



# O Manual CSS

## Índice

- [Índice](#)
- [1. Prefácio](#)
- [2. Introdução ao CSS](#)
- [2.1. Como é o CSS](#)
- [2.2. Ponto e vírgula](#)
- [2.3. Formatação e recuo](#)
- [3. Uma breve história do CSS](#)
- [4. Adicionando CSS a uma página HTML](#)
- [4.1. 1: Usando a tag de link](#)
- [4.2. 2: usando a tag de estilo](#)
- [4.3. 3: estilos em linha](#)
- [5. Seletores](#)
- [5.1. Seletores básicos](#)
- [5.2. Combinação de seletores](#)
- [5.2.1. Segmentação de um elemento com uma classe ou id](#)
- [5.2.2. Segmentação de várias classes](#)
- [5.2.3. Combinando classes e ids](#)
- [5.3. Seletores de agrupamento](#)
- [5.4. Siga a árvore de documentos com seletores](#)
- [6. Cascata](#)
- [7. Especificidade](#)
- [7.1. Como calcular a especificidade](#)
- [7.1.1. Ranhura 1](#)
- [7.1.2. Ranhura 2](#)
- [7.1.3. Ranhura 3](#)
- [7.1.4. Ranhura 4](#)
- [7.2. Importância](#)



What would you do  
with 33% less  
development time?  
Sign up free and save  
time with Auth0.

ADS VIA CARBON

- [7.3. Dicas](#)
- [7.4. Ferramentas para calcular a especificidade](#)
- [8. Herança](#)
- [8.1. Propriedades que herdam](#)
- [8.2. Forçando as propriedades a herdar](#)
- [8.3. Forçando as propriedades a NÃO herdar](#)
- [8.4. Outros valores especiais](#)
- [9. Importação](#)
- [10. Seletores de atributos](#)
  - [10.1. Seletores de presença de atributo](#)
  - [10.2. Seletores de valor de atributo exatos](#)
  - [10.3. Correspondêr a uma parte do valor do atributo](#)
- [11. Pseudoclasses](#)
- [12. Pseudo-elementos](#)
- [13. Cores](#)
  - [13.1. Cores nomeadas](#)
  - [13.2. RGB e RGBa](#)
  - [13.3. Notação hexadecimal](#)
  - [13.4. HSL e HSLa](#)
- [14. Unidades](#)
  - [14.1. Pixels](#)
  - [14.2. Percentagens](#)
  - [14.3. Unidades de medida do mundo real](#)
  - [14.4. Unidades relativas](#)
  - [14.5. Unidades de visor](#)
  - [14.6. Unidades fracionárias](#)
- [15. url\(\)](#)
- [16. calc\(\)](#)
- [17. Antecedentes](#)
- [18. Comentários](#)
- [19. Propriedades personalizadas](#)
  - [19.1. Noções básicas de utilização de variáveis](#)
  - [19.2. Criar variáveis dentro de qualquer elemento](#)
  - [19.3. Âmbito das variáveis](#)
  - [19.4. Interagindo com um valor de variável CSS usando JavaScript](#)
  - [19.5. Manipulação de valores inválidos](#)
  - [19.6. Suporte do navegador](#)
  - [19.7. CSS As variáveis diferenciam maiúsculas de minúsculas](#)
  - [19.8. Matemática em Variáveis CSS](#)

- [19.9. Consultas de mídia com variáveis CSS](#)
- [19.10. Definindo um valor de fallback para var\(\)](#)
- [20. Fontes](#)
- [20.1. família de fontes](#)
- [20.2. Peso da fonte](#)
- [20.3. font-stretch](#)
- [20.4. estilo de fonte](#)
- [20.5. tamanho da fonte](#)
- [20.6. Variante de fonte](#)
- [20.7. fonte](#)
- [20.8. Carregando fontes personalizadas usando @font-face](#)
- [20.9. Uma nota sobre o desempenho](#)
- [21. Typography](#)
- [21.1. text-transform](#)
- [21.2. text-decoration](#)
- [21.3. text-align](#)
- [21.4. vertical-align](#)
- [21.5. line-height](#)
- [21.6. text-indent](#)
- [21.7. text-align-last](#)
- [21.8. word-spacing](#)
- [21.9. letter-spacing](#)
- [21.10. text-shadow](#)
- [21.11. white-space](#)
- [21.12. tab-size](#)
- [21.13. writing-mode](#)
- [21.14. hyphens](#)
- [21.15. text-orientation](#)
- [21.16. direction](#)
- [21.17. word-break](#)
- [21.18. overflow-wrap](#)
- [22. Box Model](#)
- [23. Border](#)
- [23.1. The border style](#)
- [23.2. The border width](#)
- [23.3. The border color](#)
- [23.4. The border shorthand property](#)

- [23.5. The border radius](#)
- [23.6. Using images as borders](#)
- [24. Padding](#)
- [24.1. Specific padding properties](#)
- [24.2. Using the padding shorthand](#)
- [24.2.1. 1 value](#)
- [24.2.2. 2 values](#)
- [24.2.3. 3 values](#)
- [24.2.4. 4 values](#)
- [24.3. Values accepted](#)
- [25. Margin](#)
- [25.1. Specific margin properties](#)
- [25.2. Using the margin shorthand](#)
- [25.2.1. 1 value](#)
- [25.2.2. 2 values](#)
- [25.2.3. 3 values](#)
- [25.2.4. 4 values](#)
- [25.3. Values accepted](#)
- [25.4. Using auto to center elements](#)
- [25.5. Using a negative margin](#)
- [26. Box Sizing](#)
- [27. Display](#)
- [27.1. inline](#)
- [27.2. inline-block](#)
- [27.3. block](#)
- [27.4. none](#)
- [28. Positioning](#)
- [28.1. Static positioning](#)
- [28.2. Relative positioning](#)
- [28.3. Absolute positioning](#)
- [28.4. Fixed positioning](#)
- [28.5. Sticky positioning](#)
- [29. Floating and clearing](#)
- [29.1. Clearing](#)
- [30. z-index](#)
- [31. CSS Grid](#)
- [31.1. The basics](#)
- [31.1.1. grid-template-columns and grid-template-rows](#)

- [31.1.2. Automatic dimensions](#)
- [31.1.3. Different columns and rows dimensions](#)
- [31.1.4. Adding space between the cells](#)
- [31.1.5. Spawning items on multiple columns and/or rows](#)
- [31.1.5.1. Shorthand syntax](#)
- [31.2. More grid configuration](#)
- [31.2.1. Using fractions](#)
- [31.2.2. Using percentages and rem](#)
- [31.2.3. Using repeat\(.\)](#)
- [31.2.4. Specify a minimum width for a row](#)
- [31.2.5. Positioning elements using grid-template-areas](#)
- [31.2.5.1. Adicionando células vazias em áreas de modelo](#)
- [31.3. Preencher uma página com uma grade](#)
- [31.4. Um exemplo: cabeçalho, barra lateral, conteúdo e rodapé](#)
- [31.5. Conclusão](#)
- [32. Caixa flexível](#)
- [32.1. Suporte do navegador](#)
- [32.2. Ativar o Flexbox](#)
- [32.3. Propriedades do contêiner](#)
- [32.3.1. Alinhar linhas ou colunas](#)
- [32.3.2. Alinhamento vertical e horizontal](#)
- [32.3.2.1. Alterar o alinhamento horizontal](#)
- [32.3.2.2. Alterar o alinhamento vertical](#)
- [32.3.2.3. Uma nota sobre a linha de base](#)
- [32.3.3. Envolver](#)
- [32.4. Propriedades que se aplicam a cada item único](#)
- [32.4.1. Movendo itens antes / depois de outro usando a ordem](#)
- [32.4.2. Alinhamento vertical utilizando align-self](#)
- [32.4.3. Aumentar ou encolher um item, se necessário](#)
- [32.4.3.1. flex-grow](#)
- [32.4.3.2. flex-shrink](#)
- [32.4.3.3. base flexível](#)
- [32.4.3.4. flex](#)
- [33. Tabelas](#)
- [34. Centralização](#)
- [34.1. Centralizar horizontalmente](#)
- [34.1.1. Texto](#)
- [34.1.2. Blocos](#)
- [34.2. Centralizar verticalmente](#)

- [34.3. Centralize vertical e horizontalmente](#)
- [35. Listas](#)
- [36. Consultas de mídia e design responsivo](#)
- [36.1. Tipos de mídia](#)
- [36.2. Descritores de recursos de mídia](#)
- [36.3. Operadores lógicos](#)
- [36.4. Consultas de mídia](#)
- [37. Consultas de recursos](#)
- [38. Filtros](#)
- [38.0.1. blur\(\)](#)
- [38.0.2. opacidade\(\)](#)
- [38.0.3. sombra\(\)](#)
- [38.0.4. escala de cinzento\(\)](#)
- [38.0.5. sépia\(\)](#)
- [38.0.6. invert\(\)](#)
- [38.0.7. matiz-rotação\(\)](#)
- [38.0.8. brilho\(\)](#)
- [38.0.9. contraste\(\)](#)
- [38.0.10. saturar\(\)](#)
- [38.0.11. url\(\)](#)
- [39. Transforma](#)
- [39.1. Transformações 2D](#)
- [39.2. Combinando múltiplas transformações](#)
- [39.3. Transformações 3D](#)
- [40. Transições](#)
- [40.1. Exemplo de transição CSS](#)
- [40.2. Valores da função de temporização de transição](#)
- [40.3. CSS Transições no DevTools do navegador](#)
- [40.4. Quais propriedades você pode animar usando animações CSS](#)
- [41. Animações](#)
- [41.1. Um exemplo de animações CSS](#)
- [41.2. As propriedades de animação CSS](#)
- [41.3. Eventos JavaScript para animações CSS](#)
- [41.4. Quais propriedades você pode animar usando animações CSS](#)
- [42. Normalizando o CSS](#)
- [43. Tratamento de erros](#)
- [44. Prefixos do fornecedor](#)
- [45. CSS para impressão](#)

- [45.1. Imprimir CSS](#)
- [45.2. CSS @media impressão](#)
- [45.3. Ligações](#)
- [45.4. Margens da página](#)
- [45.5. Quebras de página](#)
- [45.6. Evite quebrar imagens no meio](#)
- [45.7. Depurar a apresentação de impressão](#)

## 1. Prefácio

Eu escrevi este livro para ajudá-lo a aprender rapidamente CSS e se familiarizar com os tópicos avançados de CSS.

CSS, uma abreviação de Cascading Style Sheets, é um dos principais blocos de construção da Web. Sua história remonta aos anos 90 e, juntamente com o HTML, mudou muito desde seus humildes primórdios.

Tendo criado sites desde antes do CSS existir, tenho visto a sua evolução.

O CSS é uma ferramenta incrível e, nos últimos anos, cresceu muito, introduzindo muitos recursos fantásticos como CSS Grid, Flexbox e CSS Custom Properties.

Este manual destina-se a um vasto público.

Primeiro, o iniciante. Eu explico CSS do zero de uma forma sucinta, mas abrangente, para que você possa usar este livro para aprender CSS a partir do básico.

Depois, o profissional. CSS é muitas vezes considerado como uma coisa secundária para aprender, especialmente por desenvolvedores JavaScript. Eles sabem que CSS não é uma linguagem de programação real, eles são programadores e, portanto, eles não devem se preocupar em aprender CSS da maneira certa. Eu escrevi este livro para você também.

Em seguida, a pessoa que conhece CSS há alguns anos, mas não teve a oportunidade de aprender as coisas novas nele. Falaremos extensivamente sobre os novos recursos do CSS, aqueles que vão construir a web da próxima década.

O CSS melhorou muito nos últimos anos e está evoluindo rapidamente.

Mesmo que você não escreva CSS para ganhar a vida, saber como o CSS funciona pode ajudar a economizar algumas dores de cabeça quando você precisa entendê-lo de tempos em tempos, por exemplo, ao ajustar uma página da web.

