



UNIVERSIDADE DE
VASSOURAS

**LABORATÓRIO DE PROGRAMAÇÃO DE
WEB SITES**

CSS

Prof. Esp. Caio Jannuzzi

Sumário

CSS

O que é	03
Iniciando	03
Sintaxe	05
Comentários	05
Variáveis	06
Aplicando fontes	07
Reset	08
Unidades de medidas	10
Cores	12
Seletores	14
Preenchimentos	16

Sumário

Display	19
Position	20
Flexbox	25
Responsividade	33
Grid	37

CSS

O que é

CSS (Cascading Style Sheets ou Folhas de Estilo em Cascata) é a linguagens de marcação (não de programação, assim como o HTML) responsável por adicionar estilos em nossas páginas feitas com HTML, como cores, tamanhos, posicionamentos e entre outros. Sem ele, as páginas são apenas um grupo de textos, links e imagens.

Iniciando

Etapa 1: Criar um arquivo CSS. A primeira coisa que precisamos fazer é criar um arquivo CSS do qual nossa página HTML possa obter seu estilo. Então em seu editor de código basta você criar um novo arquivo chamado ele de style.css. Lembre-se que o nome pode ser qualquer um, mas precisar ter o .css no formato do arquivo.

Etapa 2: Linkar seu CSS no HTML. Depois de criado precisamos conectar nosso arquivo style.css no documento HTML. Dentro da tag <head> do documento HTML, vamos adicionar uma tag <link> para conectar a nossa nova folha de estilo.



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

Certifique-se de que sua página HTML e sua folha de estilo CSS estão no mesmo nível de pasta, caso ele esteja dentro de uma pasta diferente do arquivo HTML basta colocar o nome da pasta antes do nome do arquivo separado com uma barra (/).

Etapa 3: Chame e dê estilo aos elementos. Experimente selecionar um elemento e estilizá-lo.



```
1 h1 {  
2   color: #00f;  
3   font-size: 26px;  
4 }
```

Acabamos de chamar todas tags <h1> do nosso documento HTML e adicionar o tamanho da fonte para 26px e com a cor azul (sempre atualize a página em seu navegador assim que aplicar algo novo em seu arquivo CSS ou HTML).

Sintaxe

A sintaxe do CSS é bem simples.

Primeiro precisamos indicar um seletor (pela tag, id ou class) e dentro das chaves inserimos os comandos referente à formatação, que são as propriedade e valor.

```
1 seletor {  
2   propriedade: valor;  
3 }
```

Comentários

Para adicionar um comentário em nosso documento CSS, é preciso utilizar o comando `/* */`. Portanto, todo o código que estiver dentro dessa tag não será executado.

```
1 div {  
2   /* color: blue; */  
3 }
```

Variáveis

Uma aplicação grande geralmente tem uma quantidade muito grande de arquivos CSS, e com uma quantidade de repetição de valores muito frequente. Por exemplo, a mesma cor pode ser usada em vários lugares diferentes em nosso documento, caso seja requerido uma substituição na cor, teríamos que procurar todos os lugares que essa cor foi adicionada. Já com variáveis CSS a história muda. Ela permite que um valor seja guardado em uma variável, para ser referenciado em muitos outros lugares. Isso é ótimo para nós desenvolvedores pois, caso seja solicitado a substituição de trocar o valor dessa variável, apenas um lugar será substituído para que todo nosso documento sofra alteração.

Elas são configuradas usando esta notação:

```
1 :root {  
2   --cor: black;  
3 }
```

O :root se equipara à raiz de uma árvore, que representa o documento.

E são acessadas usando a função var():



```
1 h1 {  
2   var(--cor);  
3 }
```

Aplicando fontes

Para colocar uma fonte em sua página primeiro é preciso chamar ele no seu HTML, para isso vamos entrar no site do Google Fonts. Logo em seguida, procure a fonte que você deseja e clique no botão "Select this style" para assim adicionar os estilos de fontes que você deseja.

Perceba que quando você selecionou abriu uma aba na lateral na direita. Nessa aba vai ter duas opções para aplicar a fonte em sua página, uma é o <link> e o outro @import. O <link> é para importar pelo seu documento HTML e o @import é para o seu documento CSS. Você pode escolher qualquer um, isso vai com seu gosto, mas importar pelo HTML nos trás uma melhor performance.

Perceba que logo depois de selecionar a opção <link> é apresentado um código HTML e uma propriedade CSS, o font-family. É exatamente ele que usaremos em nossa página CSS para escolher quais elementos irão receber a nossa fonte. Isso porque podemos ter muitas fontes no CSS.

Em nosso documento HTML dentro da tag <head> coloque o código HTML apresentado no site do Google Fonts e no documento CSS coloque propriedade font-family e o nome da fonte.

```
1 * {  
2   font-family: 'Nome da fonte aqui';  
3 }
```

Reset

Agora vamos desfazer os padrões dos navegadores.

"Mas espera, o que isso quer dizer?"

Quer dizer que os navegadores tem um padrão, e alguns desses padrões não são legais para nós desenvolvedores, então vamos tirar alguns deles.

Quando criamos uma página HTML, por padrão, nosso site tem um espaçamento em volta da página. Para tirar esse padrão usamos duas coisas: o padding (espaçamento dentro do conteúdo) e o margin (espaçamento fora do conteúdo). Iremos zerar eles.

```
1 body {  
2   padding: 0;  
3   margin: 0;  
4 }
```

E por último, vamos aplicar a propriedade box-sizing: border-box. Esse é super importante na criação dos elementos pois, quando criamos um container sempre usamos uma largura e altura (no nosso exemplo vai ser 300px em cada um) mas caso você queira colocar um espaçamento interno nele (padding) de 30px, o elemento vai deixar de ser 300px e será 330px.

