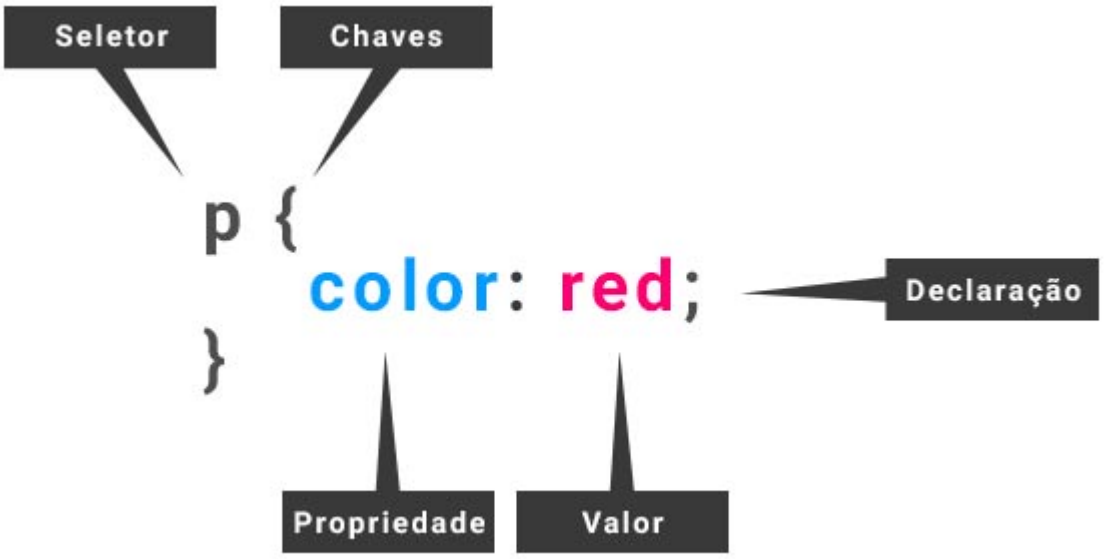
CSS



Anatomia

Funções:

calc();
rgb();

Pseudo-classe:

A pseudo-classe é usada para colocar estilos em diferentes estados, (quando passa o cursor em cima de uma certa aŕea, ou quando um link já foi visitado) Ela tem apenas um “:”.

Pseudo-elemento:

O pseudo-elemento tem dois “::”

Padrão de nomenclatura

A maioria das pessoas no CSS utilizam a nomenclatura de dash-case
Exemplo: borda-preta.

Observação: Sempre tente usar classes para o máximo de reutilização possível.

Alinhamento vertical

vertical-align: top; - **Ele só funciona quando o Display do seu elemento é Inline. Ele alinha o elemento ao lado, e esse elemento também tem que ser Inline.**

Seletores

- O '*' (Aplica o estilo na página toda)
- O '#aninhado > div' pega a DIV que está dentro da DIV #aninhado
- O '#aninhado section' pega a tag SECTION, mesmo que ela esteja dentro de outro elemento
- O 'p + ul' pega uma tag UL que esteja logo abaixo de uma tag P
- O 'p ~ ol' pega um elemento OL que esteja na mesma div PAI de um elemento P
- O '[feira]' pega o atributo 'feira' de dentro de uma tag

  O ~= é usado quando você quer pegar um valor de um atributo separado por espaço.
  Exemplo: <p feira = "escritorio empresa">
  [feira ~= "escritorio"]

  O *= é usado quando você quer pegar um valor que esteja dentro de uma frase.
  Exemplo: <p feira = "asa preta">
  [feira = "pre"]"

- span:nth-child(1) (Aplica o estilo se algum SPAN for o primeiro filho de alguém)
- section > :nth-child(odd) (Aplica o estilo nos elementos ímpares)
- section > :nth-child(even) (Aplica o estilo nos elementos pares)
- section > span:nth-of-type(2) Pega o segundo elemento SPAN de dentro de uma SECTION

Especificidade

Os que tem mais ESPECIFICIDADE:

1. Regra com **!important # Apenas use o !IMPORTANT se for realmente necessário.**
2. Seletores mais específicos
3. Última regra processada (empate)

Outros:

1. Inline
2. ID
3. Classe, Pseudo-Classe, atributo
4. Elemento, Pseudo-Elementos

Herança

Os elementos filhos muitas vezes podem herdar estilos do elemento PAI, mas não são todas as propriedades que são herdadas.

