CSS

Cascading Style Sheet

Macetes

Função `clamp(min, pref, max)`

> Usada pra deixar o tamanho responsivo sem usar media queries.

pseudoclasse `:focus-within`

> Faz com que o foco de uma div ainda seja mantido caso os filhos sejam selecionados, útil para dropdown.

Força da cascada e especificidade.

Dependendo de como o estilo for aplicado à determinada tag, uma atribuição tem mais força que as outras.

Inline > tag Style do html > id > class > tag_nativa/::before/::after > Seletor universal *

```
### Origem do estilo

inline > tag style > tag link

### Especificidade

É um cálculo matemático, onde, cada tipo de seletor e origem do estilo, possuem valores a serem
considerados.

0. Universal selector, combinators e negation pseudo-class (:not())
1. Element type selector e pseudo-elements (::before, ::after)
10. Classes e attribute selectors ([type="radio"])
100. ID selector
1000. Inline
```

!**Important - >** Não é uma boa prática, só usamos para reescrever estilos de outras bibliotecas.

Seletores

Tags existentes

Sobreescrever estilos de tags existentes não se usa nenhum tipo de identificação no prefixo.

```
body {
}
```

Classes

Escrever estilos de classes usa se um ponto como prefixo.

```
.botao{
}
```

Ids

Escrever estilos de IDs especificos usa se um jogo da velha.

```
#id{
}
```

Todas as tags com um parametro/atributo especifico

Escrever estilos para todas as tags que possuam determinado atributo

```
[parametro ou atributo da tag]{
}

/* Pode-se usar uma expressao de igualdade também */

[id="class1"]{
```

```
}

Pode-se pegar todas as tags x que tem a classe = class1

x[id="class1"]{
}
```

Seletores aninhados

É possível escrever um estilo pra um objeto específico que está dentro de outro.

```
.botao#id{
}
```

Pseudoclass Selector

Escrever estilos para estados das tags.

```
.class1:hover{
}
```

Combinators

Na árvore de tags, os combinators direcionam o estilo para as tags que estão em posições especificas, ex: Primeiro filho, todos os filhos, pai.

General Combinator

Escreve espaço entre as classes, mudará todos as tags abaixo daquela classe.

```
body class1{
}
```

Child Combinator

Escreve > entre as classes. Mudará apenas os filhos da primeira tag, aqueles que são diretos.

```
body > class1{
}
```

Adjascent **Sibling** Combinator

Escreve + entre as classes. Mudará apenas o irmão do lado direito/abaixo da arvore de tags.

```
botao + botao{
  margin-left: 15px;
}
```

General **Sibling** Combinator

Escreve ~ entre as classes. Mudará TODOS OS IRMÃOS do lado direito/abaixo da arvore de tags.

```
botao ~ botao{
  margin-left: 15px;
}
```

:Pseudoclasses

Sempre começam com `:` antes.

Seleciona elementos em determinado estado.

Eles referenciam o próprio elemento e seu estado.

Exemplo `h1:hover`

:first-child

Se for primeiro filho

```
h1:first-child {
}
```

:nth-of-type()

O enésimo item do meu tipo. Muda o estilo do enésimo irmão de um tipo especificado. Começando de 1

```
h1:nth-of-type(1) {
}
```

:nth-child(n / odd / even)

O enésimo filho do pai. Muda o estilo do enésimo filho independendo do seu tipo. Começando de 1

odd ou even significa a estilização de ímpar ou par.

```
body h1:nth-child(1) {
}
```

:hover

Estiliza se estiver com o mouse em cima.

```
body h1:hover {
}
```

:focus

Geralmente usado em inputs ou buttons, estiliza se o input estiver selecionado/clicado

```
input:focus {
}
```

:atributos booleanos

Usado quando a tag tem atributos booleanos, como required, disabled, readonly, autofocus, etc.

```
input:disabled {
}
```

REFERENCIAS PSEUDOCLASSES.

Pseudo-classes - CSS | MDN (mozilla.org)

::Pseudoelements

Sempre começam com `::` antes.

Seleciona uma parte específica de uma tag e a estiliza.

Exemplo `h1::before`

::before

Muda a parte de trás de uma tag. O ideal é colocar um content.

```
h1::before {
  content="prefixo"
}
```

::after

Muda a parte da frente. O ideal é colocar um content.

```
h1::after {
  content="sufixo"
}
```

At rules

Regras no css com @arrobas,

@import

@font-face

@keyframes

@media

Shorthand

Dada certa propriedade, como margin, em vez de escrever

`margin-top: 10px;`

`margin-left: 10px;`

`margin-right: 10px;`

`margin-bottom: 10px;`

Pode se escrever `margin: 10px 10px 10px 10px;`

Isso é o chamado **shorthand**, as propriedades que possuem um hifen podem ser **resumidas com esse uso.**

Pode se omitir alguns valores se o css entender o valor padrão daquela linha.