Mas a gente não quer isso, não é? Queremos que o elemento continue 300px e com um espaçamento interno. É aí que entra o nosso querido box-sizing: border-box. Ele irá manter o tamanho do container e com o valor do espaçamento aplicado nele.

```
1 * {  
2   padding: 0;  
3   margin: 0;  
4   box-sizing: border-box;  
5 }
```

Unidades de medidas

O CSS oferece um número de unidades diferentes para a expressão de comprimento. Iremos citar só duas delas: PX (medida absoluta) e REM (medida relativa).

Medidas absolutas: Essas são as mais comuns que vemos no dia a dia. São medidas que não estão referenciadas a qualquer outra unidade, ou seja, não dependem de um valor de referência. São unidades de medidas definidas pela física, como o pixel, centímetro, metro, etc...

Medidas relativas: O uso delas é mais apropriado para que possamos fazer ajustes em diferentes dispositivos garantindo um layout consistente e fluido em diversas mídias.

PX

Definir unidades de texto em pixels traz uma desvantagem. Quando o usuário tenta mudar o tamanho do texto pelo navegador ou na configuração do celular, o texto na nossa aplicação não aumenta. Pelo simples motivo de que o texto está definido em pixels (medidas absoluta). Isso é um problema sério de acessibilidade. É por isso que muitos desenvolvedores preferem definir o tamanho dos textos utilizando outras medidas em vez de trabalhar com pixels.

```
1 h1 {  
2   font-size: 18px;  
3 }
```

REM

Diferente do pixel, o rem é uma unidade escalável e relativa, ele varia de acordo com a dimensão root do seu navegador (por padrão essa unidade é 16px), na maior parte das vezes $1\text{rem} = 16\text{px}$. Podendo variar se o tamanho da fonte raiz for alterado.

```
1 h1 {  
2   font-size: 1rem; // Igual a 16px  
3 }
```

Como disse acima, o valor base é 16px, e isso pode acabar gerando dificuldades para que encontremos alguns tamanhos padrões que costumam ser utilizados. Por exemplo, como faríamos para atingir um tamanho de 10px utilizando rem? Para isso precisamos calcular. Mas ficar calculando esses números não é algo muito "amigável", porém, podemos adicionar um pequeno truque para nos ajudar.

Consiste em mudar o valor do font-size do html para 62,5%, que seria igual 10px. Dessa forma 1rem = 10px, fazer a conversão torna-se bem mais simples já que basta dividir o valor em pixel por 10 e se obterá o valor em rem. Exemplo: 1.2rem = 12px, 2.6rem = 26px, 5.6rem = 56px.



```
1 html {  
2   font-size: 62,5%;  
3 }  
4  
5 h1 {  
6   font-size: 1.2rem; /*equivalente a 12px*/  
7 }
```

Cores

RGB e HEX são formatos de composição de cores. Após compreender esse tema você pode escolher um dos formatos para utilizá-la mais frequentemente nos projetos. Mas esses não são os únicos formatos disponível no CSS, porém são os mais utilizados.

RGB

O processo é simples: como na vida real onde você mistura cores para obter uma outra cor como resultado, você faz a mesma coisa como o RGB.

Sua sintaxe utiliza a soma de 3 valores: Red, Green e Blue. Onde o valor máximo de todas as cores é 255 e o mínimo é 0.



```
1 h1 {  
2   color: rgb(255, 200, 10);  
3 }
```

Graças a função RGBA (red-green-blue-alpha) podemos adicionar opacidade no formato RGB.



```
1 h1 {  
2   color: rgba(255, 200, 10, 0.4);  
3 }
```

O alpha define a opacidade como um número entre 0,0 (totalmente transparente) e 1 (totalmente opaco).

HEX

Hex, ou Hexadecimal, tem sintaxe muito mais curta que o RGB. O código Hexadecimal consiste em seis letras ou números precedidos por um "#".

Os seus dois primeiros valores representam a intensidade do vermelho, terceiro e quarto a intensidade de verde e os dois últimos a intensidade de azul.

```
1 h1 {  
2   color: #00ff00;  
3 }
```

Seletores

"Okay, mas se eu tiver mais de um elemento <p> e quiser mudar a cor do texto só de um?".

É aí que entram os identificadores ID e Class:

- As Classes são uma forma de identificar um grupo de elementos. Através delas, pode-se atribuir formatação a VÁRIOS elementos de uma vez.
- O ID é uma forma de identificar UM elemento, e devem ser única para cada elemento. Através dele, pode-se atribuir formatação a um elemento em especial.

Para fazermos referência a uma Classe usando um . (ponto) com o nome da Classe e para fazermos referência a um id usando um # (hashtag) no lugar do . (ponto). Podemos ter os dois no mesmo elemento também.

```
1 .souClass {  
2   color: #f00;  
3 }
```

Nosso HTML ficará assim ao aplicar a Classe “souClass” ou ID.

```
1 <p class="souClass">Sou um elemento com Class</p>  
2  
3 <p id="souID">Sou um elemento com ID</p>
```

Outros seletores

Dependendo da estrutura da nossa aplicação, selecionar elementos acabam sendo complexos de certa forma, mas existem algumas formas de selecionar tags, ID e Classes. Vou apresentar somente três delas.

Seleciona o primeiro parágrafo que vem exatamente após a div:



```
1 div + p {  
2   color: #f00;  
3 }
```

Seleciona todos os elementos <p> com a Class "iuricode":



```
1 p.iuricode {  
2   color: #f00;  
3 }
```

Seleciona todos os parágrafos que são filhos diretos da div:



```
1 div > p {  
2   color: #f00;  
3 }
```

Preenchimentos

As propriedades de preenchimento CSS são usadas para gerar espaço em torno ou dentro do conteúdo.