**CSS** (uma abreviação de **Cascading Style Sheets**) é a linguagem que usamos para estilizar um arquivo HTML e informar ao navegador como ele deve renderizar os elementos na página.

Neste livro eu falo exclusivamente sobre o estilo de documentos HTML, embora o CSS possa ser usado para estilizar outras coisas também.

Um arquivo CSS contém várias regras CSS.

Cada regra é composta por 2 partes:

- o **seletor**
- o **bloco de declaração**

O seletor é uma cadeia de caracteres que identifica um ou mais elementos na página, seguindo uma sintaxe especial sobre a qual falaremos extensivamente em breve.

O bloco de declaração contém uma ou mais **declarações**, por sua vez compostas por um par de **propriedades** e **valores**.

Essas são todas as coisas que temos em CSS.

Organizar cuidadosamente as propriedades, associá-las a valores e anexá-las a elementos específicos da página usando um seletor é todo o argumento deste ebook.

## 2. Introdução ao CSS

### 2.1. Como é o CSS

Um **conjunto de regras** CSS tem uma parte chamada **seletor** e a outra parte chamada **declaração**. A declaração contém várias **regras**, cada uma composta por uma **propriedade** e um **valor**.

Neste exemplo, é o seletor e aplica uma regra que define o valor para a propriedade: `p { font-size: 20px; }`

```
p {  
    font-size: 20px;  
}
```

Várias regras são empilhadas uma após a outra:

```
p {  
    font-size: 20px;  
}  
  
a {  
    color: blue;  
}
```

Um seletor pode segmentar um ou mais itens:

```
p, a {  
    font-size: 20px;  
}
```

e pode direcionar marcas HTML, como acima, ou elementos HTML que contêm um determinado atributo de classe com `.my-class`, ou elementos HTML que têm um atributo específico com `..my-classid#my-id`

Seletores mais avançados permitem que você escolha itens cujo atributo corresponda a um valor específico ou também itens que respondem a pseudoclasses (mais sobre isso mais tarde)

## 2.2. Ponto e vírgula

Cada regra CSS termina com um ponto-e-vírgula. Os ponto-e-vírgula **não** são opcionais, exceto após a última regra, mas sugiro sempre usá-los para consistência e evitar erros se você adicionar outra propriedade e esquecer de adicionar o ponto-e-vírgula na linha anterior.

## 2.3. Formatação e recuo

Não há uma regra fixa para formatação. Este CSS é válido:

```
p {  
    font-size: 20px;  
  
    a {color: blue;  
}
```

mas uma dor de ver. Atenha-se a algumas convenções, como as que você vê nos exemplos acima: fixe seletores e os colchetes de fechamento à esquerda, recue 2 espaços para cada regra, tenha o colchete de abertura na mesma linha do seletor, separados por um espaço.

O uso correto e consistente de espaçamento e recuo é uma ajuda visual na compreensão do seu código.

## 3. Uma breve história do CSS

Antes de prosseguir, quero dar-lhe uma breve recapitulação da história do CSS.

CSS foi criado a partir da necessidade de estilizar páginas da web. Antes do CSS ser introduzido, as pessoas queriam uma maneira de estilizar suas páginas da web, que pareciam muito semelhantes e "acadêmicas" no passado. Você não poderia fazer muito em termos de personalização.

O HTML 3.2 introduziu a opção de definir cores embutidas como atributos de elementos HTML e tags de apresentação como `e`, mas isso se transformou rapidamente em uma situação longe de ser ideal.

O CSS nos permite mover tudo relacionado à apresentação do HTML para o CSS, para que o HTML possa voltar a ser o formato que define a estrutura do documento, em vez de como as coisas devem parecer no navegador.

O CSS está em constante evolução, e o CSS que você usou há 5 anos pode estar desatualizado, à medida que novas técnicas idiomáticas de CSS surgiram e os navegadores mudaram.

É difícil imaginar os tempos em que o CSS nasceu e como a web era diferente.

Na época, tínhamos vários navegadores concorrentes, sendo os principais o Internet Explorer ou o Netscape Navigator.

As páginas foram estilizadas usando HTML, com tags especiais de apresentação e atributos especiais, a maioria dos quais agora está obsoleta.**bold**

Isso significava que você tinha uma quantidade limitada de oportunidades de personalização.

A maior parte das decisões de estilo foram deixadas para o navegador.

Além disso, você criou um site especificamente para um deles, porque cada um introduziu diferentes tags não padronizadas para dar mais poder e oportunidades.

Logo as pessoas perceberam a necessidade de uma maneira de estilizar as páginas, de uma maneira que funcionasse em todos os navegadores.

Após a ideia inicial proposta em 1994, o CSS teve seu primeiro lançamento em 1996, quando a recomendação CSS Nível 1 ("CSS 1") foi publicada.

CSS Nível 2 ("CSS 2") foi publicado em 1998.

Desde então, o trabalho começou no CSS Nível 3. O Grupo de Trabalho CSS decidiu dividir cada recurso e trabalhar nele separadamente, em módulos.

Os navegadores não eram especialmente rápidos na implementação de CSS. Tivemos que esperar até 2002 para que o primeiro navegador

implementasse a especificação CSS completa: IE para Mac, como bem descrito neste post CSS Tricks: <https://css-tricks.com/look-back-history-css/>

O Internet Explorer implementou o modelo de caixa incorretamente desde o início, o que levou a anos de dor tentando ter o mesmo estilo aplicado consistentemente em todos os navegadores. Tivemos que usar vários truques e hacks para fazer com que os navegadores renderizassem as coisas como queríamos.

Hoje as coisas estão muito, muito melhores. Podemos simplesmente usar os padrões CSS sem pensar em peculiaridades, na maioria das vezes, e o CSS nunca foi tão poderoso.

Não temos mais números oficiais de lançamento para CSS agora, mas o CSS Working Group lança um "instantâneo" dos módulos que atualmente são considerados estáveis e prontos para serem incluídos em navegadores. Este é o último instantâneo, de 2018: <https://www.w3.org/TR/css-2018/>

CSS Nível 2 ainda é a base para o CSS que escrevemos hoje, e temos muitos recursos construídos em cima dele.

## 4. Adicionando CSS a uma página HTML

CSS é anexado a uma página HTML de diferentes maneiras.

### 4.1. 1: Usando a tag `link`

A tag é a maneira de incluir um arquivo CSS. Essa é a maneira preferida de usar o CSS como ele deve ser usado: um arquivo CSS é incluído por todas as páginas do seu site, e a alteração de uma linha nesse arquivo afeta a apresentação de todas as páginas do site.

Para usar esse método, adicione uma marca com o atributo apontando para o arquivo CSS que deseja incluir. Você o adiciona dentro da tag do site (não dentro da tag):

```
<link rel="stylesheet" type="text/css" href="myfile.css" />
```

Os atributos e também são necessários, pois informam ao navegador para qual tipo de arquivo estamos vinculando.`rel="stylesheet"`

## 4.2. 2: utilização da etiqueta `style`

Em vez de usar a tag para apontar para a folha de estilo separada contendo nosso CSS, podemos adicionar o CSS diretamente dentro de uma tag. Esta é a sintaxe:`style`

```
<style>
  ...our CSS... ;
</style>
```

Usando este método, podemos evitar a criação de um arquivo CSS separado. Acho que esta é uma boa maneira de experimentar antes de "formalizar" CSS para um arquivo separado, ou para adicionar uma linha especial de CSS apenas para um arquivo.

## 4.3. 3: estilos em linha

Os estilos embutidos são a terceira maneira de adicionar CSS a uma página. Podemos adicionar um atributo a qualquer tag HTML e adicionar CSS a ele.`style`

```
<div style="">...</div>
```

Exemplo:

```
<div style="background-color: yellow">...</div>
```

# 5. Seletores

Um seletor nos permite associar uma ou mais declarações a um ou mais elementos na página.

## 5.1. Seletores básicos

Suponha que temos um elemento na página e queremos exibir as palavras nele usando a cor amarela.`p`

Podemos segmentar esse elemento usando este seletor, que **segmenta** todo o elemento usando a tag na página. Uma regra CSS simples para alcançar o que queremos é:

```
p {  
    color: yellow;  
}
```

Cada tag HTML tem um seletor correspondente, por exemplo: `,` `,` `.` `div` `span` `img`

Se um seletor corresponder a vários elementos, todos os elementos da página serão afetados pela alteração.

Os elementos HTML têm 2 atributos que são muito comumente usados no CSS para associar o estilo a um elemento específico na página: `e` `.class` `id`

Há uma grande diferença entre esses dois: dentro de um documento HTML, você pode repetir o mesmo valor em vários elementos, mas só pode usar uma vez. Como corolário, usando classes você pode selecionar um elemento com 2 ou mais nomes de classe específicos, algo que não é possível usando ids.

As classes são identificadas usando o símbolo `.`, enquanto as ids usam o símbolo `#`

Exemplo usando uma classe:

```
<p class="dog-name">Roger</p>
```

```
.dog-name {  
    color: yellow;  
}
```

Exemplo usando uma id:

```
<p id="dog-name">Roger</p>
```

```
#dog-name {  
    color: yellow;  
}
```

## 5.2. Combinação de seletores

Até agora, vimos como segmentar um elemento, uma classe ou um id. Vamos introduzir seletores mais poderosos.

### 5.2.1. Segmentação de um elemento com uma classe ou id

Você pode direcionar um elemento específico que tenha uma classe, ou id, anexada.

Exemplo usando uma classe:

```
<p class="dog-name">Roger</p>
```

```
p.dog-name {  
    color: yellow;  
}
```

Exemplo usando uma id:

```
<p id="dog-name">Roger</p>
```

```
p#dog-name {  
    color: yellow;  
}
```

Por que você gostaria de fazer isso, se a classe ou id já fornece uma maneira de direcionar esse elemento? Você pode ter que fazer isso para ter mais

especificidade. Veremos o que isso significa mais tarde.

## 5.2.2. Segmentação de várias classes

Você pode segmentar um elemento com uma classe específica usando o , como você viu anteriormente. Você pode direcionar um elemento com 2 (ou mais) classes combinando os nomes de classe separados com um ponto, sem espaços..class-name

Exemplo:

```
<p class="dog-name roger">Roger</p>
```

```
.dog-name.roger {  
    color: yellow;  
}
```

## 5.2.3. Combinando classes e ids

Da mesma forma, você pode combinar uma classe e um id.

Exemplo:

```
<p class="dog-name" id="roger">Roger</p>
```

```
.dog-name#roger {  
    color: yellow;  
}
```

## 5.3. Seletores de agrupamento

Você pode combinar seletores para aplicar as mesmas declarações a vários seletores. Para fazer isso, você os separa com uma vírgula.

Exemplo:

```
<p>My dog name is:</p>
<span class="dog-name"> Roger </span>
```

```
p,
.dog-name {
  color: yellow;
}
```

Você pode adicionar espaços nessas declarações para torná-las mais claras:

```
p,
.dog-name {
  color: yellow;
}
```

## 5.4. Siga a árvore de documentos com seletores

Vimos como segmentar um elemento na página usando um nome de tag, uma classe ou um id.

Você pode criar um seletor mais específico combinando vários itens para seguir a estrutura da árvore de documentos. Por exemplo, se você tiver uma tag aninhada dentro de uma tag, poderá segmentá-la sem aplicar o estilo a uma tag não incluída em uma tag:spanpspanp

```
<span> Hello! </span>
<p>
  My dog name is:
  <span class="dog-name"> Roger </span>
</p>
```

```
p span {
  color: yellow;
}
```

Veja como usamos um espaço entre os dois tokens e .pspan

Isso funciona mesmo que o elemento à direita tenha vários níveis de profundidade.

Para tornar a dependência estrita no primeiro nível, você pode usar o símbolo entre os dois tokens:>

```
p > span {  
    color: yellow;  
}
```

Nesse caso, se a não for um dos primeiros filhos do elemento, ele não terá a nova cor aplicada.