As propriedades BORDER e PADDING (entre outras..) não são herdadas do elemento PAI, mas se você quiser que elas herdem, você pode usar a propriedade **INHERIT**:

Exemplos:

BORDER: **inherit**;
PADDING: **inherit**;

Margin box

Observações:

A ordem é: EM CIMA, DIREITA, EM BAIXO, ESQUERDA

- O navegador coloca um valor de **8px** para o MARGIN no Body.

Display

display: block; - Cada elemento fica em sua linha.

display: inline; - Ele fica na mesma linha que outros elementos, e a largura e altura não podem ser mudadas.

display: inline-block; - Ele fica na mesma linha que outros elementos, mas a largura e altura podem ser mudadas

Comportamento estranho no inline-block: Um espaço em branco entre dois elementos pode ser contado como um caractere, quando utilizado o inline-block. Esse efeito faz com que pareça uma linha entre os dois elementos (se os dois elementos caberem na tela. Por exemplo: 49% para cada um) e também pode fazer com que um elemento fique embaixo do outro, em vez de ficar lado a lado. Para resolver isso, tire o espaço em branco entre os dois elementos, ou diminua de 1 a 3 pixels da largura, ou coloque o font-size do body com 0px.

DE

```
</nav>
<nav>
```

PARA

```
</nav><nav>
```

Altura e largura (Width and Height)

Quando usado a porcentagem em um 'HEIGHT', certifique-se que o elemento PAI tenha uma altura definida.

Ou colocar uma altura de 100% no **HTML** e no **BODY**.

Box sizing

Content-box

box-sizing: **content-box**; Faz com que o BORDER e o PADDING possam passar da altura e da largura.

Exemplo:

```
div {
        width: 400px;
        padding: 40px;
        border: solid 20px;
}
```

De vez a sua DIV ficar com 400px, ela ficará com 640. Porque o PADDING e o BORDER estão fazendo partes da LARGURA e da ALTURA.

Border-box

box-sizing: **border-box**; - **Faz com que o BORDER e o PADDING sejam contabilizados nos 400px.**

Todos os elementos com border-box

Para fazer com que todos os elementos da sua página fique com o border-box, use:

```
* {
        box-sizing: border-box;
}
```

Overflow

overflow: hidden; - Faz com que o conteúdo que extrapolou o elemento não seja visto.
overflow: scroll; - Faz com que apareça uma barra de rolagem se o conteúdo extrapolou o elemento.
overflow: visible; - Faz com que o conteúdo que extrapolou o elemento seja visto.
overflow-x - X é vertical
overflow-y - Y é horizontal

Float

A propriedade **Float** faz com que um conteúdo flutue em um dos lados da página.

Para fazer com que um conteúdo não tenha nenhuma flutuação ao lado dele, use a propriedade **Clear**.

Cores

Existem alguns modos de colocar cores no CSS:

Pelo nome da cor,

Hexadecimal,

RGB (**red, green, blue**), RGBA,

HSL (**hue, saturation and lightness**), HSLA.

O **"A"** é referente à *transparência* da imagem.

Unidades

Unidades absolutas

PX (Pixel) - Muito usado. O 1px do css não significa exatamente 1 pixel físico dos devices.
CM (Centímetro) - Pouco usado
MM (Milímetro) - Pouco usado
PT - Ponto
IN - Polegada - Pouco usado

Unidades relativas

Viewport é o tamanho visível da página
Acho que cada unidade do Viewport *corresponde a 1%* do tamanho visível da página

1vw = 1% da largura da página
1vh = 1% da altura da página
vmin = Pega a menor dimensão visível da página
vmax = Pega a maior dimensão visível da página

Exemplo: Uma página tem 200 pixels de largura e 100 de altura. Se usarmos 1 vw e 1vh em um font-size de um texto, o texto ficará com 2px de largura e 1px de altura.