Padding

A propriedade padding é usada para gerar espaço em torno do conteúdo. O padding "limpa" uma área ao redor do conteúdo (dentro da borda) de um elemento.

Margin

A propriedade margin define o tamanho do espaço de fora da borda. O margin é usado para gerar espaço em torno do elemento.

Resumindo: Padding é o espaço entre o conteúdo e a borda, enquanto margin é o espaço fora da borda



Elas podem ser aplicadas nos quatro lados de um elemento: superior, direita, inferior e esquerda (nessa ordem).

No exemplo a seguir vamos usar o margin mas também pode ser aplicada a propriedade padding.

Irei aplicar margin de 20px iguais nos quatros lados do elemento:

```
1 .class {  
2   margin: 20px;  
3 }
```

Agora irei aplicar uma margin superior e inferior de 5px e margin esquerda e direita de 10px:

```
1 .class {  
2   margin: 5px 10px;  
3 }
```

Você também pode definir os espaços individualmente, mas lembre-se sempre da ordem que é: top, right, bottom e left.

```
1 .class {  
2   margin: 0px 5px 10px 15px;  
3 }
```

Display

Basicamente todos os elementos têm um valor padrão da propriedade display. A maioria dos elementos tem seu display configurado em block ou inline. Irei descobrir alguns elementos e seus valores do display.

Block

O elemento block, não aceita elementos na mesma linha que ele, ou seja, quebra a linha após o elemento, e sua área ocupa toda a linha onde ele é inserido.

Alguns elementos que têm como padrão block: <div>, <h1> até <h6>, <p>, <form>, <header>, <footer>, <section>, <table>.

Inline

O elemento inline não inicia em uma nova linha nem quebra de linha após o elemento, pois ele ocupa somente o espaço de seu conteúdo.

Alguns elementos que têm como padrão inline: , <a>, .

Position

A propriedade position específica como um elemento será posicionado na tela, podemos até posicionar em um ponto específico controlando com os parâmetros top, right, bottom e left.

São quatro tipos de posicionamento disponíveis: static, relative, fixed e absolute. A única configuração que não permite escolher um posicionamento para o elemento é static.

- top - Desloca o elemento na vertical (Y), o valor é a distância do elemento com o topo.
- right - Desloca o elemento na horizontal (X), o valor é a distância do elemento com a borda direita.
- bottom - Desloca o elemento na vertical (Y), o valor é a distância do elemento com a borda inferior.
- left - Desloca o elemento na horizontal (X), o valor é a distância do elemento com a borda esquerda.

Static

Este é o valor padrão de todos os elementos HTML, neste posicionamento os elementos não podem ser controlados por top, right, bottom e left, e não tem seu posicionamento afetado pelo posicionamento de outros elementos.

Relative

Um elemento com relative tem seu posicionamento relacionado com o elemento anterior.

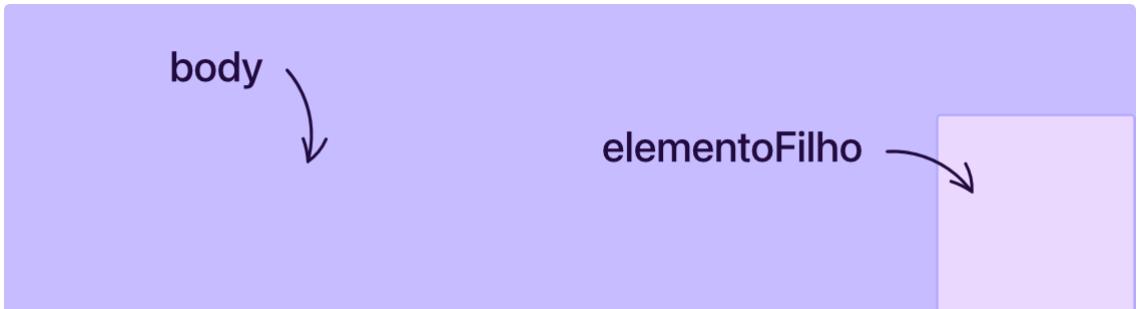
Absolute

Um elemento com absolute tem seu posicionamento relacionado com o elemento pai e não com o elemento anterior, desta maneira elementos anteriores não irão afetar seu posicionamento.

Fixed

O elemento com fixed tem o mesmo comportamento do absolute, só que como o nome já diz, ele fica fixo na tela, isso é, mesmo se acontecer a rolagem ele ficará fixado na página.

Exemplo ilustrativo:



```
1 .elementoFilho {  
2   position: absolute;  
3   bottom: 0;  
4   right: 0;  
5 }
```

Nesse exemplo, o nosso retângulo roxo é o body da nossa página. Por padrão, sempre que colocamos position absolute em um elemento ele irá se referenciar ao seu ancestral mais próximo (que no nosso caso é body). Além do retângulo roxo, temos nosso quadrado verde, ele é o nosso elemento com position absolute, o elemento fica posicionado de forma absoluta em relação ao fluxo do documento, mas de forma relativa ao seu ancestral.

Que tal um exemplo melhor?



```
1 .elementoPai {  
2   position: relative;  
3 }  
4  
5 .elementoFilho {  
6   position: absolute;  
7   bottom: 0;  
8   right: 0;  
9 }
```

Perceba que nosso elementoFilho continua com as mesmas propriedades, porém adicionamos o elementoPai e atribuímos a ele um position relative.

O que isso muda? Muda tudo!

Lembra que eu falei que o position absolute em um elemento ele irá se referenciar ao seu ancestral mais próximo? Foi o que aconteceu! Como o elementoPai é o ancestral mais próximo do elementoFilho, obrigatoriamente o elementoFilho irá se referir a ele.