As crianças diretas terão o estilo aplicado:

```
<p>  
    <span> This is yellow </span>  
    <strong>  
        <span> This is not yellow </span>  
    </strong>  
</p>
```

Os seletores irmãos adjacentes nos permitem estilizar um elemento somente se precedido por um elemento específico. Fazemo-lo utilizando o operador:+

Exemplo:

```
p + span {  
    color: yellow;  
}
```

Isso atribuirá a cor amarela a todos os elementos de extensão precedidos por um elemento:

```
<p>This is a paragraph</p>
<span>This is a yellow span</span>
```

Temos muito mais seletores que podemos usar:

- seletores de atributo
- seletores de pseudoclasse
- seletores de pseudoelementos

Encontraremos tudo sobre eles nas próximas seções.

## 6. Cascata

Cascata é um conceito fundamental de CSS. Afinal, está no próprio nome, o primeiro C do CSS – Cascading Style Sheets – deve ser uma coisa importante.

O que isso significa?

Cascata é o processo, ou algoritmo, que determina as propriedades aplicadas a cada elemento na página. Tentando convergir de uma lista de regras CSS que são definidas em vários lugares.

Fá-lo tendo em consideração:

- especificidade
- importância
- herança
- ordem no arquivo

Também cuida da resolução de conflitos.

Duas ou mais regras CSS concorrentes para a mesma propriedade aplicada ao mesmo elemento precisam ser elaboradas de acordo com a especificação CSS, para determinar qual delas precisa ser aplicada.

Mesmo que você tenha apenas um arquivo CSS carregado pela sua página, há outro CSS que fará parte do processo. Temos o navegador (agente do

usuário) CSS. Os navegadores vêm com um conjunto padrão de regras, todas diferentes entre os navegadores.

Então seu CSS entra em jogo.

Em seguida, o navegador aplica qualquer folha de estilo do usuário, que também pode ser aplicada por extensões do navegador.

Todas essas regras entram em jogo durante a renderização da página.

Veremos agora os conceitos de especificidade e herança.

## 7. Especificidade

O que acontece quando um elemento é alvo de várias regras, com seletores diferentes, que afetam a mesma propriedade?

Por exemplo, vamos falar sobre esse elemento:

```
<p class="dog-name">Roger</p>
```

Podemos ter

```
.dog-name {  
    color: yellow;  
}
```

e outra regra que tem como alvo , que define a cor para outro valor:p

```
p {  
    color: red;  
}
```

E outra regra que visa. Qual regra terá precedência sobre as outras, e por quê?p .dog-name

Insira a especificidade. **A regra mais específica vencerá.** Se duas ou mais regras tiverem a **mesma especificidade, a que aparecer por último vence.**

Às vezes, o que é mais específico na prática é um pouco confuso para iniciantes. Eu diria que também é confuso para os especialistas que não olham para essas regras com frequência, ou simplesmente as ignoram.

## 7.1. Como calcular a especificidade

A especificidade é calculada usando uma convenção.

Temos 4 slots, e cada um deles começa em 0: . O slot à esquerda é o mais importante, e o mais à direita é o menos importante. 0 0 0 0

Como funciona para números no sistema decimal: é maior que .1 0 0 00 1  
0 0

### 7.1.1. Ranhura 1

O primeiro slot, o mais à direita, é o menos importante.

Aumentamos esse valor quando temos um **seletor de elementos**. Um elemento é um nome de tag. Se você tiver mais de um seletor de elemento na regra, incremente de acordo com o valor armazenado nesse slot.

Exemplos:

```
p {  
} /* 0 0 0 1 */  
span {  
} /* 0 0 0 1 */  
p span {  
} /* 0 0 0 2 */  
p > span {  
} /* 0 0 0 2 */  
div p > span {  
} /* 0 0 0 3 */
```

## 7.1.2. Ranhura 2

O segundo slot é incrementado por 3 coisas:

- seletores de classe
- seletores de pseudoclasse
- seletores de atributo

Toda vez que uma regra atende a uma delas, incrementamos o valor da segunda coluna a partir da direita.

Exemplos:

```
.name {
} /* 0 0 1 0 */
.users .name {
} /* 0 0 2 0 */
[href$=".pdf"] {
} /* 0 0 1 0 */
:hover {
} /* 0 0 1 0 */
```

É claro que os seletores de slot 2 podem ser combinados com seletores de slot 1:

```
div .name {
} /* 0 0 1 1 */
a[href$=".pdf"] {
} /* 0 0 1 1 */
.pictures img:hover {
} /* 0 0 2 1 */
```

Um bom truque com as aulas é que você pode repetir a mesma classe e aumentar a especificidade. Por exemplo:

```
.name {
} /* 0 0 1 0 */
```

```
.name.name {
} /* 0 0 2 0 */
.name.name.name {
} /* 0 0 3 0 */
```

## 7.1.3. Ranhura 3

O slot 3 contém a coisa mais importante que pode afetar sua especificidade CSS em um arquivo CSS: o `.id`

Cada elemento pode ter um atributo atribuído, e podemos usá-lo em nossa folha de estilo para direcionar o elemento `.id`

Exemplos:

```
#name {
} /* 0 1 0 0 */
.user #name {
} /* 0 1 1 0 */
#name span {
} /* 0 1 0 1 */
```

## 7.1.4. Ranhura 4

O slot 4 é afetado por estilos embutidos. Qualquer estilo embutido terá precedência sobre qualquer regra definida em um arquivo CSS externo ou dentro da tag no cabeçalho da página `style`

Exemplo:

```
<p style="color: red">Test</p> /* 1 0 0 0 */
```

Mesmo que qualquer outra regra no CSS defina a cor, essa regra de estilo embutido será aplicada. Exceto por um caso - se for usado, que preenche o slot 5. `!important`

## 7.2. Importância

A especificidade não importa se uma regra termina com: !important

```
p {  
    font-size: 20px !important;  
}
```

Essa regra terá precedência sobre qualquer regra com mais especificidade.

Adicionar uma regra CSS fará com que essa regra seja mais importante do que qualquer outra regra, de acordo com as regras de especificidade. A única maneira de outra regra ter precedência é ter também e ter maior especificidade nos outros slots menos importantes. !important !important

## 7.3. Dicas

Em geral, você deve usar a quantidade de especificidade que você precisa, mas não mais. Dessa forma, você pode criar outros seletores para substituir as regras definidas pelas regras anteriores sem enlouquecer.

!important é uma ferramenta altamente debatida que o CSS nos oferece. Muitos especialistas em CSS defendem contra o uso. Eu me vejo usando-o especialmente ao experimentar algum estilo e uma regra CSS tem tanta especificidade que eu preciso usar para fazer o navegador aplicar meu novo CSS.!important

Mas, geralmente, não deve ter lugar em seus arquivos CSS.!important

Usar o atributo `to style` CSS também é muito debatido, já que tem uma especificidade muito alta. Uma boa alternativa é usar classes, que têm menos especificidade e, portanto, são mais fáceis de trabalhar e são mais poderosas (você pode ter várias classes para um elemento e uma classe pode ser reutilizada várias vezes).`id`

## 7.4. Ferramentas para calcular a especificidade

Você pode usar o site <https://specificity.keegan.st/> para executar o cálculo de especificidade para você automaticamente.

É útil, especialmente se você está tentando descobrir as coisas, pois pode ser uma boa ferramenta de feedback.

## 8. Herança

Quando você define algumas propriedades em um seletor no CSS, elas são herdadas por todos os filhos desse seletor.

Eu disse *alguns*, porque nem todas as propriedades mostram esse comportamento.

Isso acontece porque algumas propriedades fazem sentido para serem herdadas. Isso nos ajuda a escrever CSS de forma muito mais concisa, já que não precisamos definir explicitamente essa propriedade novamente em todos os filhos.

Algumas outras propriedades fazem mais sentido *para não* serem herdadas.

Pense em fontes: você não precisa aplicar o a cada tag da sua página. Você define a fonte da marca e todos os filhos a herdam, juntamente com outras propriedades.`font-family`

A propriedade, por outro lado, faz pouco sentido para ser herdada.`background-color`

### 8.1. Propriedades que herdam

Aqui está uma lista das propriedades que herdam. A lista não é abrangente, mas essas regras são apenas as mais populares que você provavelmente usará:

- `border-collapse` (recolher de borda)
- espaço entre bordas
- lado da legenda
- Cor
- cursor
- direção
- células vazias
- família de fontes

- tamanho da fonte
- estilo de fonte
- variante de fonte
- peso da fonte
- font-size-adjust
- font-stretch
- fonte
- espaçamento entre letras
- altura da linha
- imagem-estilo-de-lista
- posição no estilo de lista
- tipo de estilo de lista
- estilo de lista
- Órfãos
- aspas
- tamanho da guia
- alinhamento de texto
- texto-alinhar-último
- texto-decoração-cor
- recuo de texto
- texto-justificar
- texto-sombra
- transformação de texto
- visibilidade
- espaço em branco
- Viúvas
- quebra de palavras
- espaçamento entre palavras

## 8.2. Forçando as propriedades a herdar

E se você tiver uma propriedade que não é herdada por padrão, e você quiser que ela seja, em um filho?

Nos filhos, você define o valor da propriedade como a palavra-chave especial `.inherit`

Exemplo:

```
body {  
    background-color: yellow;  
}  
  
p {  
    background-color: inherit;  
}
```

## 8.3. Forçando as propriedades a NÃO herdar

Pelo contrário, você pode ter uma propriedade herdada e deseja evitá-lo.

Você pode usar a palavra-chave para revertê-la. Nesse caso, o valor é revertido para o valor original que o navegador lhe deu em sua folha de estilo padrão.`revert`

Na prática, isso raramente é usado e, na maioria das vezes, você apenas define outro valor para a propriedade para substituir esse valor herdado.

## 8.4. Outros valores especiais

Além das palavras-chave especiais que acabamos de ver, você também pode definir qualquer propriedade como:`inherit``revert`

- `initial`: use a folha de estilo padrão do navegador, se disponível. Caso contrário, e se a propriedade herdar por padrão, herde o valor. Caso contrário, não faça nada.
- `unset`: se a propriedade herda por padrão, herdar. Caso contrário, não faça nada.

## 9. Importação

De qualquer arquivo CSS, você pode importar outro arquivo CSS usando a diretiva.`@import`

Aqui está como você usá-lo:

```
@import url(myfile.css);
```

url() pode gerenciar URLs absolutas ou relativas.

Uma coisa importante que você precisa saber é que as diretivas devem ser colocadas antes de qualquer outro CSS no arquivo, ou elas serão ignoradas.

Você pode usar descritores de mídia para carregar apenas um arquivo CSS na mídia específica:

```
@import url(myfile.css) all;  
@import url(myfile-screen.css) screen;  
@import url(myfile-print.css) print;
```

## 10. Seletores de atributos

Já introduzimos vários dos seletores CSS básicos: usando seletores de elementos, classe, id, como combiná-los, como segmentar várias classes, como estilizar vários seletores na mesma regra, como seguir a hierarquia de páginas com seletores filho e filho direto e irmãos adjacentes.

Nesta seção, analisaremos os seletores de atributos e falaremos sobre seletores de pseudoclasse e pseudoelemento nas próximas 2 seções.

### 10.1. Seletores de presença de atributo

O primeiro tipo de seletor é o seletor de presença de atributo.