REM - Ela é baseada no tamanho do elemento raiz. Acho que é o HTML
EM - Ela é baseada no tamanho do elemento PAI que tenha um tamanho referenciado. É recomendado sempre usar em tamanho da fonte.

Comportamentos diferentes quando usado a porcentagem junto com a propriedade POSITION.

position: static; - *A porcentagem irá se basear no conteúdo.*
position: relative; - *A porcentagem irá se basear no conteúdo.*
position: absolute; - *A porcentagem irá se basear no conteúdo e no padding.*
position: fixed; - *A porcentagem irá se basear no viewport.*

Tamanhos padrões:

font-size: **1em**;
font-size: **100%**;
font-size: **16px**;
font-size: **12pt**;

Texto

Sempre use alternativas de fonte, para que se um estilo de fonte não esteja disponível, o navegador tenha opções para substituir essa fonte.

font-weight: 100; - **Peso da fonte**
font-style: italic; - **Estilo da fonte**
text-transform: uppercase; - **Maiúsculo, minúsculo, capitalizado...**
text-decoration: overline; - **Overline, underline, line-through…**
word-spacing: 10px; - **Espaço entre as palavras**
letter-spacing: 20px; - **Espaço entre as letras**
line-height: 2.5em; - **Altura da linha**
text-align: center; - **Alinhamento do texto**

Pseudo-classe para quando o botão for pressionado:

botao:**active** { background-color: blue; }

Position

position: static; - É a posição padrão dos componentes, ele não aceita nenhum valor de posicionamento.

position: fixed; - É uma posição que fica fixa. Os valores se baseiam nos cantos da página.

position: relative; - É uma posição que leva em consideração o seu posicionamento normal.

position: absolute; - É uma posição que se baseia nos cantos da página. Ela é igual ao fixed, mas o diferencial é que ela não fica fixa. Se a DIV Pai tiver um **Position Relative**, o ABSOLUTE vai se basear nos cantos da DIV Pai, e não nos cantos da página.

position: -webkit-sticky; Safari - O Safari usa esse "web-kit-sticky". Coloque esse valor para funcionar no Safari.

position: sticky; - É uma posição que o elemento é posicionado com base na posição de rolagem definida pelo usuário.

Media Query

@media (max-width: 600px) { display: block; } - Se a largura do dispositivo for menor ou igual a 600px, o estilo será aplicado.

@media (max-width: 1000px), (orientation: portrait) { display: block; } - Se a largura do dispositivo for menor ou igual a 1000px, **ou** a orientação estiver como Retrato, o estilo será aplicado.

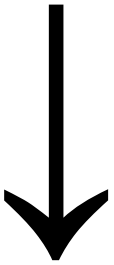
@media (min-width: 500px) and (max-width: 1000px) { display: block; } - Se a largura do dispositivo for maior ou igual a 500px **e** menor ou igual a 1000px, o estilo será aplicado.

Orientation:

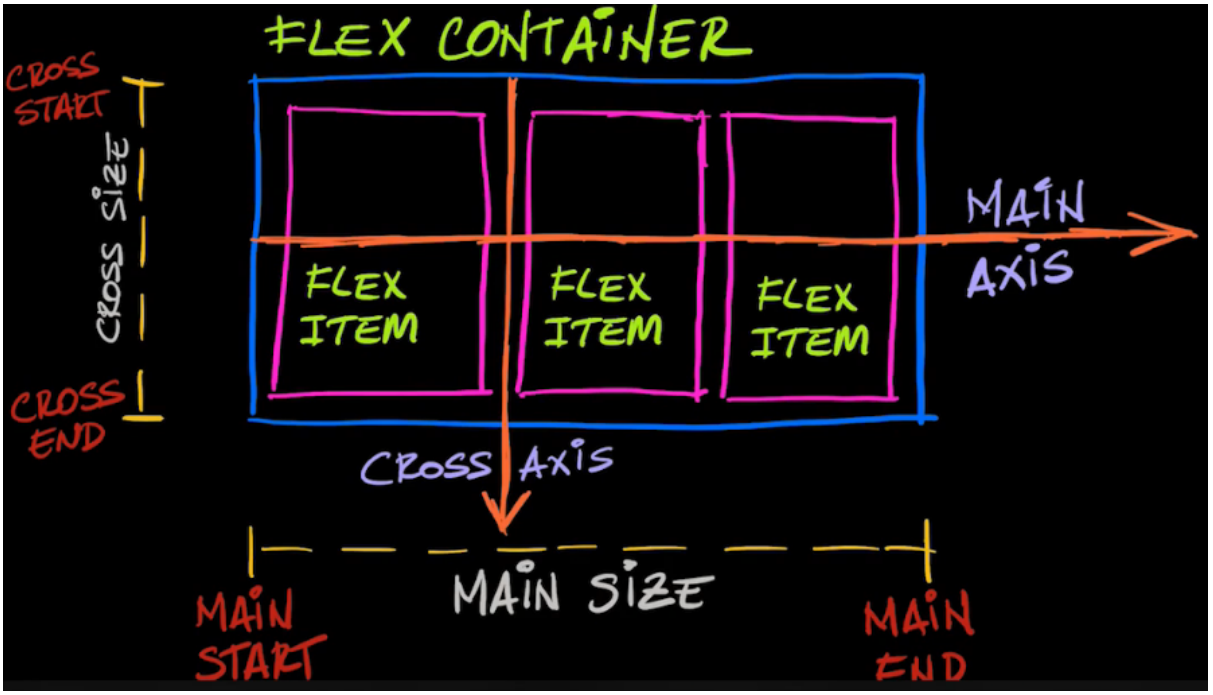
Landscape - Paisagem
Portrait - Retrato

Observações para o PADDING

Alguns elementos já vem com um **margin** e **padding** padrão. Sempre tire o valor padrão para centralizar um componente corretamente.



Flexbox



Main Axis - Eixo Principal
Cross Axis - Eixo alternativo
Display Flex - Flex Container

O Flexbox tem apenas dois eixos: Linha ou Coluna.

Dentro de um **Flex Container** existe **Flex Items**.

Main Axis * Eixo primário *

display: flex; - A Partir de agora a div é um 'Flexbox'.

Flex-direction

flex-direction: row; - O Eixo padrão de um Flexbox é o de linha, ele é baseado na largura.
flex-direction: column; - É o eixo de coluna, ele é baseado na altura.
flex-direction: column-reverse; - Ele inverte a ordem dos elementos.
flex-direction: row-reverse; - Ele inverte a ordem dos elementos e coloca os elementos no lado direito.

Flex-Wrap e **Flex-flow**

flex-wrap: nowrap; - É o valor padrão. Ele não quebra a linha.
flex-wrap: wrap; - Faz com que os elementos quebrem de linha se eles passarem da largura.
flex-wrap: wrap-reverse; - Faz com que os elementos quebrem de linha se eles passarem da largura, mas a primeira linha vai para baixo e a segunda linha vai para cima.
flex-flow: row wrap; - Faz com que o Eixo seja o de linha e quebre a linha.
flex-flow: column-reverse wrap; - Coloca o eixo como coluna, invertendo a ordem dos elementos e quebrando de linha.

Justify-content * Eixo principal *

justify-content: flex-start; - Ele justifica no começo no eixo principal.
justify-content: flex-end; - Ele justifica no final no eixo principal.
justify-content: center; - Ele justifica no centro no eixo principal.
justify-content: space-around (Em volta do elemento); - Ele coloca espaços uniformes nas laterais dos elementos no eixo principal.
justify-content: space-between; - Coloca espaço no meio entre os elementos no eixo principal.

Cross Axis * Eixo secundário *

align-items: stretch; - É o valor padrão. Ele estica o elemento.
align-items: baseline; - Alinha pela linha de base
align-items: flex-start; - Alinha no início
align-items: flex-end; - Alinha no final
align-items: center; - Alinha no meio

Mais de uma linha

align-content: flex-start; - Todas as linhas vão se alinhar no início no eixo secundário.
align-content: flex-end; - Todas as linhas vão se alinhar no final no eixo secundário.
align-content: center; - Todas as linhas vão se alinhar no meio no eixo secundário.
align-content: space-around; - Ele coloca espaços uniformes nas laterais de cada linha no eixo secundário.
align-content: space-between; - Coloca espaço no meio das linhas no eixo secundário.