"E como ficou nosso HTML?"

Perceba que nossa classe elementoFilho está dentro da classe elementoPai, por isso que ele se referenciou a ele.

```
1 <body>
2   <div class="elementoPai">
3     <div class="elementoFilho"></div>
4   </div>
5 </body>
```

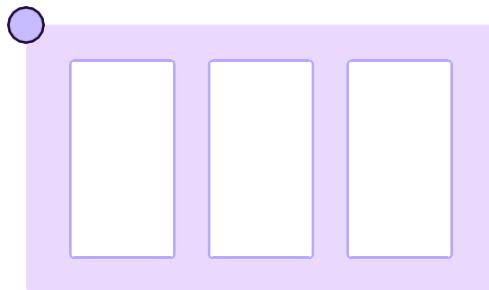
"Okay Iuri, quer dizer que se o elementoFilho ficar fora do elementoPai ele irá se referenciar ao body?"

Sim, isso mesmo!

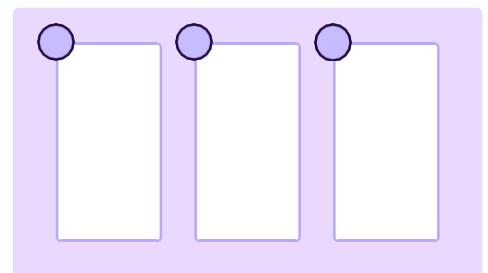
Flexbox

Flexbox são regras que configuram o alinhamento, posicionamento e distribuição de elementos através de regras em uma caixa pai, responsável por organizar o conteúdo dentro do espaço que ela possui. Facilitando, assim, a implementação de sites responsivos.

Container pai



Elementos filhos



Dá aos filhos

- Direção
- Justificação
- Alinhamento

Podem sozinhos

- Se ordenar
- Se alinhar individualmente

justify-content

A propriedade justify-content define como o navegador distribui o espaço entre e ao redor dos itens ao longo do eixo principal de um container flexível. A propriedade só terá efeito se a mesma tiver com a propriedade display flex.

```
1 .elementoPai {  
2   display: flex;  
3   justify-content: flex-start;  
4 }
```



display flex e
justify-content ↑

flex-start (padrão)

Os itens são alinhados no começo do eixo principal. Como pode ver no exemplo anterior colocamos "justify-content: flex-start;" no elemento pai.



flex-end

Os itens são alinhados no final do eixo principal.



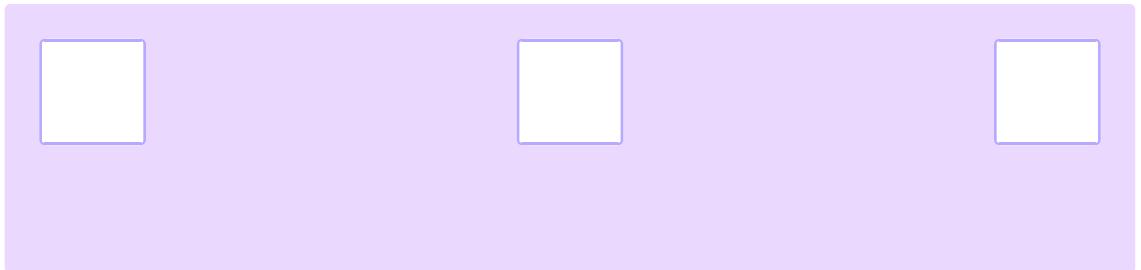
flex-center

Os itens são alinhados no meio do eixo principal.



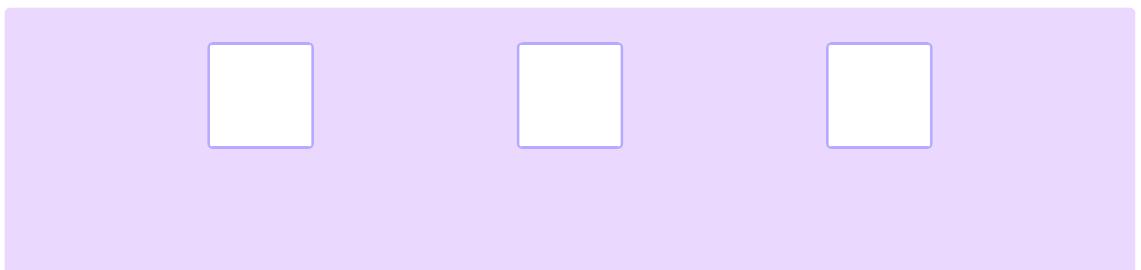
space-between

O primeiro item fica no começo do eixo principal e o último no fim, os restantes ficam alinhados entre si no meio.



space-around

Pega todo o espaço que sobrar na linha e distribui igualmente entre os elementos, deixando o espaço igual em toda a linha do eixo principal.



space-evenly

O espaço é distribuído igualmente entre cada elemento, ou seja, um ao lado do outro, com exceção do primeiro e último do eixo principal.



align-items

Nós usamos o justify-content para alinhar os itens no container, que neste caso é o eixo indo horizontalmente. Para centralizar verticalmente nós usamos a propriedade align-items, para alinhar nossos itens no eixo transversal. Tenha em mente que se o eixo principal mudar o eixo transversal muda com ele.

```
1 .elementoPai {  
2   display: flex;  
3   align-items: flex-start;  
4 }
```