Podemos verificar se um elemento **tem** um atributo usando a sintaxe. selecionará todas as tags na página que têm um atributo, independentemente de seu valor:  
[ ]p[id]pid

```
p[id] {  
  /* ... */  
}
```

## 10.2. Seletores de valor de atributo exatos

Dentro dos colchetes, você pode verificar o valor do atributo usando o `=`, e o CSS será aplicado somente se o atributo corresponder ao valor exato especificado:=

```
p[id='my-id'] {  
    /* ... */  
}
```

## 10.3. Correspondêr a uma parte do valor do atributo

Enquanto vamos verificar o valor exato, temos outros operadores:=

- `*=` verifica se o atributo contém o
- `^=` verifica se o atributo começa com o
- `$=` verifica se o atributo termina com o
- `|=` verifica se o atributo começa com o parcial e é seguido por um traço (comum em classes, por exemplo) ou apenas contém o parcial
- `~=` verifica se o parcial está contido no atributo, mas separado por espaços do resto

Todas as verificações que mencionamos diferenciam **maiúsculas de minúsculas**.

Se você adicionar um pouco antes do colchete de fechamento, a verificação não diferenciará maiúsculas de minúsculas. É suportado em muitos navegadores, mas não em todos, verifique <https://caniuse.com/#feat=css-case-insensitive.i>

## 11. Pseudoclasses

Pseudoclasses são palavras-chave predefinidas que são usadas para selecionar um elemento com base em seu **estado** ou para segmentar um filho específico.

## Eles começam com um **único colón** :

Eles podem ser usados como parte de um seletor e são muito úteis para estilizar links ativos ou visitados, por exemplo, alterar o estilo ao focalizar, focar ou segmentar o primeiro filho ou linhas ímpares. Muito útil em muitos casos.

Estas são as pseudoclasses mais populares que você provavelmente usará:

Pseudoclasse	Alvos
:active	um elemento que está sendo ativado pelo usuário (por exemplo, clicado). Usado principalmente em links ou botões
:checked	uma caixa de seleção, opção ou tipos de entrada de rádio que estão habilitados
:default	o padrão em um conjunto de opções (como, opção em uma seleção ou botões de opção)
:disabled	um elemento desativado
:empty	um elemento sem filhos
:enabled	um elemento habilitado (oposto a :disabled)
:first-child	o primeiro filho de um grupo de irmãos
:focus	o elemento com foco
:hover	um elemento pairado com o mouse
:last-child	o último filho de um grupo de irmãos
:link	um link que não foi visitado
:not()	qualquer elemento que não corresponda ao seletor passou. Por exemplo, :not(span)
:nth-child()	um elemento correspondente à posição especificada
:nth-last-child()	um elemento correspondente à posição específica, a partir do final
:only-child	um elemento sem irmãos
:required	um elemento de formulário com o atributo definido required
:root	representa o elemento. É como segmentar, mas é mais específico. Útil em <a href="#">variáveis CSS.html</a>
:target	o elemento correspondente ao fragmento de URL atual (para navegação interna na página)

## Pseudoclasse

	<b>Alvos</b>
:valid	elementos de formulário que validaram o lado do cliente com êxito
:visited	um link que foi visitado

Vamos dar um exemplo. Um comum, na verdade. Você deseja estilizar um link, para criar uma regra CSS para direcionar o elemento:a

```
a {
  color: yellow;
}
```

As coisas parecem funcionar bem, até que você clique em um link. O link volta para a cor predefinida (azul) quando você clica nele. Então, quando você abre o link e volta para a página, agora o link é azul.

Por que isso acontece?

Porque o link quando clicado muda de estado e vai para o estado. E quando foi visitado, é no estado. Para sempre, até que o usuário limpe o histórico de navegação.:active:visited

Então, para tornar o link amarelo corretamente em todos os estados, você precisa escrever

```
a,
a:visited,
a:active {
  color: yellow;
}
```

:nth-child() merece uma menção especial. Ele pode ser usado para segmentar crianças ímpares ou pares com e .:nth-child(odd):nth-child(even)

É comumente usado em listas para colorir linhas ímpares de forma diferente das linhas pares:

```
ul:nth-child(odd) {  
    color: white;  
    background-color: black;  
}
```

Você também pode usá-lo para direcionar os 3 primeiros filhos de um elemento com . Ou você pode estilizar 1 em cada 5 elementos com .:nth-child(-n+3):nth-child(5n)

Algumas pseudoclasses são usadas apenas para impressão, como , para que você possa segmentar a primeira página, todas as páginas à esquerda e todas as páginas direitas, que geralmente são estilizadas de forma ligeiramente diferente.:first:left:right

## 12. Pseudo-elementos

Pseudo-elementos são usados para estilizar uma parte específica de um elemento.

Eles começam com um colón duplo .::

Às vezes, você os identificará na natureza com um único dois-pontos, mas essa é apenas uma sintaxe suportada por razões de compatibilidade com versões anteriores. Você deve usar 2 dois pontos para distingui-los de pseudo-classes.

::before e ::after são provavelmente os pseudoelementos mais utilizados. Eles são usados para adicionar conteúdo antes ou depois de um elemento, como ícones, por exemplo.::after

Aqui está a lista dos pseudo-elementos:

<b>Pseudo-elemento</b>	<b>Alvos</b>
::after	cria um pseudoelemento após o elemento
::before	cria um pseudoelemento antes do elemento

## Pseudo-elemento

### Alvos

::first-letter	pode ser usado para estilizar a primeira letra de um bloco de texto
::first-line	pode ser usado para estilizar a primeira linha de um bloco de texto
::selection	direciona o texto selecionado pelo usuário

Vamos dar um exemplo. Digamos que você queira tornar a primeira linha de um parágrafo um pouco maior em tamanho de fonte, uma coisa comum na tipografia:

```
p::first-line {
    font-size: 2rem;
}
```

Ou talvez você queira que a primeira letra seja mais ousada:

```
p::first-letter {
    font-weight: bolder;
}
```

::after e ::before são um pouco menos intuitivos. Lembro-me de usá-los quando tive que adicionar ícones usando CSS ::before

Você especifica a propriedade para inserir qualquer tipo de conteúdo após ou antes de um elemento::content

```
p::before {
    content: url(/myimage.png);
```

```
.myElement::before {
    content: 'Hey Hey!';
```

# 13. Cores

Por padrão, uma página HTML é renderizada por navegadores da Web, infelizmente, em termos das cores usadas.

Temos um fundo branco, cor preta e links azuis. É isso.

Felizmente, o CSS nos dá a capacidade de adicionar cores aos nossos projetos.

Temos estas propriedades:

- `color`
- `background-color`
- `border-color`

Todos eles aceitam um **valor de cor**, que pode estar em diferentes formas.

## 13.1. Cores nomeadas

Primeiro, temos palavras-chave CSS que definem cores. CSS começou com 16, mas hoje há um grande número de nomes de cores:

- `aliceblue`
- `antiquewhite`
- `aqua`
- `aquamarine`
- `azure`
- `beige`
- `bisque`
- `black`
- `blanchedalmond`
- `blue`
- `blueviolet`
- `brown`
- `burlywood`
- `cadetblue`

- chartreuse
- chocolate
- coral
- cornflowerblue
- cornsilk
- crimson
- cyan
- darkblue
- darkcyan
- darkgoldenrod
- darkgray
- darkgreen
- darkgrey
- darkkhaki
- darkmagenta
- darkolivegreen
- darkorange
- darkorchid
- darkred
- 暗沙色 (darksalmon)
- darkseagreen
- darkslateblue
- darkslategray
- darkslategrey
- darkturquoise
- darkviolet
- deeppink
- deepskyblue
- dimgray
- dimgrey
- dodgerblue
- firebrick
- floralwhite
- forestgreen
- fuchsia
- gainsboro

- ghostwhite
- gold
- goldenrod
- gray
- green
- greenyellow
- grey
- honeydew
- hotpink
- indianred
- indigo
- ivory
- khaki
- lavender
- lavenderblush
- lawngreen
- lemonchiffon
- lightblue
- lightcoral
- lightcyan
- lightgoldenrodyellow
- lightgray
- lightgreen
- lightgrey
- lightpink
- lightsalmon
- lightseagreen
- lightskyblue
- lightslategray
- lightslategrey
- lightsteelblue
- lightyellow
- lime
- limegreen
- linen
- magenta

- maroon
- mediumaquamarine
- mediumblue
- mediumorchid
- mediumpurple
- mediumseagreen
- mediumslateblue
- mediumspringgreen
- mediumturquoise
- mediumvioletred
- midnightblue
- mintcream
- mistyrose
- moccasin
- navajowhite
- navy
- oldlace
- olive
- olivedrab
- orange
- orangered
- orchid
- palegoldenrod
- palegreen
- paleturquoise
- palevioletred
- papayawhip
- peachpuff
- peru
- pink
- plum
- powderblue
- purple
- rebeccapurple
- red
- rosybrown

- royalblue
- saddlebrown
- salmon
- sandybrown
- seagreen
- seashell
- sienna
- silver
- skyblue
- slateblue
- slategray
- slategrey
- snow
- springgreen
- steelblue
- tan
- teal
- thistle
- tomato
- turquoise
- violet
- wheat
- white
- whitesmoke
- yellow
- yellowgreen

plus , and which points to the property, for example useful to make the  
inherit it.transparentcurrentColorcolorborder-color

They are defined in the [CSS Color Module, Level 4](#). They are case insensitive.

Wikipedia has a [nice table](#) which lets you pick the perfect color by its name.

Named colors are not the only option.

## 13.2. RGB and RGBC

You can use the function to calculate a color from its RGB notation, which sets the color based on its red-green-blue parts. From 0 to 255:rgb()

```
p {  
    color: rgb(255, 255, 255); /* white */  
    background-color: rgb(0, 0, 0); /* black */  
}
```

rgba() lets you add the alpha channel to enter a transparent part. That can be a number from 0 to 1:

```
p {  
    background-color: rgba(0, 0, 0, 0.5);  
}
```

### 13.3. Hexadecimal notation

Another option is to express the RGB parts of the colors in the hexadecimal notation, which is composed by 3 blocks.

Black, which is expressed as or (we can shortcut the 2 numbers to 1 if they are equal).rgb(0,0,0)#000000#000

White, can be expressed as or .rgb(255,255,255)#fffff#fff

The hexadecimal notation lets express a number from 0 to 255 in just 2 digits, since they can go from 0 to "15" (f).

We can add the alpha channel by adding 1 or 2 more digits at the end, for example . Not all browsers support the shortened notation, so use all 6 digits to express the RGB part.#00000033

### 13.4. HSL and HSLa

This is a more recent addition to CSS.

HSL = Hue Saturation Lightness.

In this notation, black is and white is `.hsl(0, 0%, 0%)hsl(0, 0%, 100%)`

If you are more familiar with HSL than RGB because of your past knowledge, you can definitely use that.

You also have which adds the alpha channel to the mix, again a number from 0 to 1: `hsla()hsl(0, 0%, 0%, 0.5)`

## 14. Units

One of the things you'll use every day in CSS are units. They are used to set lengths, paddings, margins, align elements and so on.

Things like `, , ,` or percentages.`pxemrem`

They are everywhere. There are some obscure ones, too. We'll go through each of them in this section.

### 14.1. Pixels

The most widely used measurement unit. A pixel does not actually correlate to a physical pixel on your screen, as that varies, a lot, by device (think high-DPI devices vs non-retina devices).

There is a convention that make this unit work consistently across devices.

### 14.2. Percentages

Another very useful measure, percentages let you specify values in percentages of that parent element's corresponding property.

