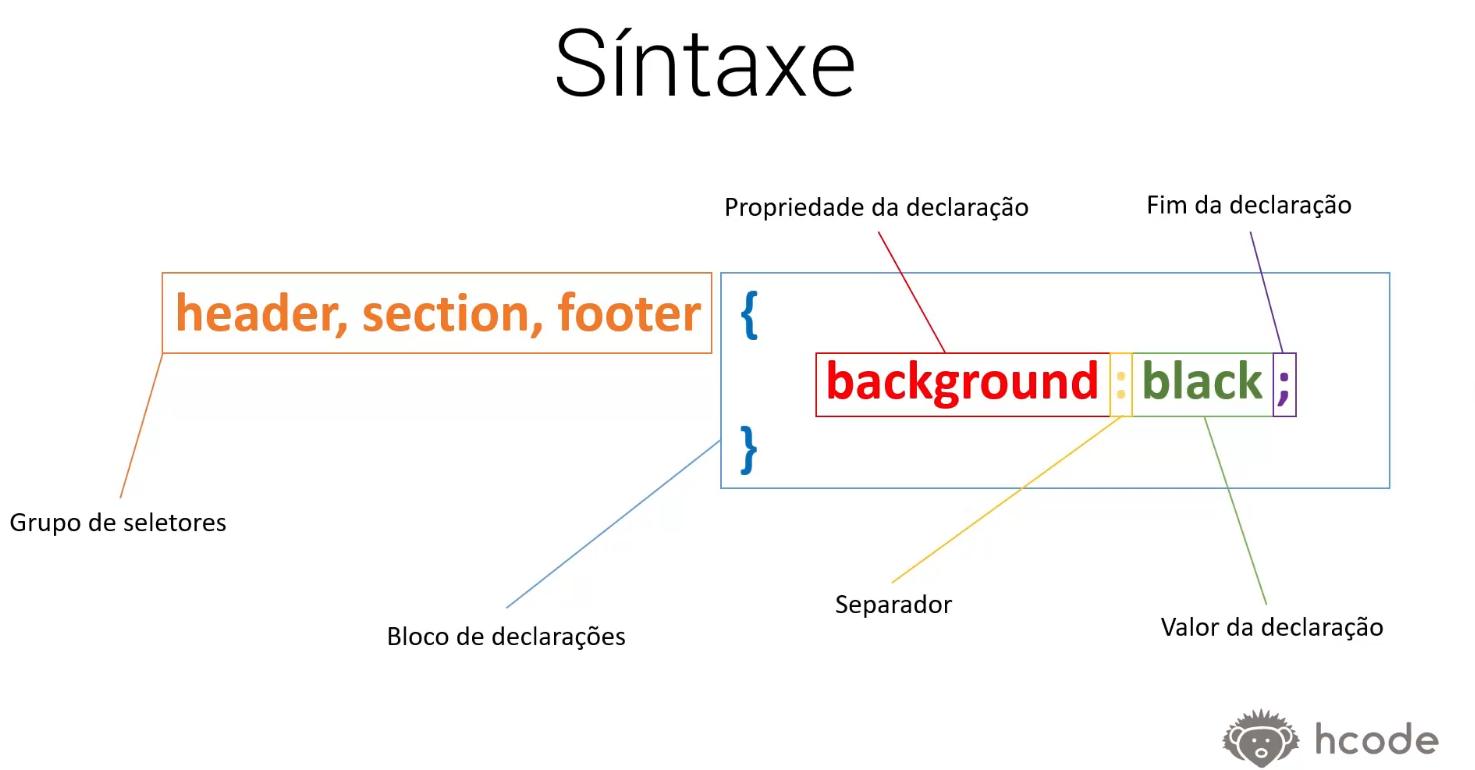
**CSS**

**13/12 – Introdução a CSS3**

CSS é a sigla para **Cascading Style Sheets**, é uma linguagem de folha de estilo usada para descrever a apresentação de um documento escrito em HTML ou XML. O CSS descreve como os elementos devem ser renderizados na tela, no papel, na fala ou em outra mídia.

Podemos comparar uma aplicação web como a construção de um prédio: a parte estrutural, as colunas, paredes, parte elétrica são como se fosse a parte do HTML, que é o esqueleto de uma aplicação. Já pintar as paredes, decorar e o planejamento de arquitetura interna é função do CSS, que é o estilo do site.

O CSS possui uma sintaxe, que é a seguinte:



**TIPOS DE SELETORES**

O CSS também possui **tipos de seletores**, que são: universal, tipo, classes, atributos, pseudo e ID.

**Seletor universal** (seleciona todas as tags possíveis para aplicar o estilo)**:**

\* {

background: black;

}

**Seletor por tipo** (seleciona o tipo da tag para aplicar o estilo)**:**

body {

background: black;

}

**Seletor por classe** (aplica o estilo a uma determinada classe, que sempre começa com ponto final (.))**:**

.bg-black {

background: black;

}

**Seletor por atributo** (aplica o estilo sempre onde há um determinado atributo)**:**

[type=”text”] {

background: black;

}

**Seletor pseudo** (seletores que especificam um estado especial do elemento selecionado. Aplica o estilo com uma determinada condição, é caracterizado pelos dois pontos (:))**:**

button:hover {

color:blue;

}

**Seletor por ID** (aplica o estilo a um determinado campo já identificado pelo atributo id no HTML, é caracterizado pela hashtag (#) no seletor)**:**

#conteudo {

background: black;

}

**CORES**

No CSS, podemos chamar as cores de várias formas, como apresentado a seguir:

**Cores por nome** (os navegadores modernos suportam no total 140 cores por nome).

white, black, red, green, blue

**Cores por RGB** (para chamar esse tipo de cor, é necessário usar uma função no CSS).

rgb(red, green, blue)

red = 0 a 255 de vermelho

green = 0 a 255 de verde

blue = 0 a 255 de azul

Em RGB, as principais cores ficam assim:

white = rgb(255, 255, 255)

black = rgb(0, 0, 0)

red = rgb(255, 0, 0)

green = rgb(0, 255, 0)

blue = rgb(0, 0, 255)

Quanto mais próximo de 0, a cor obterá um tom escuro. Quanto mais próximo de 255, a cor obterá um tom claro.

**Cores por hexadecimal** (código iniciado por hashtag e seis letras ou números)

#RRGGBB

00 a FF

0 1 2 3 4 5 6 7 8 9 A B C D E F

Em hexadecimal, as principais cores ficam assim:

white = #FFFFFF

black = #000000

red = #FF0000

green = #00FF00

blue = #0000FF

**Cores por HSL** (assim como em RGB, para chamarmos cores por HSL, usamos uma função)

hsl(H, S, L)

Hue (0 a 360), Saturation (0% a 100%), Lightness (0% a 100%)

Matiz, Saturação e Brilho

Em HSL, as principais cores ficam assim:

white = hsl(0, 100%, 100%)

black = hsl(0, 0%, 0%)

red = hsl(0, 100%, 50%)

green = hsl(120, 100%, 50%)

blue = hsl(240, 100%, 50%)

**Transparência** (determinada pelo comando alpha)

alpha

0 a 1

rgba(red, green, blue, alpha)

#RRGGBBAA

hsla(H, S, L, A)

**UNIDADES DE MEDIDA**

Em CSS, podemos usar as seguintes unidades de medida:

**Length:** essa medida, que em português significa “comprimento” representa as várias unidades diferentes para expressar comprimento no CSS. Os comprimentos sempre serão números seguidos por uma unidade de comprimento, por exemplo: 2px, 4em, 8cm, etc. Algumas propriedades aceitarão números negativos.

Existem dois tipos de unidades de medida: **absolutas** e **relativas.**

**Absolutas:**

* centímetros (cm)
* milímetros (mm)
* polegadas (in)
* pixels (px)
* pontos (pt)
* paicas (pc)

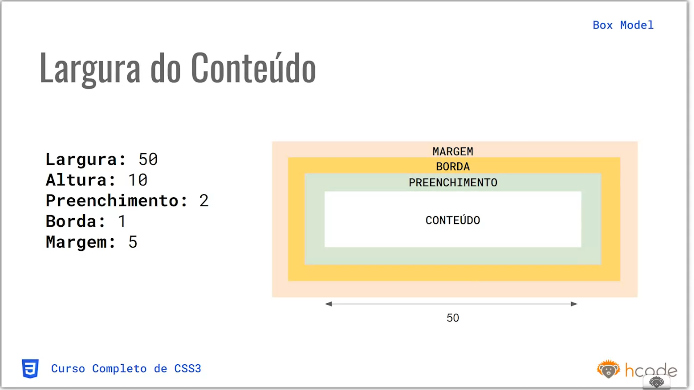
**Relativas:**

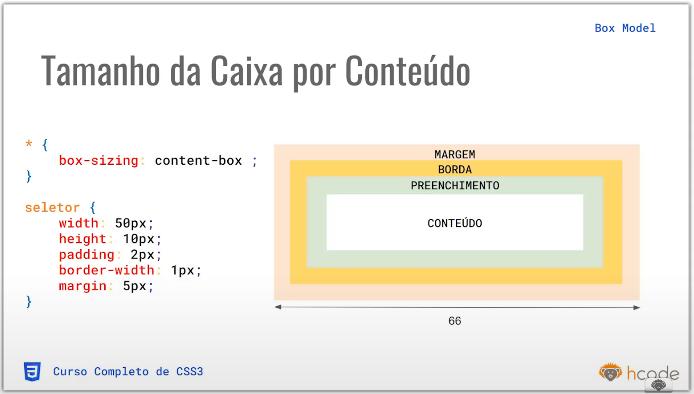
* Porcentagem (%)
* M (em)
* Root M (rem)
* Unidade do Caractere (ch)
* X (ex)
* Largura da Janela de Exibição – viewport width (vw)
* Altura da Janela de Exibição – viewport height (vh)
* Janela de Exibição Mínima (vmin)
* Janela de Exibição Máxima (vmax)
* Fração Restante (fr)

**BOX MODEL**

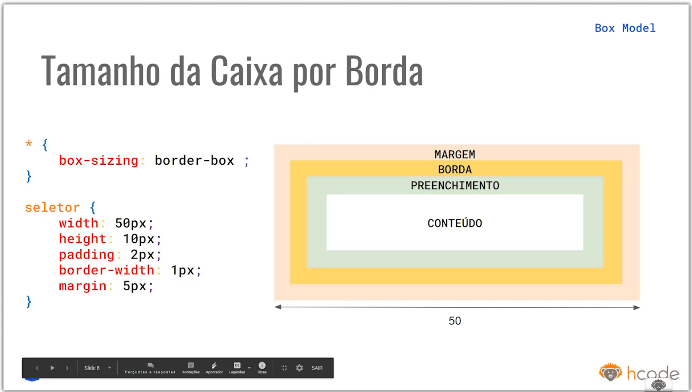
Todos os elementos HTML podem ser considerados como caixas. No CSS, o termo “modelo de caixa” é usado quando se fala em design e layout.

O modelo de caixa CSS é essencialmente uma caixa que envolve todos os elementos HTML. Consiste em: margens, bordas, preenchimento e o conteúdo.

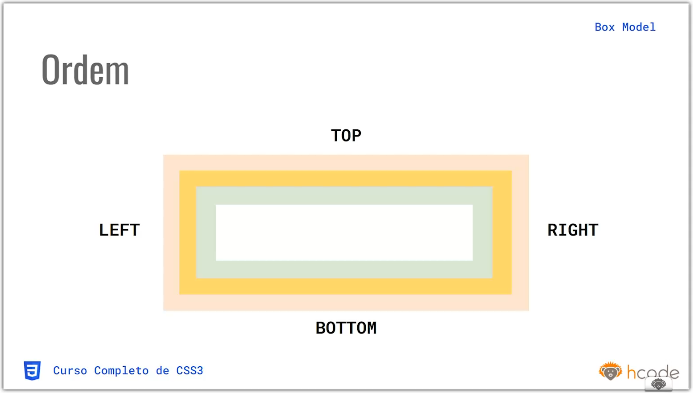




A declaração “box-sizing: content-box” especifica que o meu conteúdo define o tamanho da minha caixa. Com isso, por mais que a largura definida seja 50px, ela será somada com as outras medidas: padding (2 de cada lado = 4px), border-width (1 de cada lado = 2px) e margin (5 de cada lado = 10px). Isso totalizará 16px e será somado á largura definida inicialmente, ficando 66px.



A declaração “box-sizing: border-box” tem um comportamento diferente. Ele respeita a largura definida inicialmente, e diminui um pouco o conteúdo para contabilizar o tamanho do padding, margin e border-width, sem ultrapassar o limite de largura definido.



Ao determinar as medidas, essa é a ordem que usamos.