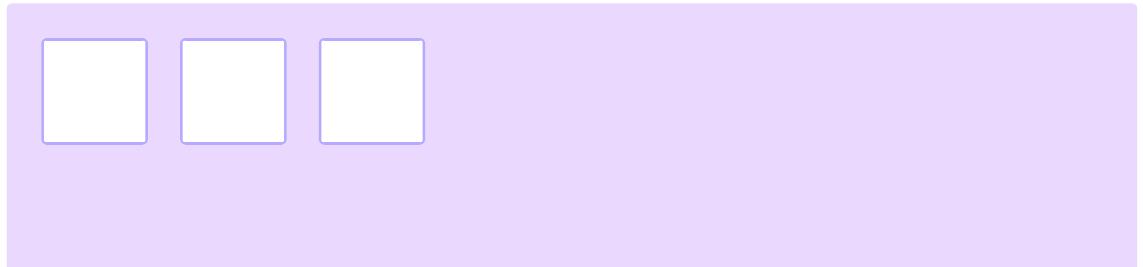


display flex e
align-items



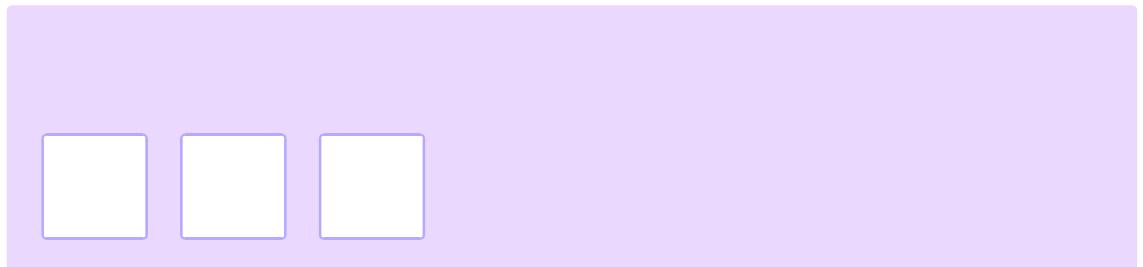
flex-start

Os itens são postos no começo do eixo transversal.



flex-end

Os itens são alocados no final do eixo transversal.



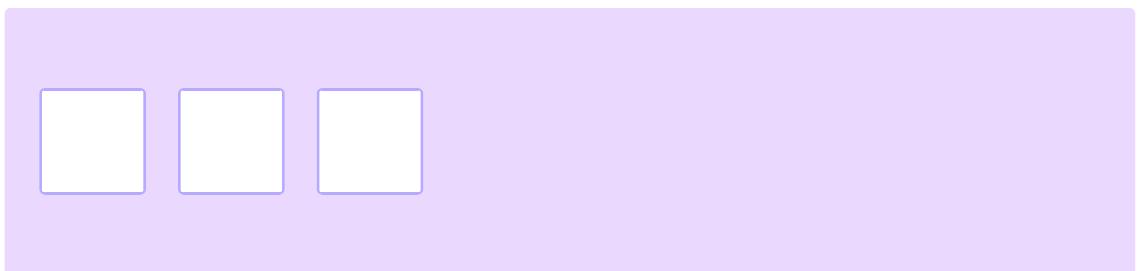
stretch

Os itens se estenderão por todo o eixo.



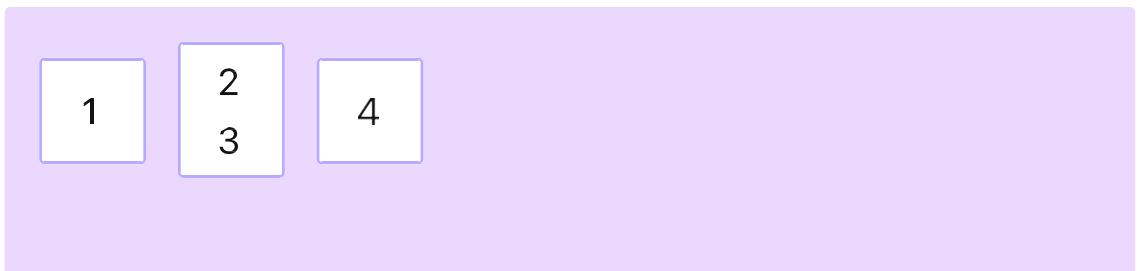
center

Os itens são alinhados no meio do eixo.



baseline

Alinha os itens de acordo com a linha base da tipografia.

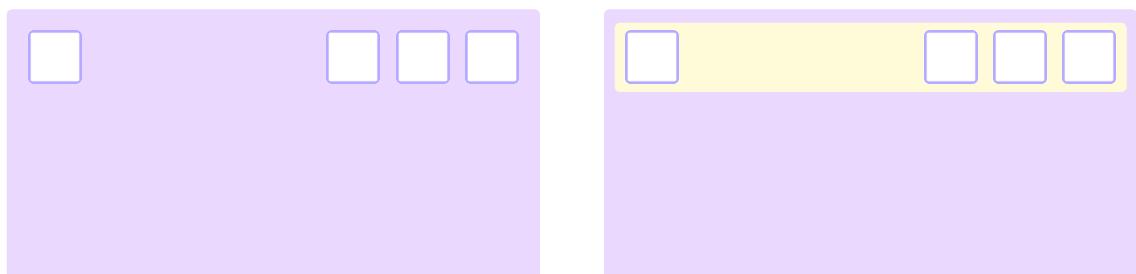


Outras propriedades

Existem outras propriedades do flexbox como flex-basis, flex-direction, flex-grow, flex-shrink, flex-wrap, order, gap, align-self e entre outros. Caso você queira ver outras propriedades, clique [aqui](#) e leia mais.