Unidades de medida.

- **px** → Padrão
- **em** → Unidade de medida em relação à div pai.

  1 **em** significa que a unidade é igual a de medida do elemento pai.

  2 **em** significa que a unidade é 2 vezes a medida do pai.
- **rem** → Unidade de medida em relação à :root ou classe html.
- **vw** → **Viewport Width**: **1vw** é **1%** da largura do **dispositivo**.
- **vh** → **Viewport Height**: **1vh** é **1%** da altura do **dispositivo**.
- **fr** → **Fração:** 1fr é 1 fração, 100% do espaço disponível, é uma unidade de medida flexível util em grids e flexboxs.

Função calc

calc(calculo) → Permite fazer contas usando medidas, px, em, percentuais etc.

rgb (red - numero, green - numero , blue - numero)

rgba (red - numero, green - numero, blue - numero, alpha %)

hsl (hue - numero, saturation %, luminance %)

hsla (hue - numero, saturation %, luminance %, alpha %)

Posicionamento

Layouts de página: Formas como os objetos serão posicionados na página.

Propriedade **position**

`position: static` → Mantém a posição sem alterações.

`position : relative` → **Libera propriedades top, bottom, right, left e z-index**. Faz com que a manipulação de posição da caixa faça ela **flutuar pra onde quiser**. As **caixas vizinhas** ainda vão **respeitar o seu espaço pré-determinado.**

O nome se chama relative pois ela cria novos atributos que se relacionam com o posicionamento de uma caixa pai. Se eu peço um top de 10px, ele vai se afastar 10px de alguma referencia.

`position: absolute` → Também libera as propriedades top, bottom, right, left e z-index. A diferença entre a **relative** é que agora **essa caixa faz parte de outra camada**, **um plano acima do outro**. Dessa forma, as outras caixas **não respeitam mais o seu espaço**, visto que ela não divide mais espaço com elas, pois está em outra camada.

A visibilidade dessas camadas pode se controlado pelo z-index.

O absoluto também referencia o posicionamento da tag pai.

`position: fixed` → O elemento agora acompanha o scroll. Também libera as propriedades top, bottom, right, left e z-index.

Foi útil quando precisei colocar uma foto em uma div de fundo que substituia o body.

Queria deixar a opacidade da foto de fundo sem diminuir a opacidade dos elementos dentro dessa div.

Nesse caso, tive que criar uma div só pra imagem e editar as propriedades de background dela.

O position absolute permitiu que a imagem ficasse atrás das outras divs debaixo.

```css
.bgimg{
    position: absolute;
    width: 100%;
    height: 100%;
    background-position: center;
    background-image: url("busao.jpg");
    background-repeat: no-repeat;
    background-size: cover;
    opacity: .2;
    z-index: -1;
}
```

Propriedades

Color:

rgb/rgba

hsl/hsla

keywords

inherit (div pai), initial (root), unset.

linear-gradient(grau, cores)

Font:

Estrutura shorthand:

```
## Shorthand

* font-style, font-variant, font-weight, font-stretch, font-size, line-height, e font-family.
```

Data types:

`font-family: nome da fonte, fallback1, fallback2;`

`font-weight: normal / bold / bolder / light / lighter / 100 / 200 / 900;`

`font-style: normal / italic / oblique;`

`font-size: tamanho da fonte;`

Fontes da web

Importar usando o Google Fonts com o **<link> ou @font-face**

Mais estilização da fonte

`font-variant : small-caps;`

`font-stretch: condensed / expanded;`

`letter-spacing: tamanho;`

`word-spacing: tamanho;`

`line-height: altura entre as linhas;`

`text-transform: uppercase / lowercase / capitalize` → similar aos metodos de string python.

Text-decoration:

> É um shorthand para a decoração do texto.
>
> Ver mais no mdn

`text-align: center / left / right;` → Alinhamento do texto.

`text-shadow:` Shorthand para a sombra, igual ao box-shadow.

Box-sizing:

Dado width e height de uma caixa no css, por padrão, **essa width e height é ignorada pelo padding**, ou seja, ao adicionar padding, os valores da altura e largura da caixa serão recalculados e se somarão o padding

Isso porque, por padrão, o cálculo da altura e da largura é feito na parte do conteúdo da caixa, não da borda. Pode-se mudar isso com a propriedade box-sizing.