Exemplos:

padding: 2px 4px 6px 8px

2px em cima

4px na direita

6px embaixo

8px na esquerda

padding: 2px 4px (margin or padding collapse)

2px em cima e embaixo (eixo Y)

4px na direita e na esquerda (eixo X)

padding: 4px

4px em todos os quatro lados

**CSS E HTML**

Nós podemos incorporar o CSS ao documento HTML por três formas:

**Estilo inline**: é quando usamos o atributo style dentro de uma tag. É a primeira em questão de prioridade.

<body style="background-color: red"></body>

**Estilo incorporado:** é quando usamos uma tag <style> para determinarmos o estilo daquele documento HTML. É a segunda em questão de prioridade.

<style>

        body{

            background-color: blue;

        }

</style>

**Estilo externo:** é quando desenvolvemos o nosso código CSS em um outro arquivo específico para CSS, e relacionamos esse arquivo com o documento HTML por meio da tag <link>.

<link rel="stylesheet" href="style.css">

**14/12 – POSITION**

Essa propriedade define como um elemento deve ser renderizado/posicionado na página. Por padrão, todos os elementos são posicionados estaticamente (static), o que faz com que o espaço seja ocupado apenas por ele mesmo.

Além do static, temos: fixed, absolute, relative e sticky como valores possíveis para posicionamento.

*static* = dos cinco valores, este é o único que não será afetado pelas propriedades de deslocamento top, right, bottom e left. Ele vai seguir o fluxo comum da página.

*relative* = utilizando o position relative, o elemento passa a aceitar as propriedades top, bottom, left e right. Com isso, é possível alterar o posicionamento do elemento.

*fixed* = esta posição faz com que o elemento fique posicionado fixamente em relação a janela de visualização, o que fará com que não tenha sua posição afetada mesmo quando o scroll da página for rolado.

*absolute* = esta posição faz com que o elemento fique posicionado fixamente em relação ao ancestral posicionado mais próximo. Caso não exista nenhum elemento ancestral posicionado, ele usará o corpo da página como relação. Isso fará com que ele acompanhe o scroll quando a página for rolada.

*sticky* = esta posição faz com que o elemento se comporte como relative e fixed em determinados momentos. Ele é posicionado em relação até que encontre uma determinada posição na janela de exibição e passe a se comportar como fixo.

**FLEXBOX**

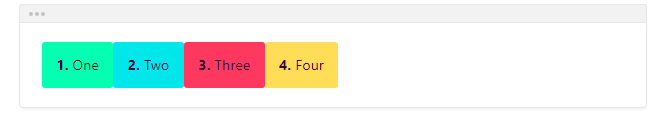
O Módulo de Caixa Flexível, mais conhecido por “Flexbox”, é um método capaz de organizar os elementos de uma interface de modo unidimensional. É possível definir a organização dos elementos como LINHA ou COLUNA e usar recursos para alinhar e distribuir após isso.

EIXOS DO FLEXBOX:

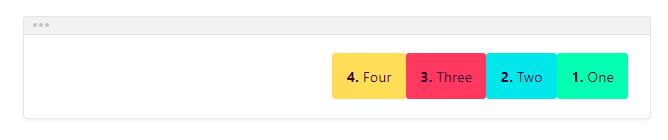
Ao definir a dimensão (row, row-reverse, column, column-reverse) por meio da propriedade flex-direction, este eixo será tratado como o principal e o outro como eixo transversal.

Exemplo:

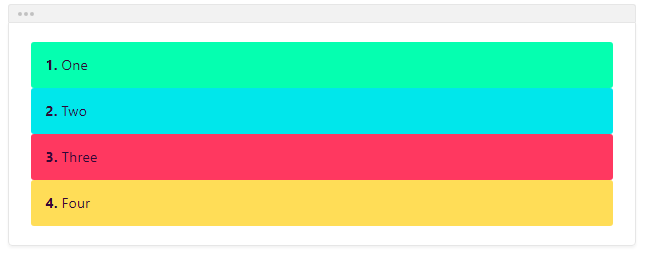
Dentro de um contêiner flex, utilizando o flex-direction **row:**

****

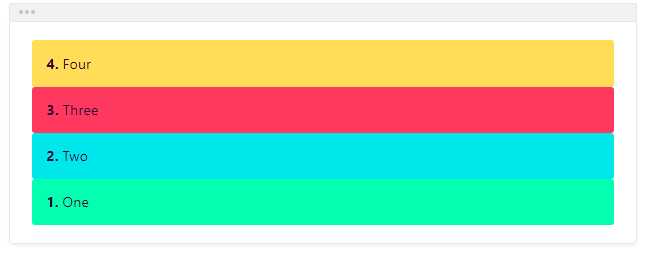
Dentro de um contêiner flex, utilizando o flex-direction **row-reverse:**



Dentro de um contêiner flex, utilizando o flex-direction **column:**

****

Dentro de um contêiner flex, utilizando o flex-direction **column-reverse:**



**VALORES GLOBAIS**

Os valores globais podem ser utilizados em todas as propriedades CSS a menos que tenha alguma regra especifica como exceção. Os valores globais são:

*initial* = este valor irá forçar a reinicialização do valor padrão da propriedade.

*inherit* = este valor irá atribuir a herança explícita para a propriedade aplicada. Assim, caso o elemento pai possua a mesma propriedade com algum valor, este será replicado para o elemento em questão.

*unset* = este valor desabilita a propriedade em questão.

*revert* = este valor tem a mesma função do valor initial, porém é aplicado apenas ao navegador Safari.

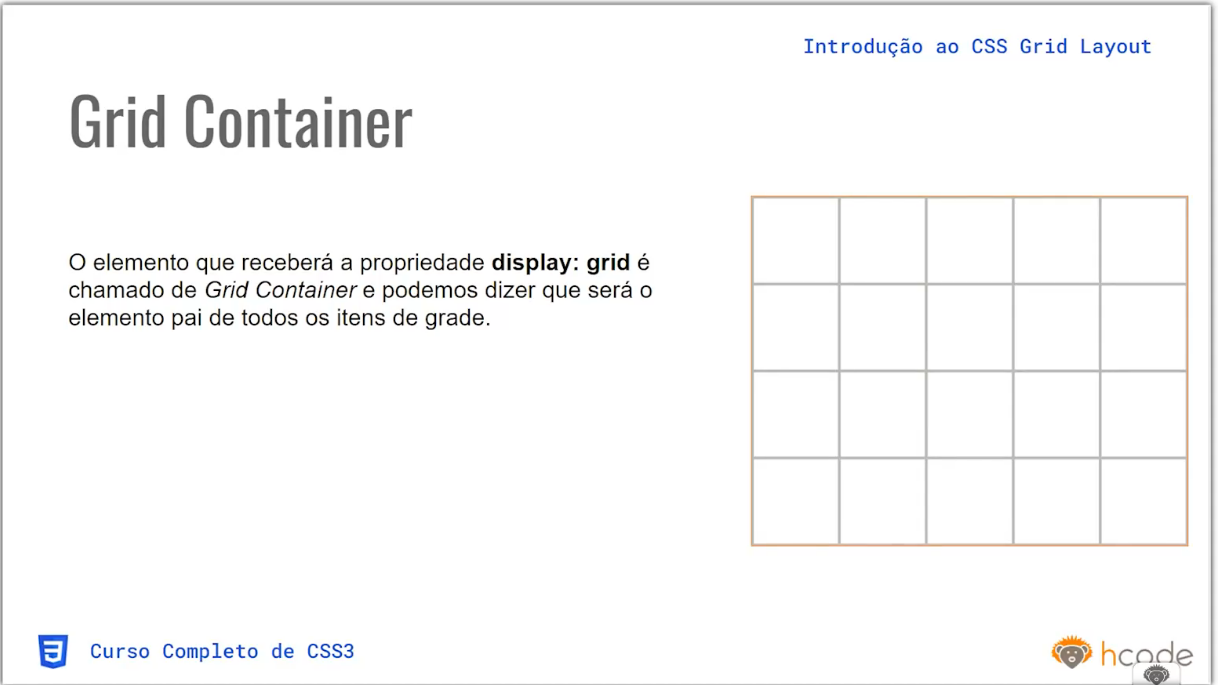
**GRID LAYOUT**

A Especificação CSS Grid Layout permite a criação de layout bidimensional, trabalhando com linhas e colunas, permitindo a criação de inúmeras variações de tipos de layout.

Organizando elementos em linhas e colunas, com Grid Layout é possível criar layout semelhante ao que era feito com tabelas, porém diferentemente das tabelas, os elementos filhos podem se sobrepor a outros conteúdos, dando aos elementos HTML o poder de ser alocado em qualquer parte do layout.

Assim como o Módulo Flexbox, o Módulo Grid Layout também irá trabalhar com um Container que funcionará como um wrapper ou embrulho dos elementos internos denominados grid items.

O Grid Layout está estruturado da seguinte forma:













**COMBINADOR DE SELETORES**

Nós podemos combinar seletores das mais variadas formas, como vemos a seguir:

COMBINADOR DESCENDENTE

form input {

background-color: #FFF;

}

Estou selecionando todas as tags <input> descendentes (filho, neto, bisneto, etc) de um formulário <form>.

COMBINADOR FILHO DIRETO

form > input {

background-color: #FFF;

}

Estou selecionando todas as tags <input> que são filhas diretas de uma tag <form> (vem imediatamente após a tag ou como filho dela).

COMBINADOR IRMÃO ADJACENTE

form **+** div {

background-color: #FFF;

}

Estou selecionando todas as <div> que possuam como irmão mais próximo a tag <form>.

COMBINADOR IRMÃO GERAL

form **~** div {

background-color: #FFF;

}

Estou selecionando todas as <div> que possuem como irmão a tag <form>.

**15/12 – PROPRIEDADES (PARTE 1)**

Propriedades são todas as informações que utilizamos para formatar um conteúdo.

Exemplo: background-color (formatar cor de fundo), width (formatar largura), etc.

A seguir, as propriedades do CSS3:

**height:** define a altura dos elementos, o que não inclui margens, bordas e espaçamentos. Permite herança explicita: uma vez definida a altura do elemento pai, é possível herdar para o elemento filho.

**obs.:** As propriedades **max-height** e **min-height** quando presentes, sobrescrevem a propriedade **height.**

Valores possíveis: auto | length | initial | inherit

Valor padrão: auto

Valor herdado: Não

**max-height:** define a altura máxima dos elementos. Se o conteúdo for maior que a altura máxima, ele irá gerar um overflow.

Valores possíveis: none | length | max-content | min-content | fit-content (length) | unset | initial | inherit

Valor padrão: none

Valor herdado: Não

**min-height:** define a altura mínima de um elemento. Se o conteúdo for menor que a altura mínima, a altura mínima será aplicada. Se o conteúdo for maior que a altura mínima, a propriedade será ignorada.

Valores possíveis: none | length | max-content | min-content | fit-content (length) | unset | initial | inherit

Valor padrão: auto

Valor herdado: Não

**width:** define a largura dos elementos, o que não inclui margens, bordas e espaçamentos.

**obs.:** As propriedades **max-width** e **min-width** quando presentes, sobrescrevem a propriedade **width**.

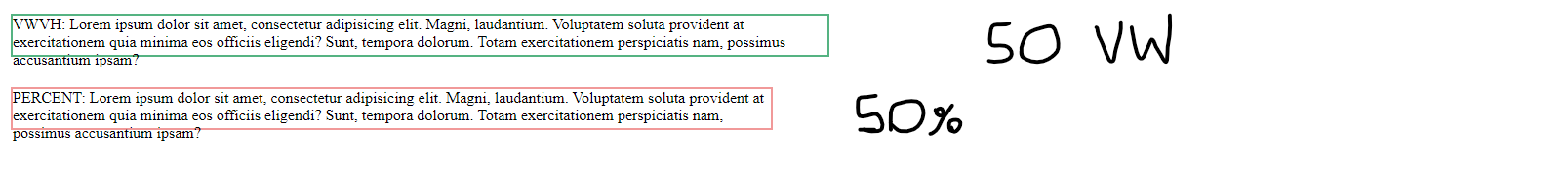
Valores possíveis: auto | length | max-content | min-content | fit-content (length) | unset | initial | inherit | revert

Valor padrão: auto

Valor herdado: não

Observação-1:Muitos falam que a width: auto é a mesma coisa que width: 100%, mas não é. A width: auto ocupa todo o viewport porque está com display:block. Quando mudamos o display para unset, a width: auto não ocupa mais todo o viewport, ou seja, o valor auto não equivale a 100%.

Observação-2:a diferença entre medir um elemento com vw e % é que o vw ignora possíveis margens que a página tenha. Já a porcentagem leva em conta essas margens. É o que pode se observar nessa imagem.



Observação-3:fit-content ajusta a largura de acordo com o conteúdo, max-content ajusta a largura como sendo a máxima possível, e min-content ajusta a largura como sendo a mínima possível.

**max-width:** define a largura máxima do elemento. Se o conteúdo for maior que a largura máxima, ele mudará automaticamente a altura do elemento. Se o conteúdo for menor que a largura máxima, a propriedade não terá efeito.