Order

order: 1; - A ordem do elemento fica por último
order: 0; - É a ordem padrão
order: -1; - A ordem do elemento será a primeira.

Align-self

O align-self serve para alinhar um elemento específico no eixo secundário.

Grow e Shrink

O **Grow e o Shrink** soma o peso de todos os elementos e divide entre eles para preencher um espaço em branco.

O **Grow** faz com que o elemento cresça para preencher um espaço em branco, e o **Shrink** faz com que o elemento encolha, assim fazendo o espaço em branco ficar maior, podendo utilizar o **grow** para aumentar.

Exemplo:

```
.flex-container {
	display: flex;
	width: 400px;
}

.flex-container div {
	width: 150px;
}

.flex-container .primeira-div {
	flex-grow: 2;
}

.flex-container .segunda-div {
	flex-grow: 1;
}
```

Esse **Flex-container** tem um espaço em branco de 100px. A **primeira DIV** ficará com **216px** e a **segunda DIV** com **183px** de largura.

Com o flex-shrink a **primeira DIV** ficará com px e a **segunda DIV** com px de largura.

Flex-basis

O **flex-basis** serve para definir a largura de um elemento, se o **flex-direction** for o de **linha**, e a **altura** se o **flex-direction** for o de **coluna**.

Ele terá mais **especificidade** que o **width** se o **flex-direction** for o de **linha**.

Definindo valores para GROW, SHRINK E BASIS em uma linha (shortcut).

flex: grow, **shrink**, **basis**
flex: 3 6 100px;

GRID

```
<div id="div-grid">
   <div id="div1">DIV 1</div>
   <div id="div2">DIV 2</div>
   <div id="div3">DIV 3</div>
   <div id="div4">DIV 4</div>
</div>
```

display: grid; - Coloca esse elemento como uma GRID.

grid-template-columns: - Define quantas colunas a sua GRID vai ter.

Formas de definir quantas colunas a sua GRID vai ter:

grid-template-columns: 50% 50%; - Isso está dizendo que sua GRID vai ter duas colunas com 50% de largura cada.

grid-template-columns: repeat(2, 50%) **ou repeat**(1, 50% 50%) **ou repeat**(1, 50%) 50% **- Repete** o 50% duas vezes e faz duas colunas com 50% cada.

grid-template-columns: 20% 1fr (*fragmento*); - A primeira coluna vai ter 20%, e a segunda coluna vai ficar com a largura restante.

minmax(100, 300px) - Faz com que sua tenha no mínimo 100px de largura e no máximo 300px de largura.

grid-template-rows: Define a altura das linhas da sua GRID.

Formas de definir a altura das linhas em uma GRID:

grid-template-rows: repeat(2, 50%); - Está definindo que as duas linhas que a sua GRID tem vai ter 50% cada.

grid-template-rows: 25%; - A primeira linha vai ter 25% e a segunda linha irá ficar com a altura restante.

grid-template-rows: 25% auto; - A primeira linha vai ter 25% e a segunda linha vai ficar com a altura restante.

Mudando a quantidade de coluna/linha que um elemento irá ocupar.

#div2 {
grid-column|row-start: 1; - Vai começar na coluna/linha 1.
grid-column|row-end: 3; - Vai terminar na coluna/linha 3.

OU

grid-column|row-start: 1; - Vai começar na coluna/linha 1.
grid-column|row-end: span 2 - Vai começar na coluna/linha 1 e terminar na coluna/linha 3.
}

Sobreposição de células.

```
#div1, #div2 {
   grid-column-start: 1;
   grid-column-end: span 1;
   grid-row-start: 1;
   grid-row-end: span 1;
}
```

Nesse caso, a **DIV2** irá **sobrepor** a **DIV1**.

Para fazer com que a **DIV2 não sobreponha a DIV1**, é só usar a propriedade **Z-INDEX**.