> `box-sizing : content-box;`
>
> ou
>
> `box-sizing : border-box;`
>
> ou
>
> `box-sizing : padding-box`

É muito utilizado pra organização.

Background:

Fundo de uma div, começa sendo transparente.

`background-color: cor`

`background-image: url`

`background-repeat:` `no-repeat / repeat-y / repeat-x`

`background-position:` `center / right / left / top / bottom / top right / top left / etc`

`background-size:` `contain (imagem contida) / cover (cobrir) / percentual /`

`background-origin:` `content-box / border-box / padding-box` → A partir de que area o background é irradiado.

`background-clip:` `content-box / border-box / padding-box` → Corte do background

`background-attachment` `: scroll / fixed` → scroll (o background anda scrolla com o conteudo), fixed (o background não scrolla, fica fixo)

Display:

Util para definir com que orientação as divs vão ser mostradas.

- `display:` `inline;` → Caixas na mesma linha, a primeira caixa não empurra a outra, pois ela tem display inline.
- `display:` `block;` → Essa caixa empurra as outras caixas pra baixo e acima dela.

block	**inline**
Ocupa toda a linha, colocando o próximo elemento abaixo desse	Elemento ao lado do outro
width e height são respeitados	width e height não funcionam
padding, margin, border irão funcionar normalmente.	Somente valores horizontais de margin, padding e border

Por natureza, algumas tags já são block ou inline, exemplo, **divs são display: block** padrão, enquanto **spans são inline**.

Flexbox

`display:` `flex;`

Jeito moderno de alinhar caixas.

Libera as propriedade do flexbox:

`flex-direction` `: (row ou column)`

Row os itens serão colocados um do lado do outro.

Column os itens serão colocados um em cima do outro.

`justify-content : center ;`

`align-items : center ;`

`align-self: center ;`

Grid

Um tipo de layout em forma de grade flexível que dá espaço em linhas e colunas para os filhos.

`display: grid;`

Libera as propriedades da grid.

`grid-template-areas:`

Define a area da grid por meio de strings que formam uma matriz, é similar ao jeito de criar uma matriz no latex.

```
body {
  display:grid;
  grid-template-areas:
  "item1 item1"
  "item3 item4"
  "item5 item5";
}
```

Nesse caso, **item 1** ocupa duas colunas e uma linha inteira.

Item 3 e **item 4** ocupam **uma coluna cada**, mas **fazem parte da mesma linha.**

Item 5 ocupa **duas colunas e a ultima linha.**

Esses nomes por si só não fazem nada, mas ao estilizar uma classe específica, podemos linkar essa classe ao nome que esta no `grid-template-area` usando o `grid-area: nome;`

```
body {
  display:grid;
  grid-template-areas:
  "item1 item1"
  "item3 item4"
  "item5 item5";
}
```

```
.item1 {
  background-color: red;
  grid-area: "item1";
}
```

Agora, item1 está linkado com o item 1 da grid-template-area;

`grid-template-rows / grid-template-columns: 1fr 100px 1fr;` Define a altura de cada linha ou a largura de cada coluna

A DIFERENÇA ENTRE DISPLAY FLEX/GRID E BLOCK/INLINE É QUE O PRIMEIRO TRABALHA COM O POSICIONAMENTO DE ELEMENTOS DENTRO DA CAIXA, ENQUANTO O SEGUNDO TRABALHA COM O POSICIONAMENTO DESSA CAIXA RELATIVO A OUTRAS.

Px vs 1fr vs Auto

Ao setar o tamanho de linhas e colunas com `grid-template-rows` e `grid-template-columns` o tamanho da caixa pode ser definido de 3 formas

- Pixels ou unidades de medida
- Fracionário
- Auto

Unidades de medida: Tamanho relativo somente a unidade de medida, não se adapta ao conteúdo ou a caixa.

Fracionário: Com unidades fracionarias, o espaço tomado pela caixa é uma fração do tamanho disponível dentro da própria caixa. **O tamanho da caixa não cresce se o conteúdo crescer.**

Auto: Tamanho se adapta ao tamanho do conteúdo da caixa. **Tamanho cresce se o conteúdo crescer.**

Shorthand grid-template

`grid-template` é um shorthand pra setar rows e columns ao mesmo tempo, separando por uma barra.

`grid-template: auto 1fr auto / auto 1fr auto;`

Repeat

Para criar grids rápido, em vez de digitar o tamanho pra todas as colunas, podemos usar a função `repeat(vezes, valor)` pra ele repetir tal valor x vezes.