Valores possíveis: none | length | max-content | min-content | fit-content (length) | unset | initial | inherit

Valor padrão: none

Valor herdado: não

**min-width:** define a largura minima dos elementos, o que não inclui margens, bordas e espaçamentos.

Valores possíveis: none | length | max-content | min-content | fit-content (length) | unset | initial | inherit

Valor padrão: auto

Valor herdado: não

**16/12 – PROPRIEDADES (PARTE 2)**

**margin-top:** define o espaço acima do elemento, fora de qualquer borda definida.

Valores possíveis: length | auto | inherit | initial

Valor padrão: 0

Valor herdado: não

**margin-right:** define o espaço do lado direito do elemento, fora de qualquer borda definida.

Valores possíveis: length | auto | inherit | initial

Valor padrão: 0

Valor herdado: não

**margin-bottom:** define o espaço inferior do elemento, fora de qualquer borda definida.

Valores possíveis: length | auto | inherit | initial

Valor padrão: 0

Valor herdado: não

**margin-left:** define o espaço do lado esquerdo do elemento, fora de qualquer borda definida.

Valores possíveis: length | auto | inherit | initial

Valor padrão: 0

Valor herdado: não

**margin (espaçamento EXTERNO):** define o espaço em torno do elemento, fora de qualquer borda definida. Esta propriedade é uma forma abreviada das propriedades:

* margin-top
* margin-right
* margin-bottom
* margin-left

Valores possíveis: length | auto | inherit | initial

Valor padrão: 0

Valor herdado: não

**padding-top:** define a distância entre a borda do topo e o conteúdo do elemento.

Valores possíveis: length | inherit | initial

Valor padrão: 0

Valor herdado: não

**padding-right:** define a distância entre a borda do lado direito e o conteúdo do elemento.

Valores possíveis: length | inherit | initial

Valor padrão: 0

Valor herdado: não

**padding-bottom:** define a distância entre a borda de baixo e o conteúdo do elemento.

Valores possíveis: length | inherit | initial

Valor padrão: 0

Valor herdado: não

**padding-left:** define a distância entre a borda do lado esquerdo e o conteúdo do elemento.

Valores possíveis: length | inherit | initial

Valor padrão: 0

Valor herdado: não

**padding:** define a distância da borda de um elemento até seu conteúdo. Esta propriedade é a abreviação das propriedades

* padding-top
* padding-right
* padding-bottom
* padding-left

O padding criará espaço extra DENTRO de um elemento, enquanto a margin cria espaço extra AO REDOR de um elemento.

Valores possíveis: length | inherit | initial

Valor padrão: 0

Valor herdado: não

**border-top-color:** define a cor da borda superior de um elemento.

Valores possíveis: color | transparent | initial | inherit

Valor padrão: A cor atual do texto do elemento

Valor herdado: não

**border-right-color:** define a cor da borda do lado direito de um elemento.

Valores possíveis: color | transparent | initial | inherit

Valor padrão: A cor atual do texto do elemento

Valor herdado: não

**border-bottom-color:** define a cor da borda inferior de um elemento.

Valores possíveis: color | transparent | initial | inherit

Valor padrão: A cor atual do texto do elemento

Valor herdado: não

**border-left-color:** define a cor da borda do lado esquerdo de um elemento.

Valores possíveis: color | transparent | initial | inherit

Valor padrão: A cor atual do texto do elemento

Valor herdado: não

**border-color:** define a cor da borda de um elemento. Esta propriedade é uma forma abreviada das propriedades:

* border-top-color
* border-right-color
* border-bottom-color
* border-left-color

Valores possíveis: color | transparent | initial | inherit

Valor padrão: A cor atual do texto do elemento

Valor herdado: não

**19/12 – PROPRIEDADES (PARTE 3)**

**border-top-style:** define o estilo da linha da borda superior de um elemento.

Valores possíveis: none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | initial | inherit | unset

Valor padrão: none

Valor herdado: não

**border-right-style:** define o estilo da linha da borda do lado direito de um elemento.

Valores possíveis: none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | initial | inherit | unset

Valor padrão: none

Valor herdado: não

**border-bottom-style:** define o estilo da linha da borda inferior de um elemento.

Valores possíveis: none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | initial | inherit | unset

Valor padrão: none

Valor herdado: não

**border-left-style:** define o estilo da linha da borda do lado esquerdo de um elemento.

Valores possíveis: none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | initial | inherit | unset

Valor padrão: none

Valor herdado: não

**border-style:** define o estilo da linha da borda nos quatro lados do elemento. Esta propriedade é uma forma abreviada das propriedades border-top-style, border-right-style, border-bottom-style e border-left-style.

Valores possíveis: none (nenhum) | hidden (oculto) | dotted (pontilhado) | dashed (tracejado) | solid (linha sólida) | double (linha dupla) | groove (linha com duas tonalidades) | ridge (linha com duas tonalidades) | inset (chanfrado) | outset (chanfrado) | initial (valor inicial) | inherit (herda do elemento pai) | unset (desligar propriedade)

Valor padrão: none

Valor herdado: não

**border-top-width:** define a largura da borda superior de um elemento.

Valores possíveis: medium | thin | thick | length | initial | inherit

Valor padrão: medium

Valor herdado: não

**border-right-width:** define a largura da borda do lado direito de um elemento.

Valores possíveis: medium | thin | thick | length | initial | inherit

Valor padrão: medium

Valor herdado: não

**border-bottom-width:** define a largura da borda inferior de um elemento.

Valores possíveis: medium | thin | thick | length | initial | inherit

Valor padrão: medium

Valor herdado: não

**border-left-width:** define a largura da borda do lado esquerdo de um elemento.

Valores possíveis: medium | thin | thick | length | initial | inherit

Valor padrão: medium

Valor herdado: não

**border-width:** define a largura da borda do elemento. Esta propriedade é uma forma abreviada das propriedades:

* border-top-width
* border-right-width
* border-bottom-width
* border-left-width

Valores possíveis: medium (médio) | thin (mais fina) | thick (mais espessa) | length (unidades de medida) | initial (valor inicial) | inherit (herda do elemento pai)

Valor padrão: medium

Valor herdado: não

**border-top-left-radius:** permite arrendondar o canto superior esquerdo da borda. Se for definido uma segunda unidade de medida, o primeiro será para a borda superior e o segundo para a borda esquerda. Caso não informe o segundo valor, o primeiro será copiado.

Valores possíveis: length [length] | initial | inherit

Valor padrão: 0

Valor herdado: não

**border-top-right-radius:** permite arrendondar o canto superior direito da borda. Se for definido uma segunda unidade de medida, o primeiro será para a borda superior e o segundo para a borda direita. Caso não informe o segundo valor, o primeiro será copiado.

Valores possíveis: length [length] | initial | inherit

Valor padrão: 0

Valor herdado: não

**border-bottom-right-radius:** permite arrendondar o canto inferior direito da borda. Se for definido uma segunda unidade de medida, o primeiro será para a borda inferior e o segundo para a borda direita. Caso não informe o segundo valor, o primeiro será copiado.

Valores possíveis: length [length] | initial | inherit

Valor padrão: 0

Valor herdado: não

**border-bottom-left-radius:** permite arrendondar o canto inferior esquerdo da borda. Se for definido uma segunda unidade de medida, o primeiro será para a borda inferior e o segundo para a borda esquerda. Caso não informe o segundo valor, o primeiro será copiado.

Valores possíveis: length [length] | initial | inherit

Valor padrão: 0

Valor herdado: não

**border-radius:** define o arredondamento das bordas do elemento. Esta propriedade é uma forma abreviada das propriedades:

* border-top-left-radius
* border-top-right-radius
* border-bottom-right-radius
* border-bottom-left-radius

Valores possíveis: length [length] [length] [length] | initial | inherit

Valor padrão: 0

Valor herdado: não

**20/12 – PROPRIEDADES (PARTE 4)**

**border-image-source:** define a fonte ou caminho da imagem que deseja usar como borda em torno do elemento.

Valores possíveis: none | image | initial | inherit

Valor padrão: none

Valor herdado: não

**border-image-width:** define a largura da imagem da borda.

Valores possíveis: number | % | auto | initial | inherit

Valor padrão: 1

Valor herdado: não

**border-image-slice:** define como a imagem da borda será dividida.

Valores possíveis: number | % | fill | initial | inherit

Valor padrão: 100%

Valor herdado: não

**border-image-outset:** define a distância entre um elemento e a imagem usada como borda neste mesmo elemento.

Valores possíveis: number | initial | inherit

Valor padrão: 0

Valor herdado: não

**border-image-repeat:** define se a imagem usada como borda em um elemento será repetida para contornar todo o elemento uniformemente, ou então se deve ser esticada para que possa circundá-lo.

Valores possíveis: stretch | repeat | round | space | initial | inherit

Valor padrão: stretch

Valor herdado: não

**border-image:** define uma imagem como a borda ao redor de um elemento. Esta propriedade é uma forma abreviada das propriedades:

* border-image-source (origem da imagem)
* border-image-width (dimensão da imagem)
* border-image-slice (o corte da imagem, onde será feito)
* border-image-outset(a distância entre o elemento e a imagem usada como borda)
* border-image-repeat (qual é a regra para repetir essa imagem)

Valores possíveis: source slice / width / outset repeat | initial | inherit

Valor padrão: none 100% 1 0 stretch

Valor herdado: não

**border-collapse:** define se as bordas de uma tabela devem ser mescladas em uma única, ou se devem se manter separadas.

Valores possíveis: collapse | separate | unset | initial | inherit

Valor padrão: separate

Valor herdado: não

**border-spacing:** define a distância que haverá entre as bordas (células) de uma tabela.

Valores possíveis: length | initial | inherit | unset

Valor padrão: 2px

Valor herdado: sim

**border-top:** define o estilo da borda superior de um elemento. Esta propriedade espera definir a largura, estilo e cor da borda. É uma forma abreviada das propriedades:

* border-top-color
* border-top-style
* border-top-width

Valores possíveis: border-top-width border-top-style border-top-color

Valor padrão: medium none color

Valor herdado: não

**border-right:** define o estilo da borda do lado direito de um elemento. Esta propriedade espera definir a largura, estilo e cor da borda. É uma forma abreviada das propriedades:

* border-right-color
* border-right-style
* border-right-width

Valores possíveis: border-right-width border-right-style border-right-color

Valor padrão: medium none color

Valor herdado: não

**border-bottom:** define o estilo da borda inferior de um elemento. Esta propriedade espera definir a largura, estilo e cor da borda. É uma forma abreviada das propriedades:

* border-bottom-color
* border-bottom-style
* border-bottom-width

Valores possíveis: border-bottom-width border-bottom-style border-bottom-color

Valor padrão: medium none color

Valor herdado: não

**border-left:** define o estilo da borda do lado esquerdo de um elemento. Esta propriedade espera definir a largura, estilo e cor da borda. É uma forma abreviada das propriedades:

* border-left-color
* border-left-style
* border-left-width

Valores possíveis: border-left-width border-left-style border-left-color

Valor padrão: medium none color

Valor herdado: não

**border:** define o estilo da borda de um elemento. Esta propriedade espera definir a largura, estilo e cor da borda. É uma forma abreviada das propriedades:

* border-width
* border-style
* border-color

Valores possíveis: border-width border-style border-color | initial | inherit | unset

Valor padrão: medium none color

Valor herdado: não

**22/12 – PROPRIEDADES (PARTE 5)**

**box-sizing:** define como a largura e altura de um elemento são calculados. Se irão incluir o preenchimento e a borda ou não.

Valores possíveis: content-box | border-box | initial | inherit | unset

Valor padrão: content-box

Valor herdado: não

**overflow-x:** especifica se o conteúdo deve ser cortado, se deve adicionar uma barra de rolagem ou se deve exibir o conteúdo do estouro de um elemento de nível de bloco quando este estoura as bordas laterais (eixo X).

Valores possíveis: visible | hidden | scroll | auto | initial | inherit | unset

Valor inicial: visible

Valor herdado: não

**overflow-y:** especifica se o conteúdo deve ser cortado, se deve adicionar uma barra de rolagem ou se deve exibir o conteúdo do estouro de um elemento de nível de bloco quando este estoura nas bordas do topo e rodapé (eixo Y).

Valores possíveis: visible | hidden | scroll | auto | initial | inherit | unset

Valor inicial: visible

Valor herdado: não

**overflow:** define o comportamento do conteúdo quando este estoura os limites definidos de altura e largura da caixa. Se não for definido o segundo valor, o primeiro será copiado para o segundo eixo. Esta propriedade é uma forma abreviada das propriedades

* overflow-x
* overflow-y

Valores possíveis: visible | hidden | scroll | auto | initial | inherit | unset

Valor inicial: visible

Valor herdado: não

**overflow-wrap:** define se uma palavra deve ser quebrada quando atingir o fim de uma linha.

Valores possíveis: normal | break-word | anywhere | initial | inherit | unset

Valor inicial: normal

Valor herdado: sim

**overflow-block:** irá determinar o que deve acontecer com um conteúdo de um elemento de bloco quando este estoura os limites do bloco. Pode ser oculto, gerar uma barra de rolagem, ou não ter nenhuma alteração. A propriedade overflow-block direciona para o overflow-y ou overflow-x dependendo do modo de escrito do documento.