Example:

```
.parent {  
    width: 400px;  
}  
  
.child {
```

```

width: 50%; /* = 200px */
}

```

## 14.3. Real-world measurement units

We have those measurement units which are translated from the outside world. Mostly useless on screen, they can be useful for print stylesheets. They are:

- `cm` a centimeter (maps to 37.8 pixels)
- `mm` a millimeter (0.1cm)
- `q` a quarter of a millimeter
- `in` an inch (maps to 96 pixels)
- `pt` a point (1 inch = 72 points)
- `pc` a pica (1 pica = 12 points)

## 14.4. Relative units

- `em` is the value assigned to that element's `font-size`, therefore its exact value changes between elements. It does not change depending on the font used, just on the font size. In typography this measures the width of the letter.`font-sizeem`
- `rem` is similar to `em`, but instead of varying on the current element font size, it uses the root element () font size. You set that font size once, and will be a consistent measure across all the page.`emhtmlrem`
- `ex` is like `em`, but instead of measuring the width of , it measures the height of the letter. It can change depending on the font used, and on the font size.`emmx`
- `ch` is like `em` but instead of measuring the height of it measures the width of (zero).`exx0`

## 14.5. Viewport units

- `vw` the **viewport width unit** represents a percentage of the viewport width. means 50% of the viewport width.`50vw`
- `vh` the **viewport height unit** represents a percentage of the viewport height. means 50% of the viewport height.`50vh`

- `vmin` the **viewport minimum unit** represents the minimum between the height or width in terms of percentage. is the 30% of the current width or height, depending which one is smaller<sup>30</sup>
- `vmax` the **viewport maximum unit** represents the maximum between the height or width in terms of percentage. is the 30% of the current width or height, depending which one is bigger<sup>30</sup>

## 14.6. Unidades fracionárias

fr são unidades fracionárias e são usadas no CSS Grid para dividir o espaço em frações.

Falaremos sobre eles no contexto do CSS Grid mais tarde.

## 15. url()

Quando falamos de imagens de plano de fundo e muito mais, usamos a função para carregar um recurso:`@import url()`

```
div {  
  background-image: url(test.png);  
}
```

Nesse caso, usei uma URL relativa, que pesquisa o arquivo na pasta onde o arquivo CSS está definido.

Eu poderia ir um nível para trás

```
div {  
  background-image: url(../test.png);  
}
```

ou entrar em uma pasta

```
div {  
  background-image: url(subfolder/test.png);
```

}

Ou eu poderia carregar um arquivo a partir da raiz do domínio onde o CSS está hospedado:

```
div {  
  background-image: url(/test.png);  
}
```

Ou eu poderia usar uma URL absoluta para carregar um recurso externo:

```
div {  
  background-image: url(https://mysite.com/test.png);  
}
```

## 16. calc()

A função permite que você execute operações matemáticas básicas em valores e é especialmente útil quando você precisa adicionar ou subtrair um valor de comprimento de uma porcentagem.`calc()`

É assim que funciona:

```
div {  
  max-width: calc(80% - 100px);  
}
```

Ele retorna um valor de comprimento, para que possa ser usado em qualquer lugar que você espere um valor de pixel.

Você pode executar

- adições usando +
- subtrações usando -
- multiplicação usando \*
- divisão usando /

Uma ressalva: com adição e subtração, o espaço ao redor do operador é obrigatório, caso contrário, não funciona como esperado.

Exemplos:

```
div {  
    max-width: calc(50% / 3);  
}
```

```
div {  
    max-width: calc(50% + 3px);  
}
```

## 17. Antecedentes

O plano de fundo de um elemento pode ser alterado usando várias propriedades CSS:

- `background-color`
- `background-image`
- `background-clip`
- `background-position`
- `background-origin`
- `background-repeat`
- `background-attachment`
- `background-size`

e a propriedade taquigráfica , que permite encurtar definições e agrupá-las em uma única linha.`background`

`background-color` aceita um valor de cor, que pode ser uma das palavras-chave de cor, ou um ou valor:`rgb``hsl`

```
p {  
    background-color: yellow;  
}
```

```
div {  
    background-color: #333;  
}
```

Em vez de usar uma cor, você pode usar uma imagem como plano de fundo para um elemento, especificando a URL de local da imagem:

```
div {  
    background-image: url(image.png);  
}
```

`background-clip` permite determinar a área usada pela imagem de plano de fundo ou cor. O valor padrão é `, que se estende até a borda externa.border-box`

Outros valores são

- `padding-box` para estender o plano de fundo até a borda de preenchimento, sem a borda
- `content-box` para estender o plano de fundo até a borda do conteúdo, sem o preenchimento
- `inherit` para aplicar o valor do pai

Ao usar uma imagem como plano de fundo, você desejará definir a posição do posicionamento da imagem usando a propriedade: `, ,` são todos os valores válidos para o eixo X e `, para o eixo Y:background-positionleftrightcenterbottom`

```
div {  
    background-position: top right;  
}
```

Se a imagem for menor que o plano de fundo, você precisará definir o comportamento usando o `. Deve , ou em todo o eixo? Este último é o valor padrão. Outro valor é .background-repeatrepeat-xrepeat-yrepeatno-repeat`

background-origin permite que você escolha onde o plano de fundo deve ser aplicado: para o elemento inteiro, incluindo o preenchimento (padrão) usando , para o elemento inteiro, incluindo a borda usando , para o elemento sem o preenchimento usando .padding-boxborder-boxcontent-box

Com podemos anexar o plano de fundo ao visor, de modo que a rolagem não afetará o plano de fundo:background-attachment

```
div {  
  background-attachment: fixed;  
}
```

Por padrão, o valor é . Há outro valor, . A melhor maneira de visualizar seu comportamento é [este Codepen](#).scrolllocal

A última propriedade em segundo plano é . Podemos usar 3 palavras-chave: , e . é o padrão.background-sizeautocovercontainauto

cover expande a imagem até que todo o elemento seja coberto pelo plano de fundo.

contain pára de expandir a imagem de plano de fundo quando uma dimensão (x ou y) cobre toda a menor borda da imagem, para que ela esteja totalmente contida no elemento.

Você também pode especificar um valor de comprimento e, em caso afirmativo, ele define a largura da imagem de plano de fundo (e a altura é definida automaticamente):

```
div {  
  background-size: 100%;  
}
```

Se você especificar 2 valores, um é a largura e o segundo é a altura:

```
div {  
    background-size: 800px 600px;  
}
```

A propriedade abreviada permite encurtar definições e agrupá-las em uma única linha.`background`

Este é um exemplo:

```
div {  
    background: url(bg.png) top left no-repeat;  
}
```

Se você usar uma imagem e a imagem não puder ser carregada, poderá definir uma cor de fallback:

```
div {  
    background: url(image.png) yellow;  
}
```

Você também pode definir um gradiente como plano de fundo:

```
div {  
    background: linear-gradient(#fff, #333);  
}
```

## 18. Comentários

CSS dá-lhe a capacidade de escrever comentários em um arquivo CSS, ou na tag no cabeçalho da página `style`

O formato são os comentários no estilo C (ou JavaScript, se preferir)./\* this  
is a comment \*/

Este é um comentário de várias linhas. Até que você adicione o token de fechamento, todas as linhas encontradas após a abertura são comentadas.\*/

Exemplo:

```
#name {  
    display: block;  
} /* Nice rule! */  
  
/* #name { display: block; } */  
  
#name {  
    display: block; /*  
        color: red;  
    */  
}
```

CSS não tem comentários embutidos, como em C ou JavaScript.//

Pay attention though - if you add before a rule, the rule will not be applied, looking like the comment worked. In reality, CSS detected a syntax error and due to how it works it ignored the line with the error, and went straight to the next line.//

Knowing this approach lets you purposefully write inline comments, although you have to be careful because you can't add random text like you can in a block comment.

For example:

```
// Nice rule!  
#name {  
    display: block;  
}
```

In this case, due to how CSS works, the rule is actually commented out. You can find more details [here](#) if you find this interesting. To avoid shooting yourself in the foot, just avoid using inline comments and rely on block comments.`#name`

# 19. Custom Properties

In the last few years CSS preprocessors had a lot of success. It was very common for greenfield projects to start with Less or Sass. And it's still a very popular technology.

The main benefits of those technologies are, in my opinion:

- They allow to nest selectors
- They provide an easy imports functionality
- They give you variables

Modern CSS has a new powerful feature called **CSS Custom Properties**, also commonly known as **CSS Variables**.

CSS is not a programming language like JavaScript, Python, PHP, Ruby or Go where variables are key to do something useful. CSS is very limited in what it can do, and it's mainly a declarative syntax to tell browsers how they should display an HTML page.

But a variable is a variable: a name that refers to a value, and variables in CSS helps reduce repetition and inconsistencies in your CSS, by centralizing the values definition.

And it introduces a unique feature that CSS preprocessors won't never have: **you can access and change the value of a CSS Variable programmatically using JavaScript**.

## 19.1. The basics of using variables

A CSS Variable is defined with a special syntax, prepending **two dashes** to a name (), then a colon and a value. Like this:--variable-name

```
:root {  
  --primary-color: yellow;  
}
```

(mais em mais tarde):root

Você pode acessar o valor da variável usando: var()

```
p {  
  color: var(--primary-color);  
}
```

O valor da variável pode ser qualquer valor CSS válido, por exemplo:

```
:root {  
  --default-padding: 30px 30px 20px 20px;  
  --default-color: red;  
  --default-background: #fff;  
}
```

## 19.2. Criar variáveis dentro de qualquer elemento

As variáveis CSS podem ser definidas dentro de qualquer elemento. Alguns exemplos:

```
:root {  
  --default-color: red;  
}
```

```
body {  
  --default-color: red;  
}
```

```
main {  
  --default-color: red;  
}
```

```
p {  
  --default-color: red;  
}
```

```
span {
```

```
--default-color: red;  
}  
  
a:hover {  
  --default-color: red;  
}
```

O que muda nesses diferentes exemplos é o **escopo**.

## 19.3. Âmbito das variáveis

A adição de variáveis a um seletor as torna disponíveis para todos os filhos dele.

No exemplo acima, você viu o uso de ao definir uma variável CSS::root

```
:root {  
  --primary-color: yellow;  
}
```

::root é uma pseudoclasse CSS que identifica o elemento raiz de uma árvore.

No contexto de um documento HTML, o uso do seletor aponta para o elemento, exceto que tem maior especificidade (tem prioridade).:roothtml:root

No contexto de uma imagem SVG, aponta para a tag.:rootsvg

Adicionar uma propriedade personalizada CSS para torná-la disponível para todos os elementos na página.:root

Se você adicionar uma variável dentro de um seletor, ela só estará disponível para filhos de:.container.container

```
.container {  
  --secondary-color: yellow;  
}
```

e usá-lo fora deste elemento não vai funcionar.

As variáveis podem ser **reatribuídas**:

```
:root {  
  --primary-color: yellow;  
}  
  
.container {  
  --primary-color: blue;  
}
```

Lá fora, será *amarelo*, mas dentro será *azul*. .container--primary-color

Você também pode atribuir ou substituir uma variável dentro do HTML usando **estilos embutidos**:

```
<main style="--primary-color: orange;">  
  <!-- ... -->  
</main>
```

As variáveis CSS seguem as regras normais de cascata CSS, com precedência definida de acordo com a especificidade

## 19.4. Interagindo com um valor de variável CSS usando JavaScript

A coisa mais legal com as variáveis CSS é a capacidade de acessá-las e editá-las usando JavaScript.

Veja como definir um valor de variável usando JavaScript simples:

```
const element = document.getElementById('my-element')  
element.style.setProperty('--variable-name', 'a-value')
```

Este código abaixo pode ser usado para acessar um valor de variável, caso a variável seja definida em :root

```
const styles = getComputedStyle(document.documentElement)
const value = String(styles.getPropertyValue('--variable-name')).trim()
```

Ou, para obter o estilo aplicado a um elemento específico, no caso de variáveis definidas com um escopo diferente:

```
const element = document.getElementById('my-element')
const styles = getComputedStyle(element)
const value = String(styles.getPropertyValue('--variable-name')).trim()
```

## 19.5. Manipulação de valores inválidos

Se uma variável for atribuída a uma propriedade que não aceita o valor da variável, ela será considerada inválida.

Por exemplo, você pode passar um valor de pixel para uma propriedade ou um valor rem para uma propriedade color.position

Nesse caso, a linha é considerada inválida e ignorada.

## 19.6. Suporte do navegador

O suporte do navegador para variáveis CSS é **muito bom**, de acordo com o Can I Use.