Posicionamento com flexbox

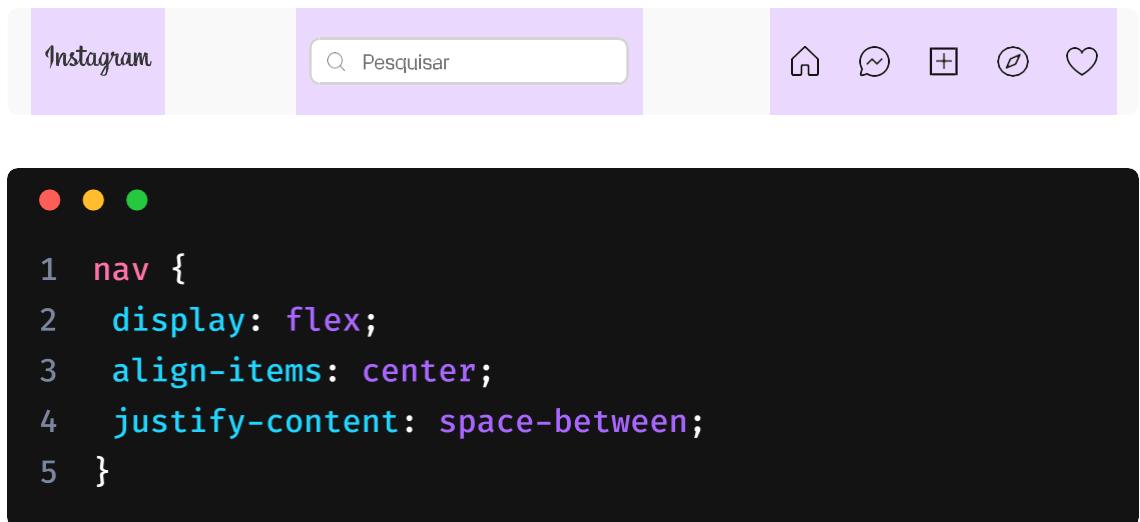
Quando falamos em posicionamento, primeiro temos que pensar qual vai ser a estrutura HTML e assim conseguimos trabalhar com flexbox.



A estrutura acima segue o mesmo padrão de um menu, certo? Perceba a forma que separei a estrutura dele.

A parte em amarelo é o nosso menu, dentro do menu temos duas coisas, um quadrado na esquerda e um grupo de quadrados na direita. Para posicionar eles precisamos agrupar esses quadrados na direita pois, caso isso não ocorra, seria difícil posicionar esses elementos.

Veremos esse exemplo em uma aplicação real.



```
1 nav {  
2   display: flex;  
3   align-items: center;  
4   justify-content: space-between;  
5 }
```

Para replicar esse menu do Instagram precisaríamos de quatro tags HTML para posicionar os elementos.

É claro que isso leva a outros fatores como tamanhos, alturas, espaçamentos e entre outras coisas. Mas se você está utilizando flexbox para criar uma estrutura e posicionar elementos, essa sempre vai ser a base.

Responsividade

Sites responsivos são aqueles que adaptam o tamanho das suas páginas ao tamanho das telas que estão sendo exibidos, como as telas de celulares, tablets e televisões (como a Netflix e o YouTube).

“Qual a forma de aplicar?”

Temos algumas formas de aplicar responsividade em nossos sites, as mais utilizadas são com o uso de frameworks e media queries. Se você está fazendo um site e tem um prazo curto para entregar ele, recomendo o uso de frameworks pois já está tudo pronto para um layout responsivo.

“Quero deixar responsivo com media queries, como eu faço?”

A responsividade é um estilo, então o media queries será aplicado em sua folha de estilo CSS. Mas antes temos que entender que temos alguns tipos de media, eles são chamados de media types.

Os types definem para que tipo de media um certo código CSS será aplicado.

Media type

Os types mais utilizados são:

- all: usado para todos os dispositivos de tipo de mídia.
- print: usado para impressoras.
- screen: usado para telas de computador, tablets, smartphones e etc.
- speech: usado para leitores de tela que “leem” a página em voz alta.

"Mas como eu faço para aplicar?"

O exemplo a seguir altera a cor de fundo da página para roxo se a janela de visualização tiver 480 pixels de largura ou menos.

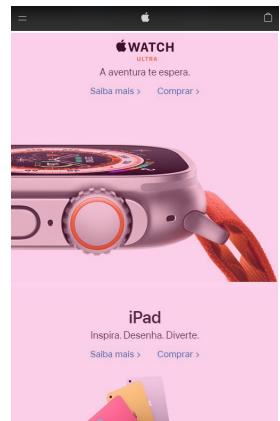
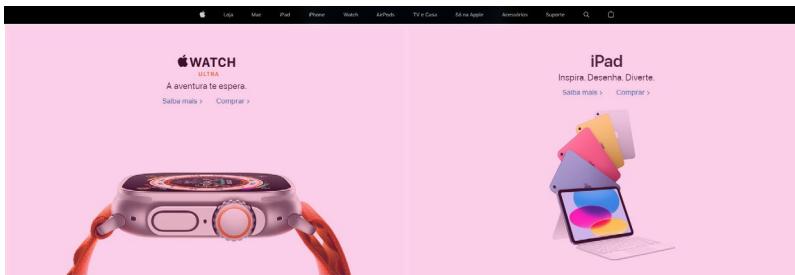
```
1 @media screen and (max-width: 480px) {  
2   body {  
3     background-color: #00ff00;  
4   }  
5 }
```

Sempre que vamos fazer algo no media queries, temos antes que dizer que é para um media que estamos desenvolvendo para isso colocamos @media. Logo em seguida o tipo dessa media (no nosso caso foi o screen) e um operador lógico (no nosso caso foi o and).

Entre os parênteses passamos o tamanho da largura que aquele media irá ser aplicado.