RAM (Repeat, Auto, Minmax)

Dado uma grid que pode representar coisas como uma seleção de imagens de um aplicativo de imagens, queremos que essa lista de imagens respeitem um valor mínimo de px e que possam se esticar ou não caso o tamanho da caixa aumente.

`grid-template-columns : repeat(auto-fit/auto-fill, minmax(<base>, 1fr))`

Object Fit → Aspect Ratio e Fitting de imagem / video dado container

Uma imagem ou vídeo pode receber o atributo object fit para saber como ele irá fitar o container.

- `fill` - This is default. The image is resized to fill the given dimension. If necessary, the image will be stretched or squished to fit
- `contain` - The image keeps its aspect ratio, but is resized to fit within the given dimension
- `cover` - The image keeps its aspect ratio and fills the given dimension. The image will be clipped to fit
- `none` - The image is not resized
- `scale-down` - the image is scaled down to the smallest version of `none` or `contain`

Aspect-Ratio

Propriedade que define o aspect ratio de alguma caixa.

`aspect-ratio: 16/9`

Content:

Permite colocar algum conteúdo dentro de uma div, geralmente strings.

Margin e margin collapsing:

Margens servem pra dar espaços entre as divs, mas existe um conceito chamado margin collapsing que só acontece em divs com display block, onde uma está em cima da outra.

O collapsing é quando eu coloco margem acima e abaixo, mas a margem de cima e de baixo entre as divs não se soma, ou seja, não fica duplicado, elas se juntam em uma margem só, sobressaindo a que tiver o maior tamanho.

Elas deveriam dar conflito, mas por conta desse conceito, não dão.

Nas laterais não funciona

Margin:

margin: auto; → calcula as margens automaticamente nas laterais, somente as laterais.

Excluindo certas classes de herdar o css.

Vamos supor que eu tenho 3 páginas.

- Menu tem a div com classe: `class="menu.classe1.classe2"`
- Carrinho tem a div com classe: `class="carrinho.classe1.classe2"`
- Produto tem a div com classe: `class="produto.classe1.classe2"`

Eu quero aplicar um estilo especifico para toda div abaixo de `classe1.classe2` Mas, eu não quero aplicar àquelas divs que estão abaixo da classe `menu`

Logo, só será aplicado em `carrinho` e `produto`

Normalmente, no css, ficaria dessa forma:

```
.classe1.classe2 {
    estilo:;
}
```

Porém, eu quero excluir o `menu`, então, utilizamos a palavra reservada `:not(classe)` para excluir aquela classe de receber os estilos.

Ficaria assim:

```
:not(menu).classe1.classe2 {
    estilo:;
```

```
}
```

Agora, os estilos serão colocados em carrinho e produto, mas não em menu.

Alinhamento e FlexBox

Eixos da flexbox.

Dependendo do eixo da flexbox, as suas propriedades vão ser alteradas.

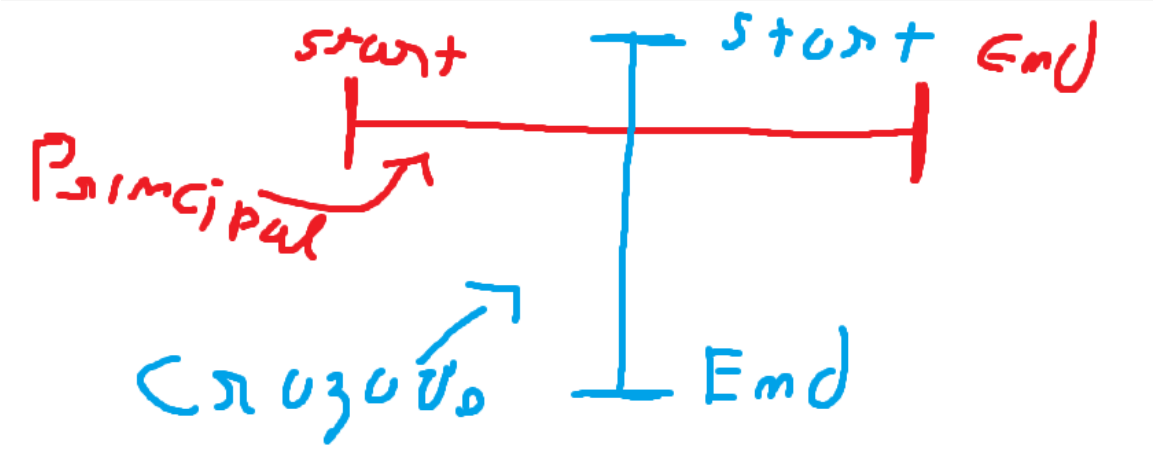
Eixos: **main** e **cross**

Mudar o **flex-direction** também muda os eixos.