As variáveis CSS estão aqui para ficar, e você pode usá-las hoje se não precisar oferecer suporte ao Internet Explorer e a versões antigas dos outros navegadores.

Se você precisar oferecer suporte a navegadores mais antigos, poderá usar bibliotecas como PostCSS ou Myth, mas perderá a capacidade de interagir com variáveis via JavaScript ou Browser Developer Tools, pois elas são transempilhadas para o bom e velho CSS sem variáveis (e, como tal, você perde a maior parte do poder das variáveis CSS).

## 19.7. CSS As variáveis diferenciam maiúsculas de minúsculas

Esta variável:

```
--width: 100px;
```

é diferente de:

```
--Width: 100px;
```

## 19.8. Matemática em Variáveis CSS

Para fazer matemática em Variáveis CSS, você precisa usar , por exemplo: calc()

```
:root {  
  --default-left-padding: calc(10px * 2);  
}
```

## 19.9. Consultas de mídia com variáveis CSS

Nada de especial aqui. As variáveis CSS normalmente se aplicam a consultas de mídia:

```
body {  
  --width: 500px;  
}  
  
@media screen and (max-width: 1000px) and (min-width: 700px) {  
  --width: 800px;  
}  
  
.container {  
  width: var(--width);  
}
```

## 19.10. Definindo um valor de fallback para var()

var() aceita um segundo parâmetro, que é o valor de fallback padrão quando o valor da variável não está definido:

```
.container {  
    margin: var(--default-margin, 30px);  
}
```

## 20. Fontes

No alvorecer da web, você só tinha um punhado de fontes que você poderia escolher.

Felizmente hoje você pode carregar qualquer tipo de fonte em suas páginas.

CSS ganhou muitos recursos agradáveis ao longo dos anos em relação a fontes.

A propriedade é a abreviação de várias propriedades:font

- font-family
- font-weight
- font-stretch
- font-style
- font-size

Vamos ver cada um deles e depois vamos abordar.font

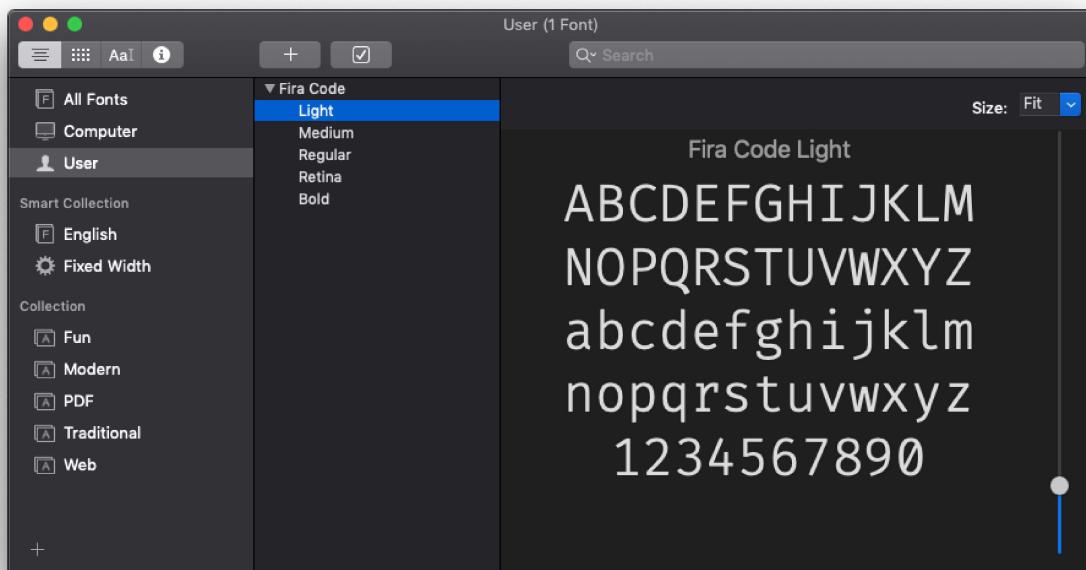
Em seguida, falaremos sobre como carregar fontes personalizadas, usando ou , ou carregando uma folha de estilo.@import@font-face

### 20.1. font-family

Define a família de *fontes* que o elemento usará.

Por que "família"? Porque o que conhecemos como fonte é, na verdade, composto de várias subfontes. que fornecem todo o estilo (negrito, itálico, leve ..) que precisamos.

Aqui está um exemplo do aplicativo Font Book do meu Mac - a família de fontes Fira Code hospeda várias fontes dedicadas abaixo:



Essa propriedade permite que você selecione uma fonte específica, por exemplo:

```
body {
    font-family: Helvetica;
}
```

Você pode definir vários valores, portanto, a segunda opção será usada se a primeira não puder ser usada por algum motivo (se não for encontrada na máquina ou se a conexão de rede para baixar a fonte falhar, por exemplo):

```
body {
    font-family: Helvetica, Arial;
}
```

Eu usei algumas fontes específicas até agora, aquelas que chamamos **de Web Safe Fonts**, pois são pré-instaladas em diferentes sistemas

operacionais.

Nós os dividimos em fontes Serif, Sans-Serif e Monospace. Aqui está uma lista de alguns dos mais populares:

## Serif

- Geórgia
- Palatino
- Times Novo Romano
- Vezes

## Sans-Serif

- Arial
- Helvetica
- Verdana
- Genebra
- Tahoma
- Lúcida Grande
- Impacto
- Trebuchet MS
- Arial Preto

## Monospace

- Courier Novo
- Correio
- Lucida Console
- Mónaco

Você pode usar todos eles como propriedades, mas não é garantido que eles estejam lá para todos os sistemas. Outros também existem, com um nível variável de apoio.`font-family`

Você também pode usar nomes genéricos:

- `sans-serif` uma fonte sem ligaduras
- `serif` uma fonte com ligaduras

- monospace uma fonte especialmente boa para o código
- cursive usado para simular peças manuscritas
- fantasy o nome diz tudo

Esses são normalmente usados no final de uma definição, para fornecer um valor de fallback no caso de nada mais poder ser aplicado:`font-family`

```
body {  
    font-family: Helvetica, Arial, sans-serif;  
}
```

## 20.2. `font-weight`

Essa propriedade define a largura de uma fonte. Você pode usar esses valores predefinidos:

- normal
- ousado
- mais ousado (em relação ao elemento pai)
- mais leve (em relação ao elemento pai)

Ou usando as palavras-chave numéricas

- 100
- 200
- 300
- 400, mapeado para normal
- 500
- 600
- 700 mapeados para bold
- 800
- 900

onde 100 é a fonte mais leve e 900 é a mais ousada.

Alguns desses valores numéricos podem não ser mapeados para uma fonte, porque isso deve ser fornecido na família de fontes. Quando um está faltando, o CSS faz com que esse número seja pelo menos tão negrito

quanto o anterior, para que você possa ter números que apontem para a mesma fonte.

## 20.3. font-stretch

Permite escolher uma face estreita ou larga da fonte, se disponível.

Isso é importante: a fonte deve estar equipada com faces diferentes.

Os valores permitidos são, de mais estreito para mais largo:

- ultra-condensed
- extra-condensed
- condensed
- semi-condensed
- normal
- semi-expanded
- expanded
- extra-expanded
- ultra-expanded

## 20.4. font-style

Permite que você aplique um estilo itálico a uma fonte:

```
p {  
    font-style: italic;  
}
```

Essa propriedade também permite os valores e . Há muito pouca, ou nenhuma, diferença entre usar e . O primeiro é mais fácil para mim, pois o HTML já oferece um elemento que significa  
**ítálico.obliquenormalitalicobliquei**

## 20.5. font-size

Essa propriedade é usada para determinar o tamanho das fontes.

Você pode passar 2 tipos de valores:

um valor de comprimento, como , , etc, ou uma porcentagem ou uma palavra-chave de valor predefinida

No segundo caso, os valores que você pode usar são:

- xx-pequeno
- x-pequeno
- pequeno
- Média
- grande
- x-grande
- xx-grande
- menor (em relação ao elemento pai)
- maior (em relação ao elemento pai)

Uso:

```
p {  
    font-size: 20px;  
}  
  
li {  
    font-size: medium;  
}
```

## 20.6. font-variant

Essa propriedade foi originalmente usada para alterar o texto para letras minúsculas e tinha apenas 3 valores válidos:

- normal
- inherit
- small-caps

Letras minúsculas significam que o texto é renderizado em "letras maiúsculas" ao lado de suas letras maiúsculas.

## 20.7. font

A propriedade permite aplicar diferentes propriedades de fonte em uma única, reduzindo a desordem.`font`

Devemos pelo menos definir 2 propriedades, e , as outras são opcionais:`font-size``font-family`

```
body {  
    font: 20px Helvetica;  
}
```

Se adicionarmos outras propriedades, elas precisam ser colocadas na ordem correta.

Esta é a ordem:

```
font: <font-stretch> <font-style> <font-variant> <font-weight> <font-size>  
<line-height> <font-family>;
```

Exemplo:

```
body {  
    font: italic bold 20px Helvetica;  
}  
  
section {  
    font: small-caps bold 20px Helvetica;  
}
```

## 20.8. Carregando fontes personalizadas usando @font-face

@font-face permite adicionar um novo nome de família de fontes e mapeá-lo para um arquivo que contém uma fonte.

Esta fonte será baixada pelo navegador e usada na página, e tem sido uma mudança tão fundamental para a tipografia na web - agora podemos usar qualquer fonte que quisermos.

Podemos adicionar declarações diretamente em nosso CSS ou vincular a um CSS dedicado à importação da fonte.`@font-face`

Em nosso arquivo CSS, também podemos usar para carregar esse arquivo CSS.`@import`

Uma declaração contém várias propriedades que usamos para definir a fonte, incluindo , o URI (um ou mais URIs) para a fonte. Isso segue a política de mesma origem, o que significa que as fontes só podem ser baixadas da origem atual (domínio + porta + protocolo).`@font-face``src`

As fontes geralmente estão nos formatos

- `woff` (Formato de fonte aberta da Web)
- `woff2` (Formato de fonte aberta da Web 2.0)
- `eot` (Tipo aberto incorporado)
- `otf` (Fonte OpenType)
- `ttf` (Fonte TrueType)

As seguintes propriedades nos permitem definir as propriedades para a fonte que vamos carregar, como vimos acima:

- `font-family`
- `font-weight`
- `font-style`
- `font-stretch`

## 20.9. Uma nota sobre o desempenho

É claro que carregar uma fonte tem implicações de desempenho que você deve considerar ao criar o design da sua página.

# 21. Tipografia

Já falamos sobre fontes, mas há mais no estilo do texto.

Nesta seção, falaremos sobre as seguintes propriedades:

- `text-transform`
- `text-decoration`
- `text-align`
- `vertical-align`
- `line-height`
- `text-indent`
- `text-align-last`
- `word-spacing`
- `letter-spacing`
- `text-shadow`
- `white-space`
- `tab-size`
- `writing-mode`
- `hyphens`
- `text-orientation`
- `direction`
- `line-break`
- `word-break`
- `overflow-wrap`

## 21.1. `text-transform`

Essa propriedade pode transformar o caso de um elemento.

Existem 4 valores válidos:

- `capitalize` em maiúsculas a primeira letra de cada palavra
- `uppercase` para maiúsculas todo o texto
- `lowercase` para minúsculas todo o texto
- `none` para desabilitar a transformação do texto, usada para evitar herdar a propriedade

Exemplo:

```
p {  
    text-transform: uppercase;  
}
```

## 21.2. text-decoration

Esta propriedade é usada para adicionar decorações ao texto, incluindo

- underline
- overline
- line-through
- blink
- none

Exemplo:

```
p {  
    text-decoration: underline;  
}
```

Você também pode definir o estilo da decoração e a cor.