Perceba que usamos o max, nesse caso estamos dizendo que será aplicado quando tiver 480 pixels de largura ou menos.

Vamos usar o site da Apple como um exemplo real de responsividade.



Veja como o site da Apple aplica responsividade em suas páginas. A grande sacada foi aumentar o tamanho dos card para quando chegar em um determinado tamanho para dispositivos mobile.

```
1 @media screen and (max-width: 480px) {  
2   .card {  
3     width: 100%;  
4   }  
5 }
```

Uma coisa super importante é que você não precisa escrever todas as propriedades de uma class ou id dentro do seu @media, somente o que você quer que mude. Que no caso o tamanho de cada card terá o tamanho da resolução do dispositivo.

Observação: caso seja feito com flexbox você precisa colocar a propriedade `flex-wrap: wrap;`. Ela define se os itens ficam na mesma linha ou se podem ser quebrados em várias linhas. Que no caso dos card da Apple os card quebraram a linha e foram para baixo.

Padrões de resoluções

- Desktops: (`min-width: 1281px`)
- Laptops, desktops: (`min-width: 1025px`) and (`max-width: 1280px`)
- Tablets, ipads: (`min-width: 768px`) and (`max-width: 1024px`)
- Tablets de baixa resolução, celulares: (`min-width: 481px`) and (`max-width: 767px`)
- A maioria dos smartphones móveis: (`min-width: 320px`) and (`max-width: 480px`)

Fonte: <https://gist.github.com/janily/8453473>

CSS Grid

O CSS Grid é um sistema de estruturação de layout. A diferença dele pro Flexbox é que ele nos permite configurar layouts em duas dimensões (linhas e colunas) sendo que no Flexbox é apenas um dimensão.

Grid Container

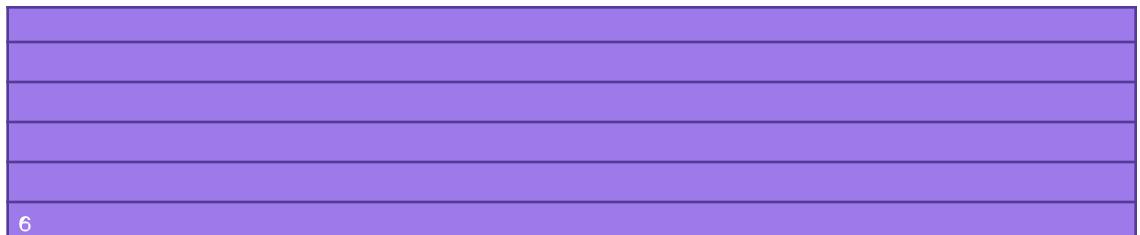
Para criar um grid container utilizamos a propriedade `display: grid` (ou `display: inline-grid` para tornar o elemento um grid container porém com comportamento inline). Assim declarando, todos os elementos filhos diretos daquele container se transformam em grid items.

```
● ● ●  
1 .container {  
2   display: grid;  
3 }
```

```
● ● ●  
1 <div class="container">  
2   <div class="item">1</div>  
3   <div class="item">2</div>  
4   <div class="item">3</div>  
5   <div class="item">4</div>  
6 </div>
```

Isso irá criar uma grid do tipo block, significando que o `.container` irá ocupar a linha inteira.

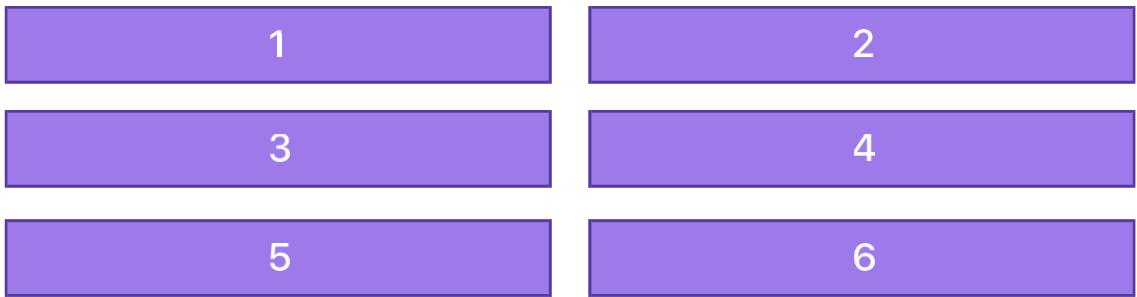
Uma coisa interessante que podemos utilizar algumas propriedade do Flexbox no CSS Grid, como justify-items, align-items, justify-content, align-content. Nesse caso, iremos utilizar e adicionar o gap de 16px entre os elementos filhos (item). Mas podemos utilizar as propriedade grid-column-gap, grid-row-gap do CSS Grid para criar espaçamentos em lugares mais específicos.



grid-template-columns

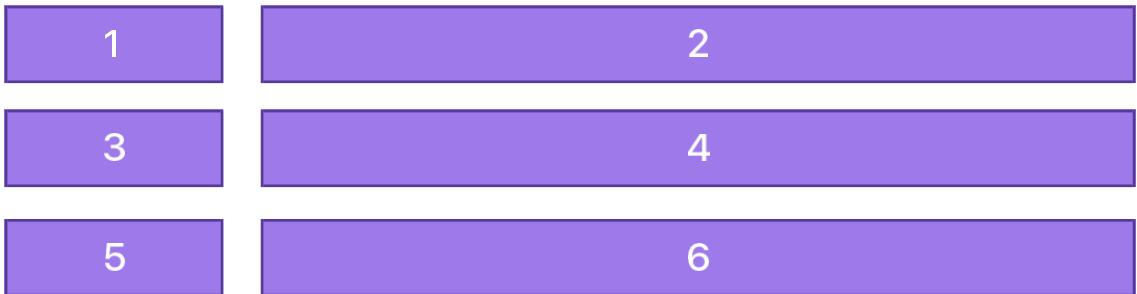
Para criar colunas, usaremos a propriedade grid-template-columns, que recebe os valores de tamanho de cada coluna separados por espaço. Iremos, por exemplo, criar duas colunas com 200px cada.