Disponível apenas no Firefox.

Valores possíveis: visible | hidden | scroll | auto | initial | inherit | unset

Valor inicial: auto

Valor herdado: não

**overflow-inline:** irá especificar o que deve ser exibido quando o conteúdo estoura os limites de um elemento de linha onde foi definida e pode ser usada na condição de uma media query. Esta propriedade irá direcionar para as propriedades overflow-y ou overflow-x dependendo do modo de escrita do documento.

Disponível apenas no Firefox.

Valores possíveis: visible | hidden | scroll | auto | initial | inherit | unset

Valor inicial: auto

Valor herdado: não

**resize:** determina se e como um elemento HTML que possua essa propriedade poderá ser redimensionado pelo usuário.

Valores possíveis: none | both | horizontal | vertical | block | inline | initial | inherit | unset

Valor inicial: none

Valor herdado: não

**z-index:** define a ordem de pilha de um elemento. O elemento com maior ordem de pilha está sempre na frente de um elemento com uma ordem de pilha menor.

Valores possíveis: auto | number | initial | inherit | unset

Valor inicial: auto

Valor herdado: não

**outline-style:** define o estilo da linha de contorno do elemento.

Valores possíveis: auto | dashed | dotted | double | groove | hidden | initial | inherit | inset | none | outset | revert | ridge | solid | unset

Valor inicial: none

Valor herdado: não

**outline-color:** define a cor da linha de contorno do elemento.

Valores possíveis: invert | color | initial | inherit | unset

Valor padrão: invert | currentColor

Valor herdado: não

**outline-width:** define a largura da linha de contorno do elemento.

Valores possíveis: medium | thin | thick | length | initial | inherit | unset

Valor padrão: medium

Valor herdado: não

**outline-offset:** define a distância entre a linha de contorno e a borda do elemento.

Valores possíveis: length | initial | inherit | unset

Valor padrão: 0

Valor herdado: não

**outline:** define uma linha ao redor do elemento, fora das bordas, com o objetivo de destacar o elemento. Esta propriedade é uma forma abreviada das propriedades:

* outline-width
* outline-style
* outline-color

Valores possíveis: outline-width outline-style outline-color| initial | inherit | unset

Valor padrão: medium auto invert

Valor herdado: não

**23/12 – PROPRIEDADES (PARTE 6)**

**display:** define o comportamento de exibição de um elemento.

Valores possíveis: value\* | initial | inherit | unset

Valor padrão: depende do elemento

Valor herdado: não

\*Values:

* inline
* block
* inline-block
* contents
* flex
* inline-flex
* grid
* inline-grid
* list-item
* run-in
* none
* table
* inline-table
* table-caption
* table-column-group
* table-header-group
* table-footer-group
* table-cell
* table-column
* table-row
* initial
* inherit

**flex-direction:** determina a direção de como os elementos dentro de uma flexbox são organizados (em linha ou coluna). Essa propriedade vai no contâiner, que recebe a propriedade display flex.

Valores possíveis: row | row-reverse | column | column-reverse | initial | inherit | unset

Valor padrão: row

Valor herdado: não

**flex-wrap:** define se os itens flexíveis devem quebrar ou não.

Valores possíveis: nowrap | wrap | wrap-reverse | initial | inherit | unset

Valor padrão: nowrap

Valor herdado: não

**flex-flow:** define o fluxo dos itens flexíveis. Esta propriedade é uma forma abreviada das propriedades flex-direction e flex-wrap.

Valores possíveis: flex-direction flex-wrap | initial | inherit | unset

Valor padrão: row nowrap

Valor herdado: não

**justify-content:** define o alinhamento dos itens flexíveis no eixo principal (eixo X).

Valores possíveis: flex-start | flex-end | center | space-between | space-around | space-evenly | initial | inherit | unset

Valor padrão: flex-start

Valor herdado: não

**align-items:** define o alinhamento dos itens flexíveis no eixo secundário (eixo Y). É o equivalente a definir a propriedade align-self em cada um dos elementos filhos. Um elemento filho pode possuir a propriedade align-self para substituir o alinhamento definido no container com esta propriedade.

Valores possíveis: normal | stretch | center | start | end | flex-start | flex-end | self-start | self-end | baseline | first baseline | last baseline | safe center | unsafe center | initial | inherit | unset

Valor padrão: normal

Valor herdado: não

**25/12 – PROPRIEDADES (PARTE 7)**

**align-content:** define como cada linha será alinhada dentro de um contêiner flex após ter sido ativada a propriedade **flex-wrap**. É semelhante ao align-items.

Valores possíveis: normal | stretch | center | start | end | flex-start | flex-end | baseline | first baseline | last baseline | space-between | space-around | space-evenly | safe center | unsafe center | initial | inherit | unset

Valor padrão: stretch

Valor herdado: não

**place-content:** é uma abreviação das propriedades **align-content** e **justify-content**.

Se o segundo valor não for informado, o primeiro valor será usado para ambos.

Valores possíveis: normal | stretch | center | start | end | flex-start | flex-end | baseline | first baseline | last baseline | space-between | space-around | space-evenly | safe center | unsafe center | initial | inherit | unset

Valor padrão: normal

Valor herdado: não

**justify-items:** é usada em containers grid e alinham os itens de suas áreas de grade no eixo embutido. É o equivalente a definir a propriedade **justify-self** em cada um dos elementos filhos.

Valores possíveis: legacy | auto | normal | stretch | center | start | end | flex-start | flex-end | self-start | self-end | left | right | baseline | first baseline | last baseline | safe center | unsafe center | legacy right | legacy left | legacy center | initial | inherit | unset

Valor padrão: normal

Valor herdado: não

**place-items:** é uma abreviação das propriedades **align-items** e **justify-items**. Se o segundo valor não for informado, o primeiro valor será usado para ambos.

Valores possíveis: legacy | auto | normal | stretch | center | start | end | flex-start | flex-end | self-start | self-end | left | right | baseline | first baseline | last baseline | safe center | unsafe center | legacy right | legacy left | legacy center | initial | inherit | unset

Valor padrão: auto

Valor herdado: não

**justify-self:** define o alinhamento do item (individual) no eixo primário dentro de um contêiner flexível.

Valores possíveis: auto | normal | first | last | start | end | stretch | center | flex-start | flex-end | self-start | self-end | left | right | baseline | first baseline | last baseline | initial | inherit | unset

Valor padrão: auto

Valor herdado: não

**flex-basis:** define o comprimento inicial da base (largura) um item flexível.

Valores possíveis: length | auto | fill | max-content | min-content | fit-content | content | initial | inherit | unset

Valor padrão: auto

Valor herdado: não

**flex-grow:** define quanto o item crescerá em relação ao restante dos itens flexíveis dentro do mesmo contêiner flex.

Valores possíveis: number | initial | inherit | unset

Valor padrão: 0

Valor herdado: não

**flex-shrink:** define como o item será reduzido em relação ao restante dos itens flexíveis do mesmo contêiner flex.

Valores possíveis: number | initial | inherit | unset

Valor padrão: 1

Valor herdado: não

**flex:** define o comprimento flexível dos itens flexíveis. Esta propriedade é uma forma abreviada das propriedades:

* flex-grow
* flex-shrink
* flex-basis

Valores possíveis: flex-grow flex-shrink flex-basis | auto | initial | inherit | unset

Valor padrão: 0 1 auto

Valor herdado: não

**order:** define a ordem de um item flexível em relação aos outros itens flexíveis dentro do mesmo contêiner.

Valores possíveis: number | initial | inherit | unset

Valor padrão: 0

Valor herdado: não

**align-self:** define o alinhamento de um item (individual) que está no eixo secundário dentro do contêiner flexível.

Valores possíveis: auto | stretch | center | flex-start | flex-end | baseline | initial | inherit | unset

Valor padrão: auto

Valor herdado: não

**place-self:** é uma abreviação das propriedades **align-self** e **justify-self.** Se o segundo valor não for informado, o primeiro valor será usado para ambos.

Valores possíveis: auto | stretch | center | flex-start | flex-end | baseline | initial | inherit | unset

Valor padrão: auto

Valor herdado: não

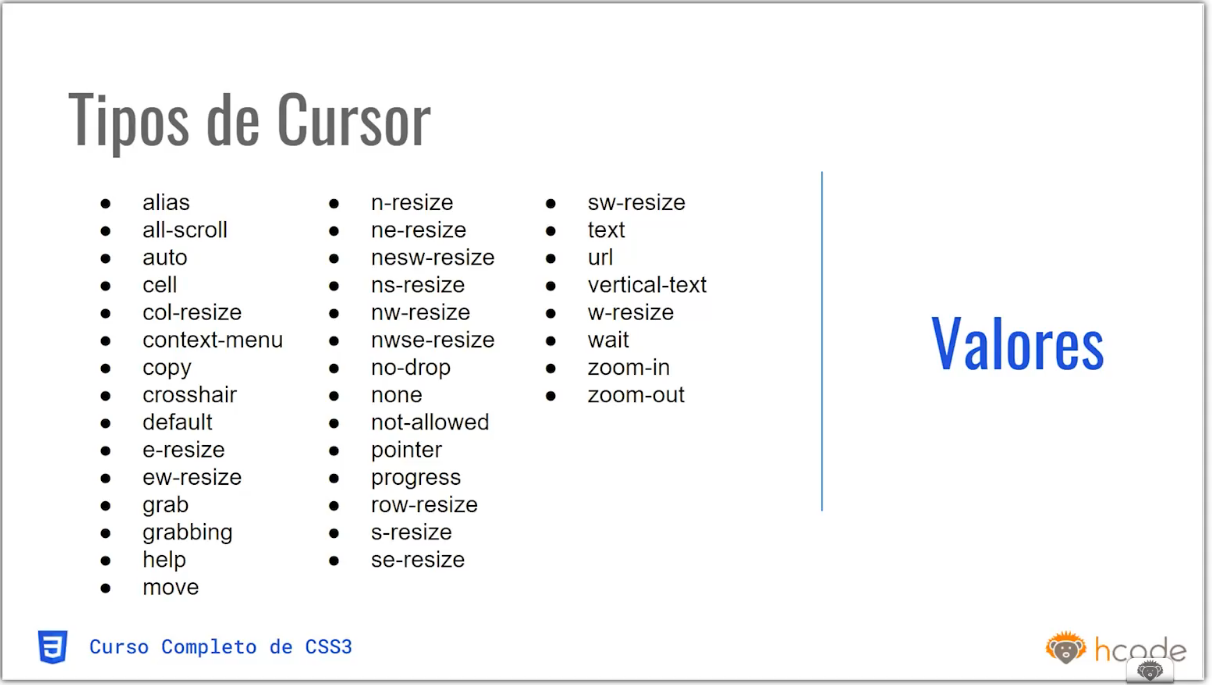
**26/12 – PROPRIEDADES (PARTE 8)**

**cursor:** define o cursor do mouse que será exibido quando o mouse passar por determinado elemento.

Valores possíveis: auto | *cursor-type\** | initial | inherit | unset

Valor padrão: auto

Valor herdado: sim



**background-color:** define a cor do fundo de um elemento. O tamanho do fundo de um elemento inclui seu tamanho total, preenchimento e borda (mas não a área da margem).

Valores possíveis: color | transparent | initial | inherit | unset

Valor padrão: transparent

Valor herdado: não

**background-image:** define uma imagem como plano de fundo de um elemento.

Valores possíveis: url | none | initial | inherit | unset

Valor padrão: none

Valor herdado: não

**background-size:** define o tamanho de uma imagem de plano de fundo.

Valores possíveis: auto | length | cover | contain | initial | inherit | unset

Valor padrão: auto

Valor herdado: não

**background-position:** define a posição inicial de uma imagem de plano de fundo.

Valores possíveis: top | right | bottom | left | center | length | x% y% | initial | inherit | unset

Valor padrão: 0% 0%

Valor herdado: não

**background-repeat:** define como a imagem de fundo será repetida.

Valores possíveis: repeat | repeat-x | repeat-y | no-repeat | initial | inherit | unset

Valor padrão: repeat

Valor herdado: não

**background-attachment:** define como uma imagem de plano de fundo irá se comportar durante o scroll da página.

Valores possíveis: scroll | fixed | local | initial | inherit | unset

Valor padrão: scroll

Valor herdado: não

**27/12 – SASS**

**DEFINIÇÃO**

O SASS é um pré-processador do CSS, ou seja, uma linguagem de extensão de folhas de estilo de nível profissional mais madura, estável e poderosa. O SASS é compatível com todas as versões do CSS.

O SASS tem sido apoiado ativamente por mais de 14 anos e tem sido escolhido pela indústria como a principal linguagem de extensão do CSS.

**PORQUE UTILIZAR SASS?**

Os pré-processadores CSS são linguagens de script que estendem os recursos padrão do CSS. Eles nos permitem usar lógica em nosso código CSS, como variáveis, aninhamento, herança, mixings, funções e operações matemáticas.