Exemplo:

```
p {  
    text-decoration: underline dashed yellow;  
}
```

Os valores de estilo válidos são , , , .soliddoubleddotteddashedwavy

Você pode fazer tudo em uma linha ou usar as propriedades específicas:

- text-decoration-line
- text-decoration-color

- **text-decoration-style**

Exemplo:

```
p {
    text-decoration-line: underline;
    text-decoration-color: yellow;
    text-decoration-style: dashed;
}
```

## 21.3. text-align

Por padrão, o texto align tem o valor, o que significa que o texto começa no "início", origem 0, 0 da caixa que o contém. Isso significa canto superior esquerdo nos idiomas da esquerda para a direita e canto superior direito nos idiomas da direita para a esquerda.`start`

Os valores possíveis são , , , , (é bom ter um espaçamento consistente nas extremidades da linha):`start end left right center justify`

```
p {
    text-align: right;
}
```

## 21.4. vertical-align

Determina como os elementos embutidos são alinhados verticalmente.

Temos vários valores para este imóvel. Primeiro, podemos atribuir um valor de comprimento ou porcentagem. Eles são usados para alinhar o texto em uma posição maior ou menor (usando valores negativos) do que a linha de base do elemento pai.

Então temos as palavras-chave:

- **baseline** (o padrão), alinha a linha de base à linha de base do elemento pai

- `sub` torna um elemento subscrito, simulando o resultado do elemento `HTMLsub`
- `super` torna um elemento sobrescrito, simulando o resultado do elemento `HTMLsup`
- `top` alinhar a parte superior do elemento à parte superior da linha
- `text-top` alinhar a parte superior do elemento à parte superior da fonte do elemento pai
- `middle` alinhar o meio do elemento ao meio da linha do pai
- `bottom` alinhar a parte inferior do elemento à parte inferior da linha
- `text-bottom` alinhar a parte inferior do elemento à parte inferior da fonte do elemento pai

## 21.5. line-height

Isso permite que você altere a altura de uma linha. Cada linha de texto tem uma determinada altura de fonte, mas depois há espaçamento adicional verticalmente entre as linhas. Essa é a altura da linha:

```
p {  
    line-height: 0.9rem;  
}
```

## 21.6. text-indent

Recue a primeira linha de um parágrafo por um comprimento definido ou por uma porcentagem da largura do parágrafo:

```
p {  
    text-indent: -10px;  
}
```

## 21.7. text-align-last

Por padrão, a última linha de um parágrafo é alinhada após o valor. Use essa propriedade para alterar esse comportamento:`text-align`

```
p {  
    text-align-last: right;  
}
```

## 21.8. word-spacing

Modifica o espaçamento entre cada palavra.

Você pode usar a palavra-chave para redefinir valores herdados ou usar um valor de comprimento: normal

```
p {  
    word-spacing: 2px;  
}  
  
span {  
    word-spacing: -0.2em;  
}
```

## 21.9. letter-spacing

Modifica o espaçamento entre cada letra.

Você pode usar a palavra-chave para redefinir valores herdados ou usar um valor de comprimento: normal

```
p {  
    letter-spacing: 0.2px;  
}  
  
span {  
    letter-spacing: -0.2em;  
}
```

## 21.10. text-shadow

Aplique uma sombra ao texto. Por padrão, o texto agora tem sombra.

Essa propriedade aceita uma cor opcional e um conjunto de valores que definem

- o deslocamento X da sombra do texto
- o deslocamento Y da sombra do texto
- o raio de desfoque

Se a cor não for especificada, a sombra usará a cor do texto.

Exemplos:

```
p {  
    text-shadow: 0.2px 2px;  
}  
  
span {  
    text-shadow: yellow 0.2px 2px 3px;  
}
```

## 21.11. white-space

Define como o CSS manipula o espaço em branco, novas linhas e guias dentro de um elemento.

Os valores válidos que recolhem o espaço em branco são:

- `normal` recolhe o espaço em branco. Adiciona novas linhas quando necessário à medida que o texto atinge o final do contêiner
- `nowrap` recolhe o espaço em branco. Não adiciona uma nova linha quando o texto atinge o final do contêiner e suprime qualquer quebra de linha adicionada ao texto
- `pre-line` recolhe o espaço em branco. Adiciona novas linhas quando necessário à medida que o texto atinge o final do contêiner

Os valores válidos que preservam o espaço em branco são:

- pre preserva o espaço em branco. Não adiciona uma nova linha quando o texto chega ao final do contêiner, mas preserva a quebra de linha adicionada ao texto
- pre-wrap preserva o espaço em branco. Adiciona novas linhas quando necessário à medida que o texto atinge o final do contêiner

## 21.12. tab-size

Define a largura do caractere de tabulação. Por padrão, é 8 e você pode definir um valor inteiro que define os espaços de caracteres necessários ou um valor de comprimento:

```
p {  
    tab-size: 2;  
}  
  
span {  
    tab-size: 4px;  
}
```

## 21.13. writing-mode

Define se as linhas de texto são dispostas horizontal ou verticalmente e a direção na qual os blocos progridem.

Os valores que você pode usar são

- horizontal-tb (padrão)
- vertical-rl o conteúdo é disposto verticalmente. Novas linhas são colocadas à esquerda do anterior
- vertical-lr o conteúdo é disposto verticalmente. Novas linhas são colocadas à direita do anterior

## 21.14. hyphens

Determina se os hífens devem ser adicionados automaticamente ao ir para uma nova linha.

Os valores válidos são

- none (padrão)
- manual adicione um hífen somente quando já houver um hífen visível ou um hífen oculto (um caractere especial)
- auto adicione hífens quando determinado que o texto pode ter um hífen.

## 21.15. text-orientation

Quando está em um modo vertical, determina a orientação do texto.`writing-mode`

Os valores válidos são

- mixed é o padrão e, se um idioma é vertical (como o japonês), ele preserva essa orientação, enquanto gira o texto escrito em idiomas ocidentais.
- upright faz com que todo o texto seja orientado verticalmente
- sideways torna todo o texto orientado horizontalmente

## 21.16. direction

Define a direção do texto. Os valores válidos são e :ltr rtl

```
p {  
  direction: rtl;  
}
```

## 21.17. word-break

Essa propriedade especifica como quebrar linhas dentro de palavras.

- normal (padrão) significa que o texto é apenas quebrado entre palavras, não dentro de uma palavra
- break-all o navegador pode quebrar uma palavra (mas nenhum hífen é adicionado)

- `keep-all` suprimir o embrulho macio. Usado principalmente para texto CJK (chinês/japonês/coreano).

Falando em texto CJK, a propriedade é usada para determinar como as linhas de texto são quebradas. Eu não sou um especialista com essas línguas, então vou evitar cobri-lo.

## 21.18. overflow-wrap

Se uma palavra for muito longa para caber em uma linha, ela pode transbordar para fora do contêiner.

Essa propriedade também é conhecida como `word-wrap`, embora isso não seja padrão (mas ainda funcione como um alias)

Esse é o comportamento padrão (`overflow-wrap: normal;`)

Podemos usar:

```
p {  
    overflow-wrap: break-word;  
}
```

para quebrá-lo no comprimento exato da linha, ou

```
p {  
    overflow-wrap: anywhere;  
}
```

se o navegador vir que há uma oportunidade de soft wrap em algum lugar mais cedo. Nenhum hífen é adicionado, em qualquer caso.

Esta propriedade é muito semelhante a `.word-break`. Podemos querer escolher este em línguas ocidentais, enquanto tem tratamento especial para línguas não-ocidentais.

## 22. Modelo de caixa

Cada elemento CSS é essencialmente uma caixa. Cada elemento é uma caixa genérica.

The box model explains the sizing of the elements based on a few CSS properties.

From the inside to the outside, we have:

- the content area
- padding
- border
- margin

The best way to visualize the box model is to open the browser DevTools and check how it is displayed:

The screenshot shows the Firefox DevTools Layout panel. At the top, there's a tree view with 'Box Model' expanded. Below it is a visual representation of a box with various dimensions labeled: 'margin' (0), 'border' (0), 'padding' (0), and 'content-box' (226.133x139.167). The total width is 234.133 and height is 139.167. The 'static' value for position is also shown. The bottom part of the panel lists 'Box Model Properties' with the following values:

box-sizing	content-box	line-height	46.6667px
display	inline	position	static
float	none	z-index	auto

Here you can see how Firefox tells me the properties of a element I highlighted. I right-clicked on it, pressed Inspect Element, and went to the Layout panel of the DevTools.span

See, the light blue space is the content area. Surrounding it there is the padding, then the border and finally the margin.

By default, if you set a width (or height) on the element, that is going to be applied to the **content area**. All the padding, border, and margin

calculations are done outside of the value, so you have to take this in mind when you do your calculation.

You can change this behavior using Box Sizing.

## 23. Border

The border is a thin layer between padding and margin. Editing the border you can make elements draw their perimeter on screen.

You can work on borders by using those properties:

- `border-style`
- `border-color`
- `border-width`

The property can be used as a shorthand for all those properties:`border`

`border-radius` is used to create rounded corners.

You also have the ability to use images as borders, an ability given to you by and its specific separate properties:`border-image`

- `border-image-source`
- `border-image-slice`
- `border-image-width`
- `border-image-outset`
- `border-image-repeat`

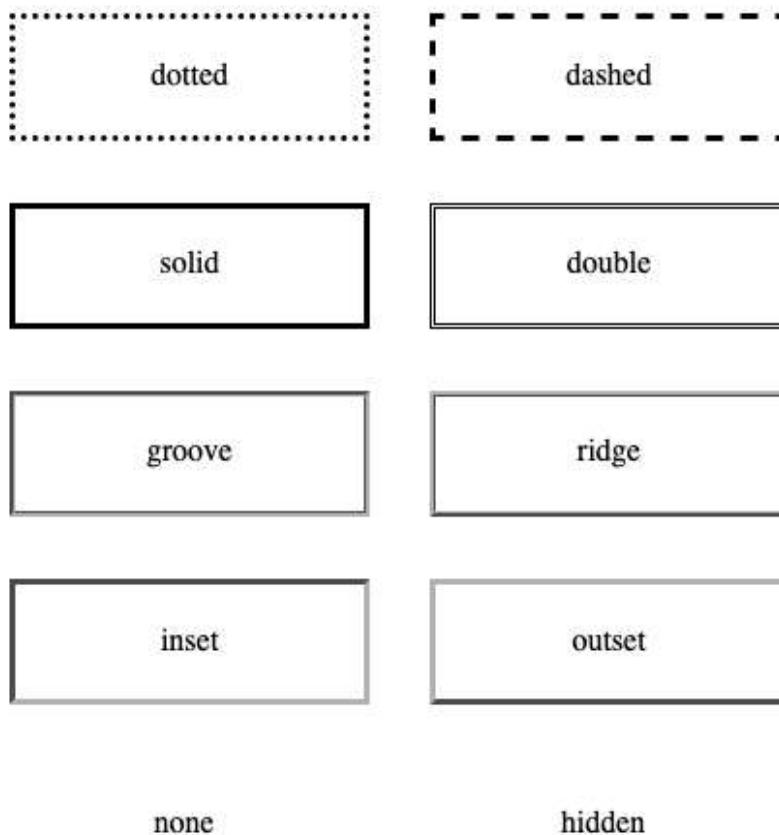
Let's start with `.border-style`

### 23.1. The border style

The property lets you choose the style of the border. The options you can use are:`border-style`

- `dotted`

- dashed
- solid
- double
- groove
- ridge
- inset
- outset
- none
- hidden



Check [this Codepen](#) for a live example

The default for the style is `, so to make the border appear at all you need to change it to something else. is a good choice most of the times.`

You can set a different style for each edge using the properties

- `border-top-style`
- `border-right-style`
- `border-bottom-style`
- `border-left-style`

or you can use with multiple values to define them, using the usual Top-Right-Bottom-Left order:`border-style`

```
p {  
    border-style: solid dotted solid dotted;  
}
```

## 23.2. The border width

`border-width` is used to set the width of the border.