> Row = Row principal.
>
> Column = Column principal.

O **align-items** trabalha com o eixo cruzado

O **justify-content** trabalha com o eixo principal



Propriedades da flexbox.

Ao transformar uma caixa em uma flexbox, ela libera uma série de propriedades.

- `flex-direction:` `row / row-reversed / column / column-reversed` → Reversed fazem com que os itens comecem de baixo pra cima e da direita pra esquerda, tipo àrabe

ou um mangá

- `flex-wrap: wrap` -> Faz com que objetos que ocupam a mesma linha sejam jogados pra baixo se faltar espaço pra eles preencherem. Isso cria uma nova row embaixo da atual, ela se comporta como um novo eixo.

  `wrap-reverse`; → Cria uma nova linha acima em vez de embaixo.

  `nowrap;`

- `flex-flow: flex-direction flex-wrap` -> Shorthand para o flex direciton e o flex wrap.
- `justify-content: center / flex-end / flex-start / space-between / space-around / space evenly` Da mesma forma da justificação de texto, essa propriedade faz com que o conteudo seja alinhado no eixo principal.
- `align-items: stretch / center / flex-start / flex-end` → Justificação de itens no eixo cruzado
- `gap: unidade de medida` → Espaço entre os elementos, como se fossem margens

Propriedades dos itens do flexbox

- `flex-basis: tamanho do eixo principal do item` **->** Essa propriedade é aplicada aos itens de uma flexbox, e substitui o width (direction = row) ou height (direction = column) dos itens do flexbox
- `flex-grow: 0 / 1 / 2 / 3...` → O flex grow faz com que um elemento ocupe o resto do espaço vazio do eixo principal de uma flexbox dinâmicamente. Se existir mais de um elemento com esse flex grow, o número informado indica quantas frações daquele espaço tal elemento deve ocupar.
- `flex-shrink: 0 / 1 / 2 / 3...` → O flex shrink faz com que um elemento diminua em relação ao espaço dos outros, ele distribui o espaço que ele tinha para os outros elementos.
- `flex: flex-grow flex-shrink flex-basis;` → Shorthand para as três propriedades acima. O flex basis tem preferencia sobre o flex grow e shrink
- `order: numero`

Animações

Propriedade Transition

O transition é uma propriedade que cria uma transição entre os estados de propriedades.

Shorthand `transition: property duration time-function delay`

Propriedade Animation

Adiciona uma animação customizada para o elemento.

Shorthand: `animation: name duration time-function delay iteration direction fill play-state`

Keyframes

Define uma animação e guarda seu nome

```
@keyframes nome {
  from {
    propriedades
  }

  to {
    propriedades
  }
}

Ou

@keyframes nome {
  0% {
    propriedades
  }

  100% {
    propriedades
  }
}

Pode se usar percentuais intermediários
```

`iteration:` Numero de iterações → numero ou infinite

`direction:` Direção da animação, → normal, reverse, alternate ou alternate reverse.

`fill:` Como as propriedades do elemento vão ficar depois que a animação for iniciada → forwards, backwards, both.

- `forwards`: o elemento fica com as propriedades do ultimo frame
- `backwards`: o elemento começa com as propriedades do primeiro frame, mesmo que a animação ainda não tiver começado.
- `both:` Uso dos dois.

Scroll Snap

Usado para dar snap em itens que estão próximos

Para isso é preciso um carrossel ou algum container sequencial

No container que contem os itens individuais: `scroll-snap-type : x/y mandatory/proximity`

Nos itens de cada container: `scroll-snap-align: center`

Gradiente no texto.

O esquema é fazer o background como um linear gradient, fazer uma mascara do background pra ficar como se fosse um luma key no texto e depois deixar o texto transparente.

```
.container {
  background: gradient
  -webkit-background-clip: text;
  color: transparent;
}
```

Desativar seleção de itens

No item: `pointer-events: none`

É legal usar com transições e animações de entrada.

Attr

Attr é uma função do css que pega o **conteúdo de um atributo html** e pode ser usado no css. Por exemplo, o `content:` de um pseudoelemento pode ser `content: attr(atributo)`, string, emoji, etc.

Mudando variaveis css pelo javascript

Para mudar uma variavel css pelo javascript, invocamos `document.documentElement.style.seProperty(”—nome-da-variavel”, valor)`