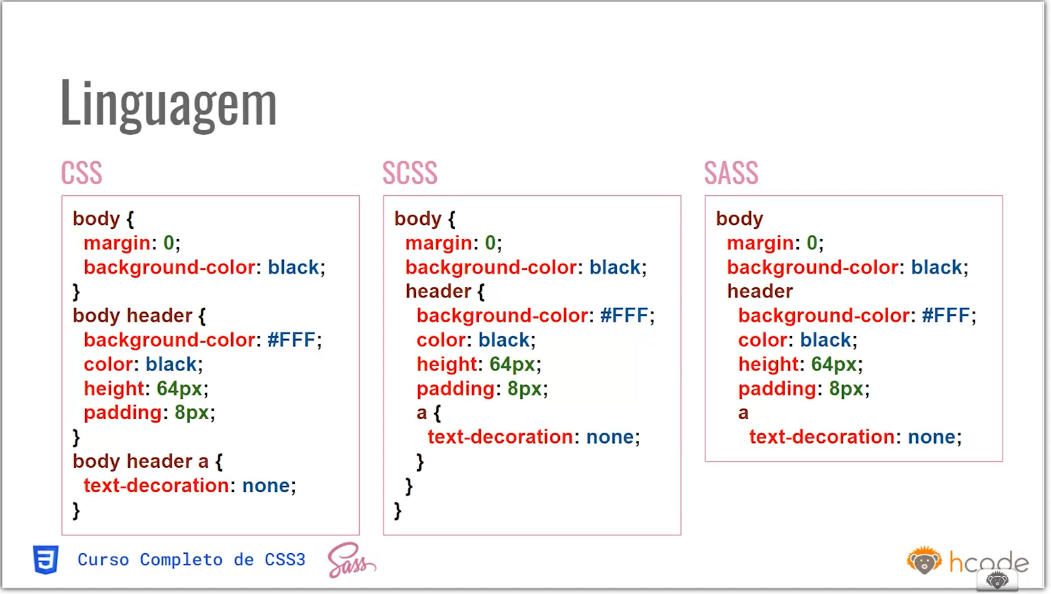
Os pré-processadores CSS facilitam a automatização de tarefas repetitivas, reduzem o número de erros e inchaço do código, criam fragmentos de código reutilizáveis e garantem a compatibilidade com versões anteriores.

**OUTROS PRÉ-PROCESSADORES CSS**

Cada pré-processador CSS tem sua própria sintaxe que é compilada em CSS regular para que os navegadores possam renderizá-la no lado do cliente.

Atualmente, os três pré-processadores CSS mais populares e estáveis são SASS, LESS, Stylus e Styled Components. No entanto, existem muitos menores também.

Todos os pré-processadores CSS fazem coisas semelhantes, mas de uma maneira diferente e com suas próprias sintaxes. Cada um deles possui alguns recursos avançados exclusivos e também o seu próprio ecossistema (ferramentas, frameworks, bibliotecas).

****

**PÓS-INSTALAÇÃO**

Após a instalação do SASS, podemos usar o parâmetro **---watch,** que monitora um arquivo ou diretório. Esse parâmetro irá avisar o SASS sempre que um arquivo ou diretório monitorado for alterado, e então irá recompilar o arquivo CSS. Para ativar esse monitoramento, digitamos o seguinte comando no terminal:

npm --watch input.scss:output.css

É possível também monitorar um diretório completamente, e gerar um código correspondente em outro diretório. Basta utilizarmos o sinal de dois-pontos (:):

sass --watch src/sass:dist/assets/css

O SASS irá monitorar o diretório *src/sass* buscando por alterações, e irá compilar o CSS final para a pasta *dist/assets/css*.

**NESTING (ANINHAMENTO)**

O aninhamento (ou “nesting”, em inglês) é um recurso que tem como objetivo deixar a vida do desenvolvedor de estilos mais fácil. Ajudando a evitar a repetição de seletores, o nesting nos permite escrever um bloco de estilo dentro de outro bloco de estilo, deixando o CSS mais próximo da estrutura de árvore no HTML.

**PALAVRA DE CAUTELA**

Ao se acostumar com esse recurso, é comum que os desenvolvedores se sintam motivados a criar todos os estilos aninhados. Porém, isso pode se tornar um problema para visualização do código e também no tamanho final do seu arquivo CSS. Lembre-se que cada aninhamento irá gerar um seletor mais específico, que por sua vez produzirá um arquivo maior e que precisará de mais banda de conexão para ser baixado.

**LISTA DE SELETORES**

É possível também fazer o aninhamento de lista de seletores que são separados por vírgula. O SASS é inteligente para lidar com essa situação e irá criar uma lista de seletores no arquivo CSS final, gerando assim um resultado menor.

**COMBINADORES DE SELETOR**

É possível também aninhar seletores que usam combinadores como:

* descendente
* filho direto
* irmão geral
* irmão adjacente

**28/12 – VARIÁVEIS**

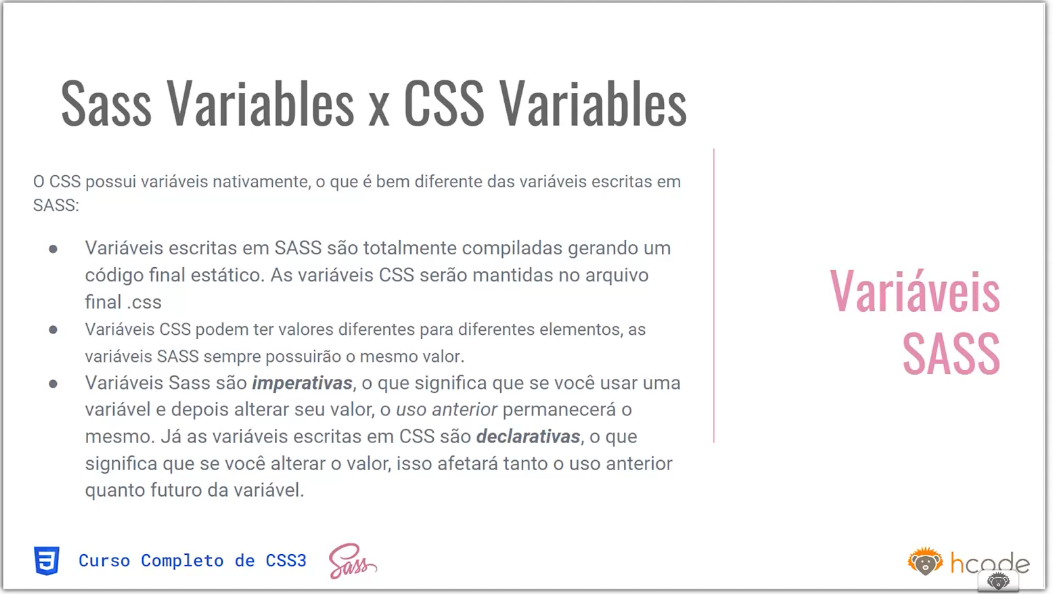
**O QUE SÃO?**

Elas são espaços na memória do computador que usamos temporariamente. Os dados encontrados nesses espaços podem mudar, ou variar. Sempre que for necessário utilizar um valor em vários lugares é possível utilizar variáveis.

As variáveis SASS são simples: você atribui um valor a um nome que começa com S e, em seguida, pode se referir a esse nome em vez do próprio valor.

Apesar de sua simplicidade, elas são uma das ferramentas mais utilizadas dentro do SASS. As variáveis tornam possível reduzir a repetição, fazer matemática complexa, configurar bibliotecas e muito mais.

**VARIÁVEIS DO SASS X VARIÁVEIS DO CSS**

****

**UTILIZANDO VARIÁVEIS**

SASS

$variavel: 10px;

.class {

value: $variavel

}

CSS

.class {

value: 10px;

}

DICA: Variáveis no SASS tratam hífen e underscore de maneira idêntica. Isso significa que $font-size e $font\_size referem-se a mesma variável.

**A FLAG !default**

Normalmente, quando você atribui um valor a uma variável, se essa variável já tiver um valor, s**eu valor antigo é sobrescrito**. Mas se você estiver escrevendo uma biblioteca SASS, convém permitir que seus usuários configurem as variáveis de sua biblioteca antes de usá-las para gerar CSS.

Para tornar isso possível, o SASS fornece o sinalizador **!default**. Isso atribui um valor a uma variável apenas se essa variável não for definida ou seu valor for nulo. Caso contrário, o valor existente será usado.

**COMO UTILIZAR**

Variáveis definidas com !default podem ser configuradas ao carregar um módulo com a regra @use. Bibliotecas SASS costumam usar variáveis !default para permitir que seus usuários configurem o CSS da biblioteca.

Sintaxe:

@use <url> with(<variable>; <value>, <variable>; <value>)



**VARIÁVEIS NATIVAS**

Variáveis nativas, também conhecidas como built-in, são variáveis que não podem ser alteradas, e atuam como constantes.

**@use** "sass:math" as math;

*// This assignment will fail.*

math.$pi: 0;

**ESCOPO DE VARIÁVEIS**

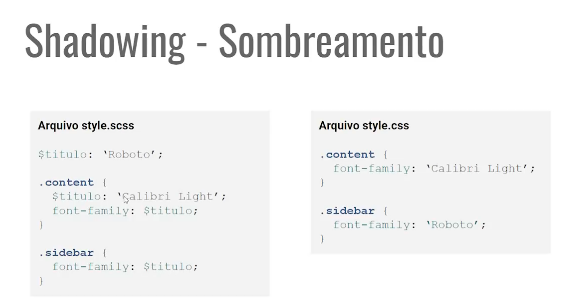
As variáveis declaradas no nível superior de uma folha de estilo são globais. Isso significa que elas podem ser acessadas em qualquer lugar em seu módulo após terem sido declarados. Mas isso não é verdade para todas as variáveis. Aquelas declaradas em blocos (chaves no SCSS ou código indentado no SASS) são geralmente locais e só podem ser acessados dentro do bloco em que foram declarados.



**SHADOWING (SOMBREAMENTO)**

Variáveis locais podem até ser declaradas com o mesmo nome de uma variável global. Se isso acontecer, na verdade existem duas variáveis diferentes com o mesmo nome: uma local e uma global.

Isso ajuda a garantir que um autor escrevendo uma variável local não altere acidentalmente o valor de uma variável global da qual ele nem mesmo tem conhecimento.



**30/12 – REGRAS DE ESTILO SASS**

As regras de estilo são impostas para mudar o comportamento de algo. São determinadas por meio da arroba (exemplo: @media). As regras de estilo do SASS são:

**@use** = carrega mixins, funções e variáveis de outras folhas de estilo SASS para a SUA folha de estilo.

**@forward** = carrega uma folha de estilo SASS e disponibiliza seus mixins, funções, e variáveis quando sua folha de estilo é carregada com a regra @use.

**@import** = uso semelhante ao da regra @use, porém está depreciado.

**@function** = define funções personalizadas em operações complexas que podem ser usadas em expressões SassScript.

**@mixin** – são pedaços de código de estilo que podem ser reutilizados.

**@include –** aplica os mixins em determinado seletor.

**@extend** = permite que os seletores herdem estilos uns dos outros. Se você deseja definir uma classe em sua folha de estilo com todas as regras de estilo de outra classe junto com suas próprias regras de estilo específicas, pode usar a regra @extend.

**@error** = imprime uma mensagem de erro e interrompe a compilação.

**@warn** = imprime um aviso sem parar totalmente a compilação.

**@debug** = imprime uma mensagem para fins de depuração. É útil para verificar o valor de variáveis ao escrever o código SASS para depurá-lo e ver como ele se comportará com diferentes valores de entrada.

**@at-root** = coloca estilos dentro dele na raiz do documento CSS. Não é usado com frequência em SASS, mas ajuda na implementação de aninhamento avançado com funções de seletor. Podemos usar esta regra com qualquer seletor (geralmente os seletores aninhados definidos dentro de chaves de outro seletor) para compilá-los como seletor raiz na folha de estilo e não como um seletor aninhado.

**CONTROLE DE FLUXO**

Regras de controle de fluxo como **@if**. **@each**. **@for** e **@while** controlam se ou quantas vezes os estilos são emitidos.

O SASS também tem algum comportamento especial para regras simples de CSS: elas podem conter interpolação e podem ser aninhadas em regras de estilo. Alguns deles, como @media e @supports, também permitem que script SASS seja usado diretamente na própria regra sem interpolação.