#div1 {
z-index: 1; Pode ser 10, 100, 1000.. Quanto maior o número, maior a preferência para sobrepor os outros elementos.
z-index: -1; Irá fazer com que a **DIV2 volte a sobrepor a DIV1**.
}

Dando nome para as colunas

grid-template-columns:
[**primeira-coluna**] 1fr
[**coluna-do-meio meio-coluna**] 1fr *** Dois nomes ***
[**metade-2 meio-2**] 1fr [**ultima-coluna**];

Usando o nome:

grid-column-start: **primeira-coluna**;
grid-column-end: **ultima-coluna**;

OU

grid-column: **primeira-coluna / fim**

Nesse **shortcut**, é preciso usar o **/**

Com linha

grid-row: **1 / span 2**;

Outro atalho

row-**start** column-**start** row-**end** column-**end**
grid-area: 1 / meio-1 / span 3 / fim**;**

Dando espaço entre as linhas e colunas.

column-gap: 20px; **- Dá um espaço de 20px entre as colunas**
row-gap: 20px; **- Dá um espaço de 20px entre as linhas**
gap: 50px 10px; **- Dá um espaço 50px nas linhas e 10px nas colunas**
gap: 10px; **- Dá um espaço de 10px tanto nas linhas e nas colunas**

Dando nomes para os elementos

```
header {
    grid-area: cabecalho;
}

nav {
    grid-area: navegacao;
}
```

```
main {
        grid-area: conteudo;
}

footer {
        grid-area: rodape;
}

body {
        display: grid;
        grid-template-columns: 300px 1fr;
        grid-template-rows: 100px 1fr 100px;
        grid-template-areas:
                COLUNA COLUNA
                'cabecalho cabecalho' LINHA
                'navegacao conteudo' LINHA
                'rodape rodape'; LINHA
}
```

Isto pode ser usado para mudar o layout do seu site em dispositivos móveis.

```
@media(max-width: 700px) {
        body {
           grid-template-columns: 1fr;
           grid-template-rows: 80px 1fr 100px;
           grid-template-areas: 'navegacao' 'conteudo' 'rodape';
        }

        header { display: none; }
}
```

Formas de centralizar sua GRID:

O **JUSTIFY** e o **ALIGN** do **flexbox** também podem ser usados na **GRID**.

Outros:

Transform

**** Translate**

translateX: Ele faz com que o elemento de um distanciamento no eixo horizontal sem mudar o flow(overflow).

translateY: Ele faz com que o elemento de um distanciamento no eixo vertical sem mudar o flow(overflow).

translate(100px, 150px) - 100px de distanciamiento no eixo Horizontal (X) e 150px de distanciamento no eixo Vertical (Y)

**** Scale**

scaleX(2) - 2 vezes o tamanho no eixo Horizontal (X)
scaleY(2) - 2 vezes o tamanho no eixo Vertical (Y)
scale(2) - 2 vezes o tamanho nos dois eixos

**** Rotate**

rotate(45deg) - Rotaciona o elemento em 45deg
rotate(-45deg) - Rotaciona o elemento em 45deg negativos
rotate(0.5turn) - Dá meia volta no elemento (180deg)

**** Skew**

skewX(20deg) - Inclina 20deg no eixo X
skewY(20deg) - Inclina 20deg no eixo Y
skew(20deg) - Inclina 20deg em ambos os eixos

**** Juntando**

translateX(100px) **scale**(1.5) **rotate**(-15deg)

Transition

transition-property: background-color, border-radius;
transition-duration: 2s, 1.5s;

transition-property - São as propriedades que irão fazer parte da transição.
transition-duration - É o tempo que a transição vai durar.

transition-delay: 1s; Vai esperar 1 segundo para começar a transição.

** Timing Function

transition-timing-function: ease-in; - Dá uma suavizada no começo da transição

transition-timing-function: ease-out; - Dá uma suavizada no final da transição

transition-timing-function: ease-in-out; - Dá uma suavizada no começo e no final da transição

transition-timing-function: cubic-bezier(.15,.89,.85,.26) - Ver ainda

aaa