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 200px 200px;  
4 }
```



Mas também podemos criar um layout em que a primeira coluna ocupa um espaço fixo e a segunda ocupa todo o resto do espaço disponível. Isto é possível usando a unidade fr, introduzida exatamente para ser utilizada como valor de tamanho para linhas e colunas em container.

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 200px 1fr;  
4 }
```

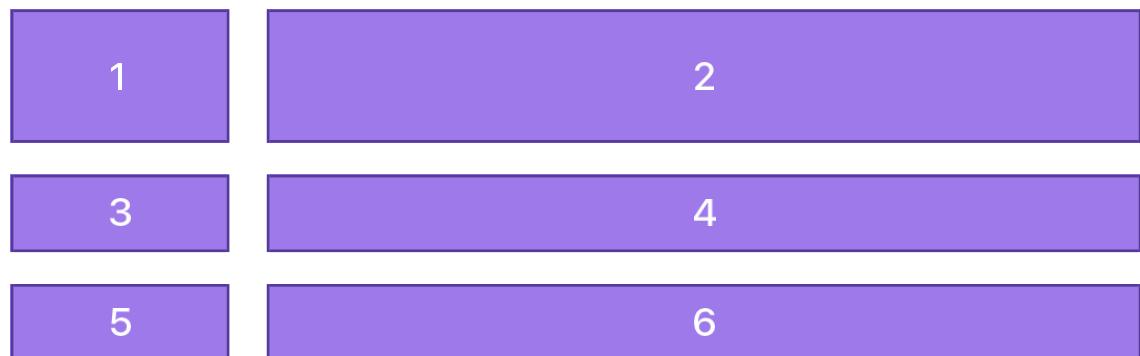


A unidade `fr` representa uma fração do espaço livre disponível em uma linha ou coluna. Podemos misturar diversos valores `fr` para formar layouts automáticos que respeitam certas proporções.

grid-template-rows

Exatamente da mesma maneira nós declaramos as linhas utilizando a propriedade `grid-template-rows`.

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 200px 1fr;  
4   grid-template-rows: 200px 100px;  
5 }
```



Dentro do nosso container agora teremos 2 linhas declaradas com os valores de 200px e 100px. Assim como nossas colunas, o tamanho das linhas preenchem o espaço disponível proporcionalmente.

grid-template-areas

Você deve ter notado que nossos itens vão se organizando na grid de acordo com a ordem em que estão no HTML.

Porém, podemos dar nomes para cada área da nossa grade e depois indicar onde cada elemento deve ir.

Vamos alterar o nosso HTML. iremos colocar as classes header, sidenav, main e footer nos itens da grid para podermos indicar a posição de cada um deles depois.

```
● ● ●  
1 <div class="container">  
2   <div class="item header">Header</div>  
3   <div class="item sidenav">Sidenav</div>  
4   <div class="item main">Main</div>  
5   <div class="item footer">Footer</div>  
6 </div>
```

Para cada item, vamos declarar a propriedade grid-area, que serve para indicar o nome da área em que cada elemento deverá ocupar na grid.

```
● ● ●  
1 .header {  
2   grid-area: header;  
3 }  
4 .sidenav {  
5   grid-area: sidenav;  
6 }  
7 .content {  
8   grid-area: main;  
9 }  
10 .footer {  
11   grid-area: footer;  
12 }
```

Para dar nomes às áreas de uma mesma linha, escrevemos dentro de "" (aspas). Ao abrir novas aspas estaremos indicando que estamos indo para a linha seguinte.

```
1 .container {  
2   grid-template-areas: "header header"  
3                         "sidenav main"  
4                         "footer footer";  
5 }
```

No código acima, repetimos header e footer duas vezes porque declaramos duas colunas e queremos que eles ocupem as duas.

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 30% 60%;  
4   grid-template-rows: 30px 80px 20px;  
5   grid-template-areas: "header header"  
6                         "sidenav main"  
7                         "footer footer";  
8 }
```

Header

Aside

Main

Footer

grid-auto-rows

Mas temos um problema. Nem sempre todo o conteúdo de um container grid será previsível. E se tivéssemos uma lista dinâmica? Nossos itens irão se comportar dessa forma:

Um

Dois

Três

Quatro

Cinco

Seis

Sete

Os elementos “Cinco”, “Seis” e “Sete” continuam organizados em colunas como os anteriores, mas eles já não seguem os mesmos tamanhos das linhas. Isso acontece porque quando temos mais elementos do que células no nosso container grid, os elementos que sobram criam novas células implícitas que ocupam as colunas disponíveis e criam novas linhas que não tinham sido declaradas.

Para controlar o grid explícito, podemos a propriedade `grid-auto-rows` para definir como o nosso grid implícito será disposto.

```
1 .container {  
2   display: grid;  
3   gap: 10px;  
4  
5   /* grid explícito */  
6   grid-template-columns: 2fr 1fr;  
7   grid-template-rows: 100px 100px 100px;  
8  
9   /* grid implícito */  
10  grid-auto-rows: 100px;  
11 }
```

Um

Dois

Três

Quatro

Cinco

Seis

Sete

Com o uso de grid-auto-rows: 100px definimos agora que todas as novas linhas terão 100px de tamanho. Como a propriedade grid-auto-rows aceita muito mais do que apenas um valor, podemos apresentar nossa lista dinâmica.