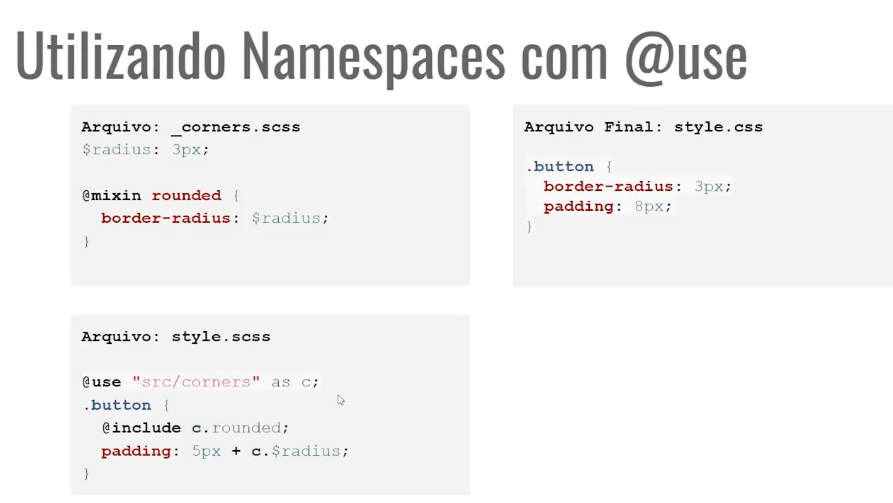
**31/12 - @use**

A regra @use carrega mixins, funções e variáveis de outras folhas de estilo SASS para a sua folha de estilo. Em outras palavras, a regra @use nos permite dividir nossa folha de estilo em seções menores e mais práticas e colocá-las dentro de outras folhas de estilo. **Folhas de estilo carregadas por @use são chamadas de “módulos”.** O SASS também fornece módulos integrados cheios de funções úteis.

A regra @use mais simples é escrita @use “<url>”, que carrega o módulo no URL fornecido. Todos os estilos carregados dessa forma serão incluídos exatamente uma vez na saída CSS compilada, não importa quantas vezes esses estilos sejam carregados.



É importante usar nomes curtos para módulos com @use, para facilitar. Veja como podemos fazer isso utilizando Namespaces.



Além disso, ao indicar o arquivo desejado com @use, não há a necessidade de digitar a extensão do arquivo. O SASS consegue achar o arquivo em questão de forma fácil e intuitiva. Se você digitar @use “colors”, irá carregar automaticamente colors.scss, colors.sass ou colors.css.

Outra coisa interessante que existe no SASS são os **Load Paths** (caminhos de carregamento), que você declara para que o SASS examine esse caminho para localizar o módulo. Por exemplo, se você passar *node\_modules/susy/sass* como um caminho de carregamento, você pode usar *@use “susy”* para carregar *node\_modules/susy/sass/susy.scss.* Porém, os módulos sempre serão carregados em relação **ao arquivo atual** primeiro, em caso de conflitos.

Existem também os **Partials**, arquivos SASS que devem ser carregados apenas como módulos, não compilados por conta própria nem gerando um arquivo final. Eles começam com \_ .

**@forward**

A regra @forward carrega uma folha de estilo SASS e disponibiliza seus mixins, funções e variáveis quando sua folha de estilo é carregada com a regra @use. Ele torna possível organizar bibliotecas SASS em muitos arquivos, enquanto permite que seus usuários carreguem um único arquivo de entrada.

A regra é escrita em @forward “<url>”. Ele carrega o módulo na URL fornecida exatamente como @use, **mas torna os membros públicos do módulo carregado disponíveis para os usuários do seu módulo como se eles tivessem sido definidos diretamente no seu módulo.** No entanto, esses membros não estão disponíveis em seu módulo – se você quiser, precisará escrever uma regra @use também.

****

Exemplo: após criar dois módulos partials para cores e definição de cantos, crio um arquivo bootstrap usando @forward.

@forward “cores”;

@forward “cantos”;

É como se eu estivesse falando: eu não vou utilizar ele aqui, irei disponibilizar ele para quem incluir meu arquivo bootstrap.

**ADICIONANDO UM PREFIXO**

Como os membros do módulo geralmente são usados com um namespace, nomes curtos e simples são geralmente a opção mais legível. Mas esses nomes podem não fazer sentido fora do módulo em que estão definidos, então **@forward** tem a opção de adicionar um prefixo extra para todos os membros que ele encaminha.

Isso é escrito @forward “<url>” as “<prefixo>-\*” e adiciona o prefixo fornecido ao início de cada mixin, função e nome de variável encaminhado pelo módulo. Por exemplo, se o módulo define um membro chamado reset e é encaminhado como list-\*, as folhas de estilo seguintes irão se referir a ele como list-reset.

****

**@import**

Com a regra @import, o SASS extende os recursos da mesma regra do CSS com a capacidade de importar folhas de estilo SASS e CSS, fornecendo acesso á mixins, funções e variáveis e combinando CSS de várias folhas de estilo.

Ao contrário das importações CSS simples, que exigem que o navegador faça várias solicitações HTTP á medida que renderiza sua página, **as importações SASS são inteiramente tratadas durante a compilação.**

As importações SASS tem a mesma sintaxe das importações CSS, exceto que permitem que várias importações sejam separadas por virgulas, em vez de exigir que cada uma tenha seu próprio @import. Além disso, na sintaxe recuada, os URLs importados não precisam ter aspas.

Porém, a equipe SASS **desencoraja o uso contínuo** da regra @import. O SASS irá gradualmente eliminá-lo ao longo dos próximos anos e, eventualmente, removê-lo inteiramente da linguagem. Em vez disso, é preferível usar a regra @use.

**@function**

Como o próprio nome diz, a regra @function serve para trabalharmos com funções (trechos de código que executam alguma ação e retornam algum valor). Funções permitem que você defina operações complexas utilizando os valores suportados pelo SASS (number, string, colors, lista de valores, que você pode reutilizar em sua folha de estilo). Eles facilitam a abstração de fórmulas e comportamentos comuns de uma forma legível.

As funções são definidas usando a @função na regra, que é escrita @função <nome> (<argumentos...>) {...}

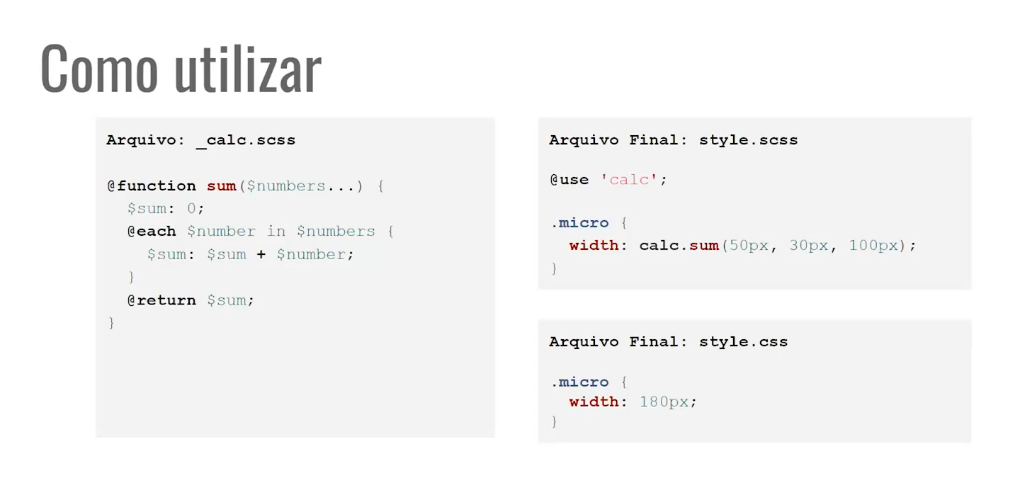
O nome de uma função pode ser qualquer identificador SASS. Ele só pode conter instruções universais, bem como a regra @return, que indica o valor a ser usado como resultado da chamada de função. As funções são chamadas usando a sintaxe de função CSS normal.

**CURIOSIDADE**

Nomes de funções, como todos os identificadores SASS, tratam de hifens e underlines como idênticos. Isso significa que scale-color e scale\_color referem-se á mesma função. Este é um resquício histórico desde os primeiros dias do SASS, quando ele só permitia underlines em nomes de identificadores.

Depois que o SASS adicionou suporte para hífens para corresponder á sintaxe do CSS, os dois se tornaram equivalentes para tornar a migração mais fácil.

**COMO UTILIZAR**



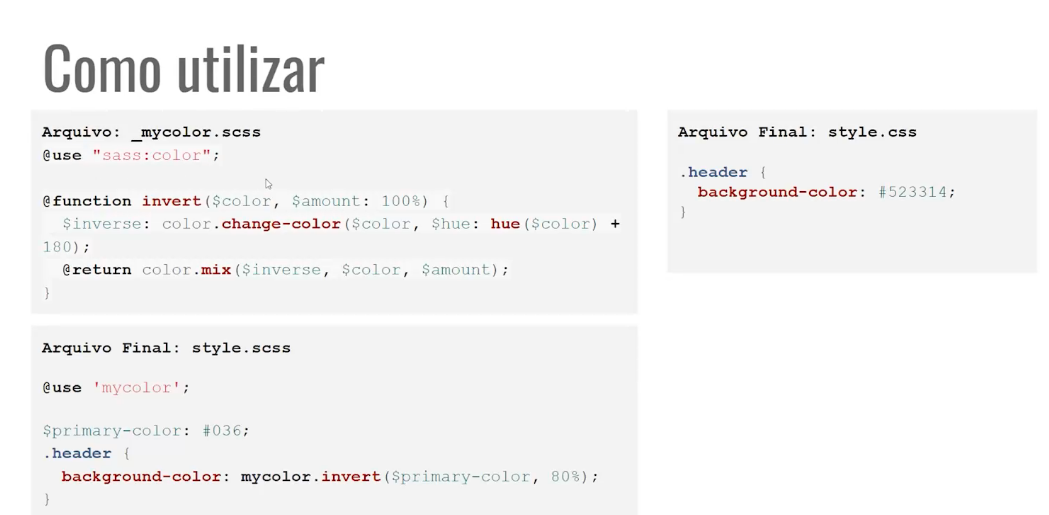
**REGRA @return**

A regra @return indica o valor a ser usado como resultado da chamada de uma função. Só é permitido dentro de um corpo de @function, e cada @function deve terminar com um @return.

Quando um @return é encontrado, ele termina imediatamente a função e retorna seu resultado.

Utilizar um @return no meio do código pode ser útil para lidar com casos extremos ou casos onde um algoritmo mais eficiente está disponível sem envolver a função inteira em um bloco @else.

**COMO UTILIZAR @RETURN**



**@mixin**

Mixins permitem que você defina estilos que podem ser reutilizados em sua folha de estilo. Eles tornam mais fácil evitar o uso de classes não semânticas como float-left e distribuir coleções de estilos em bibliotecas.

Mixins são definidos usando a regra @mixin, que é escrita @mixin <nome> {...} ou @mixin nome (<argumentos...>) {...}. O nome de um mixin pode ser qualquer identificador SASS e pode conter qualquer instrução diferente de instruções de nível superior.

**@include**

Mixins são incluídos no contexto atual usando a regra @include, que é escrita @include <nome> ou @include <nome> (<argumentos...>), com o nome do mixin sendo incluído.



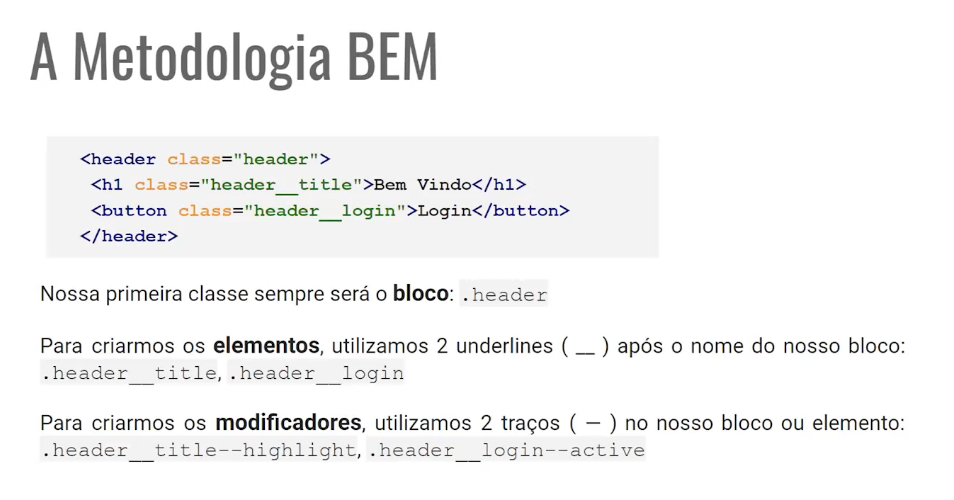
**@extend**

Muitas vezes ao projetarmos uma página, há casos em que uma deve ter todos os estilos de outra classe, bem como seus próprios estilos específicos.

Por exemplo, a metodologia BEM encoraja classes modificadoras que vão nos mesmos elementos que blocos ou classes de elementos. Mas isso pode criar um HTML desordenado, está sujeito a erros por esquecer de incluir as duas classes e pode trazer preocupações de estilo não semântico para sua marcação.

**A METODOLOGIA BEM**

Essa metodologia é aplicada na nomenclatura das classes CSS dos nossos elementos HTML. A sigla BEM significa Block Element Modifier (em português: Bloco Elemento Modificador) – esses 3 pilares são as bases dessa metodologia e também são as categorias nas quais vamos dividir nossos elementos.



Sendo aplicada essa metodologia BEM, observe o problema com que nos deparamos:

.error{

border: 1px #f00;

background-color: #fdd;

}

.error—serious{

border-width: 3px;

}

Utilizando a regra @extend, esse problema é resolvido de forma bem simples:



**DIFERENÇA ENTRE @extend E @mixins**

Ao contrário dos mixins, que copiam estilos para a regra de estilo atual, @extend **atualiza as regras de estilo que contém o seletor estendido para que também contenham o seletor estendido.** Ao estender os seletores, SASS faz uma unificação inteligente: ele nunca gera seletores como *#principal#rodape* que não podem corresponder a nenhum elemento.

Isso garante que os seletores complexos sejam intercalados para que funcionem independentemente da ordem em que os elementos HTML estão aninhados.

O @extend corta os seletores redundantes o máximo possível, ao mesmo tempo em que garante que a especificidade seja maior ou igual a do extensor.

O @extend sabe quando um seletor corresponde a tudo o que o outro faz e pode combiná-los.

O @extend lida de forma inteligente com combinadores, seletores universais e pseudo-classes que contém seletores.

**01/01 - @error**

Ao escrever mixins e funções que aceitam argumentos, geralmente você deseja garantir que esses argumentos tenham os tipos e formatos que sua API espera. Se não forem, o usuário precisa ser notificado e seu mixin / função precisa parar de funcionar.

O SASS torna isso fácil com a regra @error, que é escrita @error <expression>. Ele imprime o valor da expressão (geralmente uma string) junto com um rastreamento de pilha indicando como o mixin ou função atual foi chamado. Depois que o erro é impresso, o SASS para de compilar a folha de estilo e diz a qualquer sistema que o esteja executando que ocorreu um erro.



**@warn**

Ao escrever mixins e funções, você pode alertar os usuários a passar certos argumentos ou determinados valores. Eles podem estar passando argumentos legados que agora estão obsoletos ou podem estar chamando sua API de uma forma que não é muito ideal.

A regra @warn foi projetada exatamente para isso. É escrito @warn <expression> e imprime o valor da expressão (geralmente uma string) para o usuário, junto com um rastreamento de pilha indicando como o mixin ou função atual foi chamado. **Ao contrário da regra @error, no entanto, ela não interroga o SASS inteiramente.**



**@debug**

Ás vezes, é útil ver o valor de uma variável ou expressão enquanto você desenvolve sua folha de estilo. É para isso que serve a regra @debug: é escrita @debug <expression> e imprime o valor dessa expressão junto com o nome do arquivo e o numero da linha.



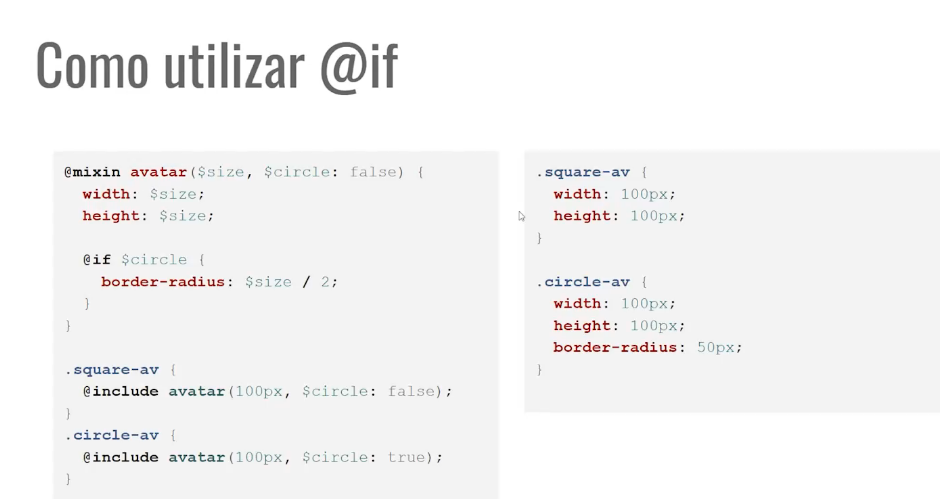
**@at-root**

A regra @at-root é geralmente escrita @at-root <seletor> {...} e faz com que tudo dentro dela seja **emitido na raiz do documento em vez de usar o aninhamento normal.** É mais frequentemente usado ao fazer aninhamento avançado com o seletor pai do script SASS e funções de seletores.

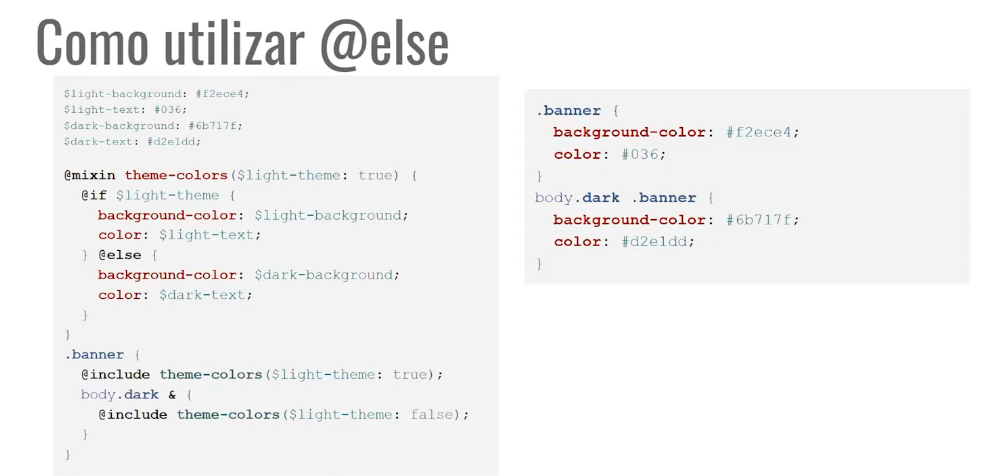


**Controle de fluxo @if e @else**

A regra @if é escrita como @if <expressão> {...} e controla se seu bloco é avaliado ou não (incluindo a emissão de qualquer estilo como CSS). A expressão geralmente retorna verdadeiro ou falo – se a expressão retornar verdadeiro, o bloco é avaliado, e se a expressão retornar falso, não é.



O @else funciona como negação. Se a condição de @if for falso e houver um @else, esse @else será aplicado. Observe:



**Controle de fluxo @for**

A regra @for, escrita:

**@for <variable> from <expression> to <expression> {…}**

ou

**@for <variable> from <expression> through <expression> {…}**

Conta a partir de um número (o resultado da primeira expressão) para outro (o resultado da segunda) e avalia um bloco para cada número intermediário. **Ele é usado para iterar (repetir) e compilar um bloco de estilo com um valor diferente a cada vez.**

Cada número ao longo do caminho é atribuído a um determinado nome de variável. Se **to** for usado, o número final é excluído; se **through** for usado, ele é incluído.

for $size from 1 to 3 (ele ignorará o 3)

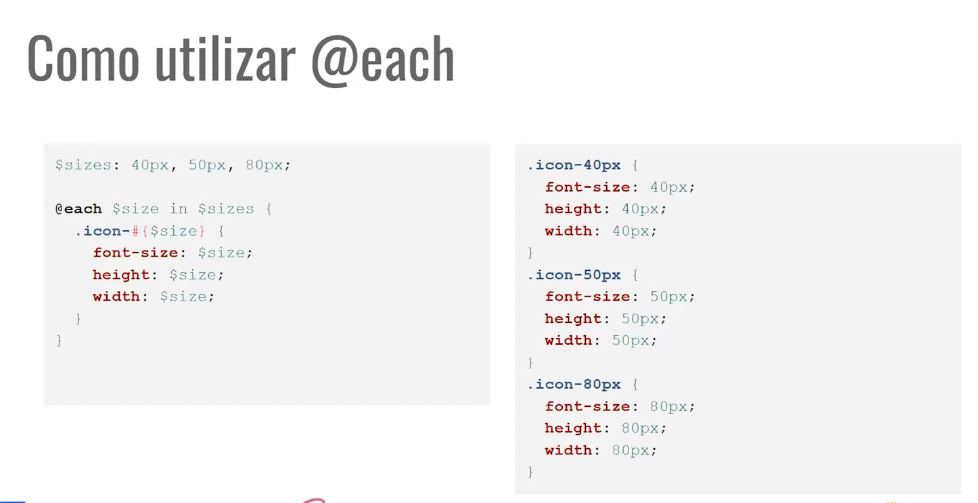
for $size from 1 through 3 (ele incluirá o 3)



**Controle de fluxo @each**

A regra @each é semelhante a regra @for, porém ela itera sobre valores de **listas ou mapas.** Ela facilita a emissão de estilos ou avaliação de código para cada elemento de uma **list** ou cada par de um **map.** É ótimo para estilos repetitivos que tem apenas algumas variações entre eles.

Geralmente é escrito **@each <variable> in <expression> {...},** onde a expressão retorna uma lista. O bloco é avaliado para cada elemento da lista, por sua vez, que é atribuído ao nome de variável fornecido.



**Controle de fluxo @while**

A regra @while, escrita @while <expression> {...}, avalia seu bloco se sua expressão retornar verdadeira. Então, se sua expressão ainda retornar verdadeira, ele avalia seu bloco novamente. Isso continua até que a expressão finalmente retorne falso (loop infinito).



**02/01 – FUNÇÕES ESPECIAIS**

São algumas funções dentro do SASS que quando nós executamos e ela gera o código CSS, ele ainda vai manter uma função, e vai gerar uma função compatível com CSS.

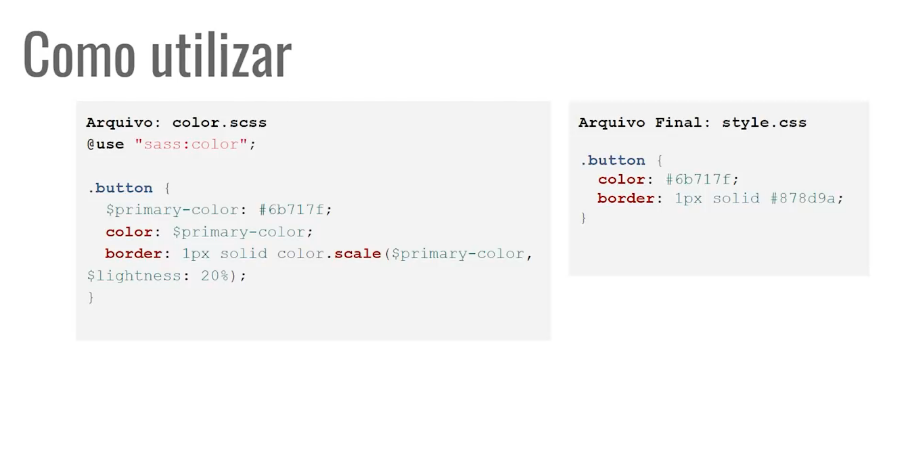
CSS define muitas funções e a maioria delas funciona muito bem com a sintaxe de função normal do SASS. Eles são analisados como chamadas de função, resolvidos para funções CSS simples e compilados como estão para CSS.

Existem algumas exceções, porém, que possuem sintaxe especial que não pode ser analisada apenas como uma expressão SASS. Essas que são as funções especiais, que retornam strings sem aspas.

**MÓDULOS NATIVOS**

O SASS fornece muitos módulos nativos que contém funções úteis (e ás vezes mixins). Esses módulos podem ser carregados com a regra @use como qualquer folha de estilo definida pelo usuário e suas funções podem ser chamadas como qualquer outro membro do módulo.

Todos os URLs de módulos integrados começam com sass: para indicar que fazem parte do próprio SASS.



**LISTA DE MÓDULOS BUILT-IN**

SASS possui os seguintes módulos nativos:

* **sass:math** – Módulo que disponibiliza funções para manipular números.
* **sass:string** – Módulo que permite facilmente combinar, buscar ou fatiar uma parte de uma string.
* **sass:color** – Módulo que cria novas cores baseadas em cores existentes, tornando fácil a criação de temas.
* **sass:list** – Módulo que permite acessar e alterar valores em uma lista.
* **sass:map** – Módulo que permite acessar valores associados a uma chave e muito mais.
* **sass:selector** – Módulo que permite acesso ao poderoso mecanismo de seletores do SASS.
* **sass:meta** – Módulo que exibe detalhes do funcionamento interno do SASS.

**SASS CLI – DART SASS**

O CLI (Command Line Interface) são os comandos do SASS que usamos dentro do terminal. Eles podem ser acessados usando o comando “sass --help”.

O Dart é a linguagem de programação da qual o SASS foi feito.

**OPERADORES**

O SASS oferece suporte a vários operadores úteis para trabalhar com valores diferentes. Isso inclui os operadores matemáticos padrão como + e \*, bem como operadores para vários outros tipos:

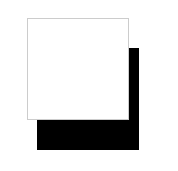
* == e != são usados para verificar se dois valores são iguais.
* +, -, \*, / e % têm seu significado matemático usual para números, com comportamentos especiais para unidades que correspondem ao uso de unidades em matemática científica.
* <, <=, > e >= verifica se dois números são maiores ou menores que um do outro.
* and, or e not tem o comportamento booleano usual. SASS considera todos os valores “verdadeiros”, exceto falso e nulo.
* +, - e / podem ser usados para concatenar strings.

**20/07 – RELEMBRANDO ALGUNS CONCEITOS**

**Box-shadow:** define a sombra da caixa de um elemento. (Todo elemento de uma página web é como uma caixa para o CSS, que é o que chamamos de box model. O box-shadow é o sombreamento dessa caixa)

**Valores possíveis:** none | initial | inset (sombra interna) | unset (sombra externa) | inherit |

h-offset v-offset blur spread color

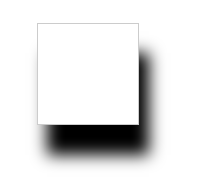
 box-shadow: 10px 30px 0 0 black;

h-off v-off blur spread color

Perceba que o deslocamento horizontal (no eixo X, o lateral) é de 10px. Por isso, vemos 10px de sombra no lado direito do box. Caso quiséssemos que essa sombra aparecesse do lado esquerdo, iríamos definir um número negativo para o h-offset.

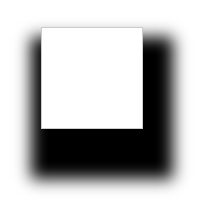
Veja também que o deslocamento vertical (no eixo Y, em cima ou em baixo) é de 30px, um pouco maior que o sombreamento horizontal. Por isso, vemos 30px de sombra abaixo do box. Caso quiséssemos que essa sombra aparecesse em cima do boz, iríamos definir um número negativo para o v-offset.

Agora, veja como a sombra fica quando definimos um blur:

 box-shadow: 10px 30px 15px 0 black;

Agora, vemos que os 15px de blur que adicionamos ao box-shadow deixaram a sombra do elemento levemente apagada, censurada, como um borrão.

Já usando o spread, temos o seguinte resultado:

box-shadow: 10px 30px 15px 20px black;

O spread faz com que o alcance da sombra do elemento se torne maior. Perceba que a sombra aumentou, mas o h-offset e v-offset não mudaram. O que aumentou a sombra foi o spread.

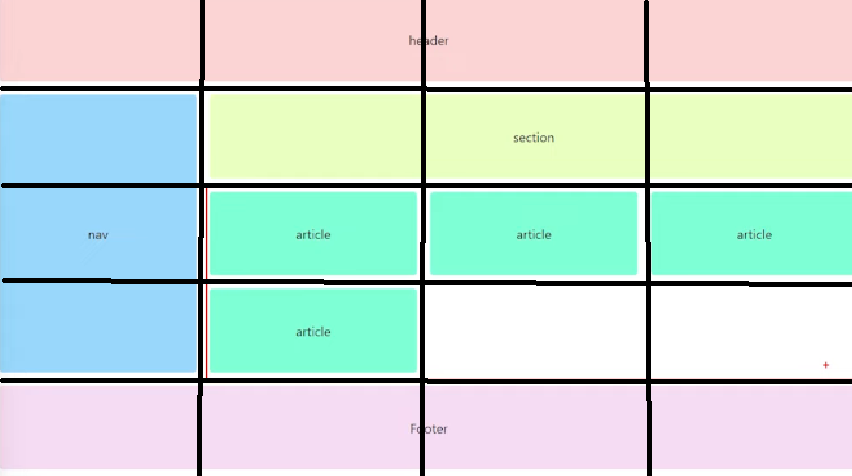
Há também a propriedade inset, que vai fazer com que a sombra se encontre dentro do próprio elemento, dessa forma:

box-shadow: 10px 30px 15px 20px black inset;

**DISPLAY GRID**

O Sistema de Grid do CSS nos permite desenvolver nosso layout utilizando um sistema de grade. Ele é formado por linhas e colunas.

Antes de colocarmos a mão no código, é importante vermos como o design do nosso site vai ficar. Podemos utilizar o Figma para isso:



Vemos então que o sistema de Grid que iremos usar possui 5 linhas e 4 colunas. Podemos determinar as medidas de cada linha utilizando **grid-template-rows,** passando a medida de cada linha:

**grid-template-rows:** 100px repeat(3, 1fr) 100px

Em outras palavras:

* a primeira linha do grid (o header) terá 100px de altura,
* as 3 linhas do meio terão medidas iguais (1fr, por isso foi usado o repeat)
* a quinta linha do grid (o footer) terá 100px de altura

Podemos determinar as medidas de cada coluna utilizando **grid-template-columns**, passando a medida de cada coluna:

**grid-template-columns:** 240px repeat(3, 1fr)

Em outras palavras:

* a primeira coluna do grid (nav) terá 240px de largura
* as 3 colunas restantes terão medidas iguais de 1fr (fração restante)

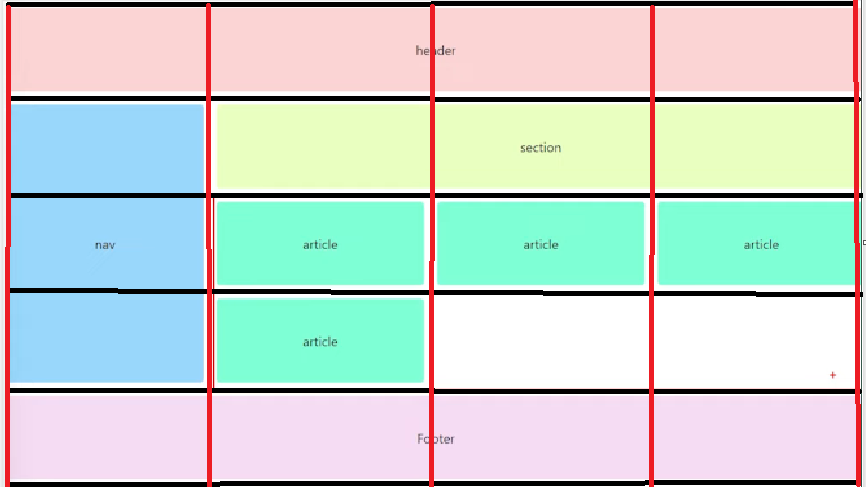
É possível também usar a propriedade **gap**, que irá determinar um vão entre as margens de cada célula do grid:

**gap:** 100px

Em outras palavras:

* cada célula terá espaçamento de 100px entre elas.

Além disso, podemos definir o tamanho de cada elemento do Grid em relação á linhas e colunas, utilizando **grid-column** e **grid-row.** Antes de fazer isso, devemos prestar atenção nas Grid Lines, que são linhas imaginárias que compõem o elemento Grid. No nosso caso, temos:



**6 Grid Lines de linha e 5 Grid Lines de coluna** (até as extremidades contam)

Veja como poderíamos definir o tamanho do header nesse layout:

**grid-column: 1 / 5** (ocupa da Grid Line 1 até a 5, referindo-se a coluna)

**grid-row: 1 / 2** (ocupa da Grid Line 1 até 2, referindo-se a linha)

Para se referir a última ou penúltima Grid Line, podemos usar **-1 (última) e -2 (penúltima).** Veja a aplicação disso para o nav:

**grid-column: 1 / 2**

**grid-row: 2 / -2** (ocupa da Grid Line 2 até a penúltima)

Além disso, há também a possibilidade de incluirmos novos elementos em nosso Grid que ainda não foram declarados. Por exemplo, numa consulta a um banco de dados, não sabemos quantos elementos virão. Por isso, com as propriedades **grid-auto-rows** e **grid-auto-columns,** podemos indicar qual vai ser o tamanho das novas linhas quando elas forem criadas.

**grid-auto-rows:** 100px (todas as linhas criadas dentro do elemento Grid pai terão 100px de altura)

**grid-auto-columns:** 100px (todas as colunas criadas dentro do elemento Grid pai terão 100px de largura)

**GRID AREA**

Podemos configurar nosso grid de um jeito mais fácil e intuitivo utilizando Grid Areas. Para fazer isso, nós definimos em cada elemento do nosso grid, qual nome o representará na grid-area:

header {

            grid-area: header;

        }

        section {

            grid-area: section;

        }

        nav {

            grid-area: nav;

        }

        footer {

            grid-area: footer;

        }

Após isso, nós configuramos o template as linhas e colunas do nosso grid pai, como a seguir:

*.grid-wrapper*{

            height: 100vh;

            display: grid;

            grid-template-rows: 100px 1fr 100px;

            grid-template-columns: 240px 1fr;

        }

Tendo 3 linhas e 2 colunas, nosso elemento Grid terá 6 células. Nós determinamos o que vai ficar em cada célula utilizando **grid-template-areas:**

*.grid-wrapper*{

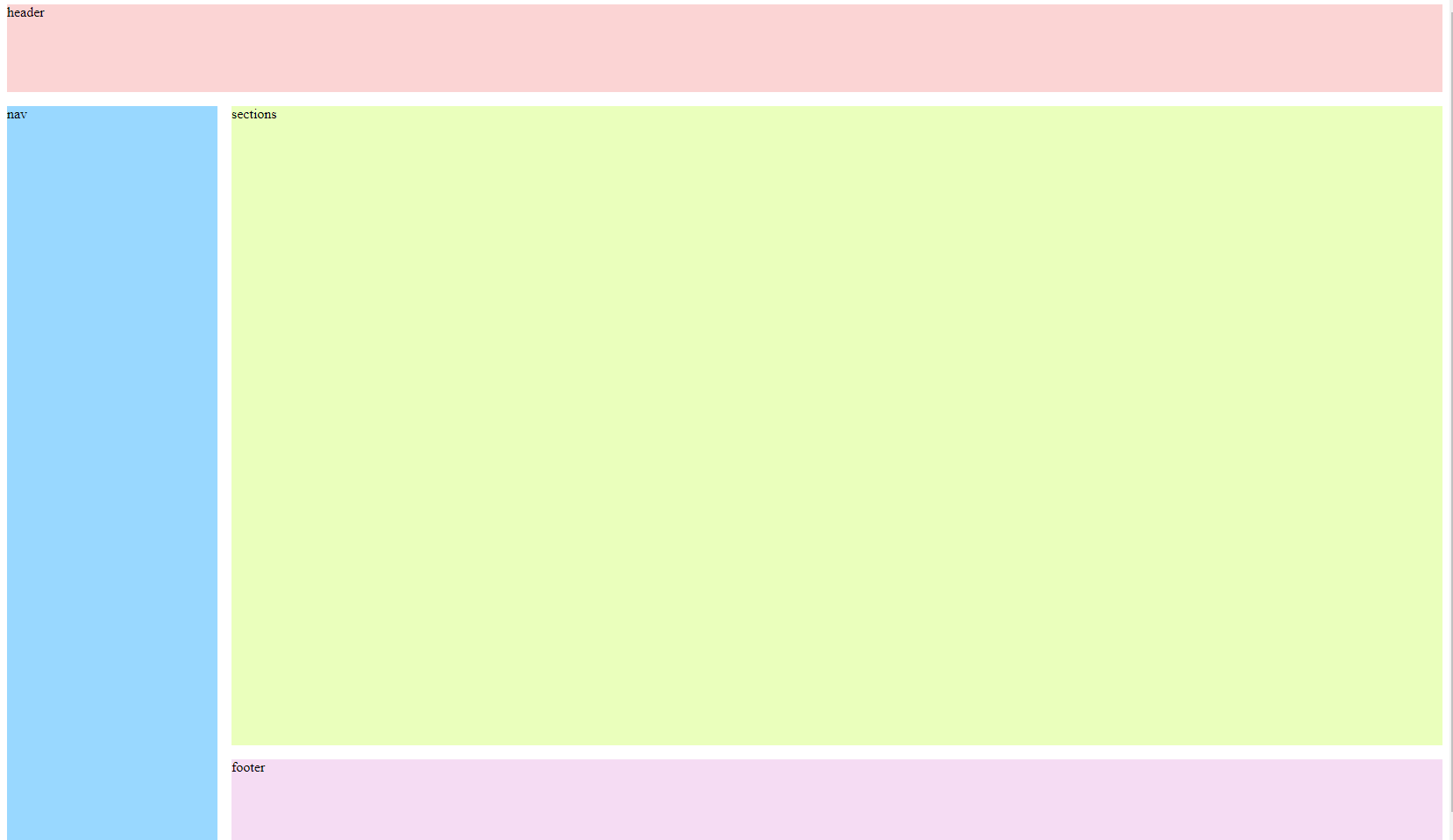
            grid-template-areas:

                "header header"

                "nav section"

                "nav footer";

        }



O resultado é esse! ;)

Além disso, usando as Grid Areas, podemos utilizar media queries para remover o nav caso o tamanho da tela do dispositivo seja pequeno. Veja:

@media only screen and (max-width: 600px){

            nav{

                display: none;

            }

*.grid-wrapper*{

                grid-template-areas:

                    "header header"

                    "section section"

                    "footer footer";

            }

        }



O resultado é esse! ;)