You can use one of the pre-defined values:

- `thin`
- `medium` (the default value)
- `thick`

or express a value in pixels, em or rem or any other valid length value.

Example:

```
p {  
    border-width: 2px;  
}
```

You can set the width of each edge (Top-Right-Bottom-Left) separately by using 4 values:

```
p {  
    border-width: 2px 1px 2px 1px;  
}
```

or you can use the specific edge properties , , , .`border-top-width``border-right-width``border-bottom-width``border-left-width`

## 23.3. The border color

border-color is used to set the color of the border.

If you don't set a color, the border by default is colored using the color of the text in the element.

You can pass any valid color value to .border-color

Example:

```
p {  
    border-color: yellow;  
}
```

You can set the color of each edge (Top-Right-Bottom-Left) separately by using 4 values:

```
p {  
    border-color: black red yellow blue;  
}
```

or you can use the specific edge properties , , , .border-top-color border-right-color border-bottom-color border-left-color

## 23.4. The border shorthand property

Those 3 properties mentioned, , and can be set using the shorthand property .border-widthborder-styleborder-colorborder

Example:

```
p {  
    border: 2px black solid;  
}
```

You can also use the edge-specific properties , , , .border-topborder-rightborder-bottomborder-left

Example:

```
p {  
    border-left: 2px black solid;  
    border-right: 3px red dashed;  
}
```

## 23.5. O raio da borda

border-radius é usado para definir cantos arredondados para a borda. Você precisa passar um valor que será usado como o raio do círculo que será usado para arredondar a borda.

Uso:

```
p {  
    border-radius: 3px;  
}
```

Você também pode usar as propriedades específicas da borda , , , .border-top-left-radiusborder-top-right-radiusborder-bottom-left-radiusborder-bottom-right-radius

## 23.6. Usando imagens como bordas

Uma coisa muito legal com bordas é a capacidade de usar imagens para estilizá-las. Isso permite que você seja muito criativo com as fronteiras.

Temos 5 propriedades:

- border-image-source
- border-image-slice
- border-image-width
- border-image-outset
- border-image-repeat

e a taquigrafia . Eu não vou entrar em muitos detalhes aqui, pois as imagens, pois as bordas precisariam de uma cobertura mais aprofundada como a que eu posso fazer neste pequeno capítulo. Eu recomendo a leitura da [entrada do almanaque CSS Tricks na imagem de borda](#) para obter mais informações.`border-image`

## 24. Preenchimento

A propriedade CSS é comumente usada em CSS para adicionar espaço no lado interno de um elemento.`padding`

Lembrar:

- `margin` adiciona espaço fora de uma borda de elemento
- `padding` adiciona espaço dentro de uma borda de elemento

### 24.1. Propriedades específicas de preenchimento

`padding` tem 4 propriedades relacionadas que alteram o preenchimento de uma única borda de uma só vez:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

O uso deles é muito simples e não pode ser confundido, por exemplo:

```
padding-left: 30px;  
padding-right: 3em;
```

### 24.2. Usando a taquigrafiapadding

`padding` é uma abreviatura para especificar vários valores de preenchimento ao mesmo tempo e, dependendo do número de valores inseridos, ele se comporta de maneira diferente.

## 24.2.1. 1 valor

O uso de um único valor aplica isso a **todos os** preenchimentos: superior, direito, inferior, esquerdo.

```
padding: 20px;
```

## 24.2.2. 2 valores

O uso de 2 valores aplica-se o primeiro à **parte inferior e superior**, e o segundo à **esquerda e à direita**.

```
padding: 20px 10px;
```

## 24.2.3. 3 valores

O uso de 3 valores aplica-se o primeiro à parte **superior**, o segundo à **esquerda e à direita**, o terceiro à **parte inferior**.

```
padding: 20px 10px 30px;
```

## 24.2.4. 4 valores

O uso de 4 valores aplica-se o primeiro à parte **superior**, o segundo à **direita**, o terceiro à **parte inferior**, o quarto à **esquerda**.

```
padding: 20px 10px 5px 0px;
```

Assim, a ordem é *superior-direita-inferior-esquerda*.

## 24.3. Valores aceitos

`padding` aceita valores expressos em qualquer tipo de unidade de comprimento, os mais comuns são px, em, rem, mas existem muitos outros.

# 25. Margem

A propriedade CSS é comumente usada em CSS para adicionar espaço ao redor de um elemento.`margin`

Lembrar:

- `margin` adiciona espaço fora de uma borda de elemento
- `padding` adiciona espaço dentro de uma borda de elemento

## 25.1. Propriedades de margem específicas

`margin` tem 4 propriedades relacionadas que alteram a margem de uma única aresta de uma só vez:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

O uso deles é muito simples e não pode ser confundido, por exemplo:

```
margin-left: 30px;  
margin-right: 3em;
```

## 25.2. Usando a taquigrafia `margin`

`margin` é uma abreviatura para especificar várias margens ao mesmo tempo e, dependendo do número de valores inseridos, ele se comporta de maneira diferente.

### 25.2.1. 1 valor

O uso de um único valor aplica isso a **todas as** margens: superior, direita, inferior, esquerda.

```
margin: 20px;
```

## 25.2.2. 2 valores

O uso de 2 valores aplica-se o primeiro à **parte inferior e superior**, e o segundo à **esquerda e à direita**.

```
margin: 20px 10px;
```

## 25.2.3. 3 valores

O uso de 3 valores aplica-se o primeiro à parte **superior**, o segundo à **esquerda e à direita**, o terceiro à **parte inferior**.

```
margin: 20px 10px 30px;
```

## 25.2.4. 4 valores

O uso de 4 valores aplica-se o primeiro à parte **superior**, o segundo à **direita**, o terceiro à **parte inferior**, o quarto à **esquerda**.

```
margin: 20px 10px 5px 0px;
```

Assim, a ordem é *superior-direita-inferior-esquerda*.

## 25.3. Valores aceitos

margin aceita valores expressos em qualquer tipo de unidade de comprimento, os mais comuns são px, em, rem, mas existem muitos outros.

Ele também aceita valores percentuais e o valor especial .auto

## 25.4. Usando para centralizar elementos auto

auto pode ser usado para dizer ao navegador para selecionar automaticamente uma margem, e é mais comumente usado para centralizar um elemento da seguinte maneira:

```
margin: 0 auto;
```

Como dito acima, o uso de 2 valores aplica o primeiro para baixo e para cima, e o segundo **para a esquerda e direita**.

A maneira moderna de centralizar elementos é usar o Flexbox e sua diretiva.justify-content: center;

Navegadores mais antigos, é claro, não implementam o Flexbox, e se você precisar suportá-los ainda é uma boa escolha.`margin: 0 auto;`

## 25.5. Utilização de uma margem negativa

`margin` é a única propriedade relacionada ao dimensionamento que pode ter um valor negativo. É extremamente útil também. Definir uma margem superior negativa faz com que um elemento se move sobre os elementos antes dele e, dado valor negativo suficiente, ele será movido para fora da página.

Uma margem inferior negativa sobe os elementos depois dela.

Uma margem direita negativa faz com que o conteúdo do elemento se expanda além do tamanho de conteúdo permitido.

Uma margem esquerda negativa move o elemento que sobrou sobre os elementos que a precedem e, dado valor negativo suficiente, ela será movida para fora da página.

## 26. Dimensionamento da caixa

O comportamento padrão dos navegadores ao calcular a largura de um elemento é aplicar a largura e a altura calculadas à **área de conteúdo**, sem levar em consideração qualquer preenchimento, borda e margem.

Esta abordagem provou ser bastante complicada de trabalhar.

Você pode alterar esse comportamento definindo a propriedade `box-sizing`

A propriedade é uma grande ajuda. Tem 2 valores: `box-sizing`

- `border-box`
- `content-box`

`content-box` é o padrão, aquele que tivemos por eras antes se tornou uma coisa `box-sizing`

`border-box` é a nova e grande coisa que estamos procurando. Se você definir isso em um elemento:

```
.my-div {  
  box-sizing: border-box;  
}
```

o cálculo de largura e altura inclui o preenchimento e a borda. Apenas a margem é deixada de fora, o que é razoável, já que em nossa mente também normalmente vemos isso como uma coisa separada: a margem está fora da caixa.

Esta propriedade é uma pequena mudança, mas tem um grande impacto. CSS Tricks até declarou um [dia internacional de conscientização sobre o dimensionamento de caixas](#), apenas dizendo, e recomenda-se aplicá-lo a todos os elementos da página, fora da caixa, com isto:

```
*,  
*:before,  
*:after {  
  box-sizing: border-box;  
}
```

## 27. Exibição

A propriedade de um objeto determina como ele é renderizado pelo navegador.`display`

É uma propriedade muito importante e, provavelmente, aquela com o maior número de valores que você pode usar.

Esses valores incluem:

- `block`
- `inline`
- `none`
- `contents`
- `flow`
- `flow-root`
- `table` (e todos os)`table-*`
- `flex`
- `grid`
- `list-item`
- `inline-block`
- `inline-table`
- `inline-flex`
- `inline-grid`
- `inline-list-item`

além de outros que você provavelmente não usará, como `.ruby`

Escolher qualquer um deles alterará consideravelmente o comportamento do navegador com o elemento e seus filhos.

Nesta seção, analisaremos os mais importantes não abordados em outro lugar:

- `block`
- `inline`
- `inline-block`
- `none`

Veremos alguns dos outros em capítulos posteriores, incluindo a cobertura de , e .tableflexgrid

## 27.1. inline

Inline é o valor de exibição padrão para cada elemento no CSS.

Todas as marcas HTML são exibidas embutidas fora da caixa, exceto alguns elementos como , e , que são definidos como pelo agente do usuário (o navegador).divpsectionblock

Os elementos embutidos não têm nenhuma margem ou preenchimento aplicado.

O mesmo para altura e largura.

Você *pode* adicioná-los, mas a aparência na página não será alterada - eles são calculados e aplicados automaticamente pelo navegador.

## 27.2. inline-block

Semelhante ao , mas com e são aplicados conforme especificado.inlineinline-blockwidthheight

## 27.3. block

Como mencionado, normalmente os elementos são exibidos em linha, com exceção de alguns elementos, incluindo

- div
- p
- section
- ul

que são definidos como pelo navegador.block

Com o , os elementos são empilhados um após o outro, verticalmente, e cada elemento ocupa 100% da página.display: block

Os valores atribuídos às propriedades e são respeitados, se você defini-los, juntamente com e `width height margin padding`

## 27.4. none

O uso faz com que um elemento desapareça. Ele ainda está lá no HTML, mas simplesmente não é visível no navegador.`display: none`

# 28. Posicionamento

O posicionamento é o que nos faz determinar onde os elementos aparecem na tela e como eles aparecem.

Você pode mover elementos e posicioná-los exatamente onde quiser.

Nesta seção, também veremos como as coisas mudam em uma página com base em como elementos diferentes interagem uns com os outros.`position`

Temos uma propriedade CSS principal: `.position`

Ele pode ter esses 5 valores:

- `static`
- `relative`
- `absolute`
- `fixed`
- `sticky`

## 28.1. Posicionamento estático

Esse é o valor padrão para um elemento. Elementos posicionados estáticos são exibidos no fluxo de página normal.

## 28.2. Posicionamento relativo

Se você definir um elemento, agora poderá posicioná-lo com um deslocamento, usando as propriedades `position: relative`

- Início
- Direita
- fundo
- Esquerda

que são chamadas **de propriedades de deslocamento**. Eles aceitam um valor de comprimento ou uma porcentagem.

Veja [este exemplo que fiz no Codepen](#). Eu crio um contêiner pai, um contêiner filho e uma caixa interna com algum texto:

```
<div class="parent">
  <div class="child">
    <div class="box">
      <p>Test</p>
    </div>
  </div>
</div>
```

com algum CSS para dar algumas cores e preenchimento, mas não afeta o posicionamento:

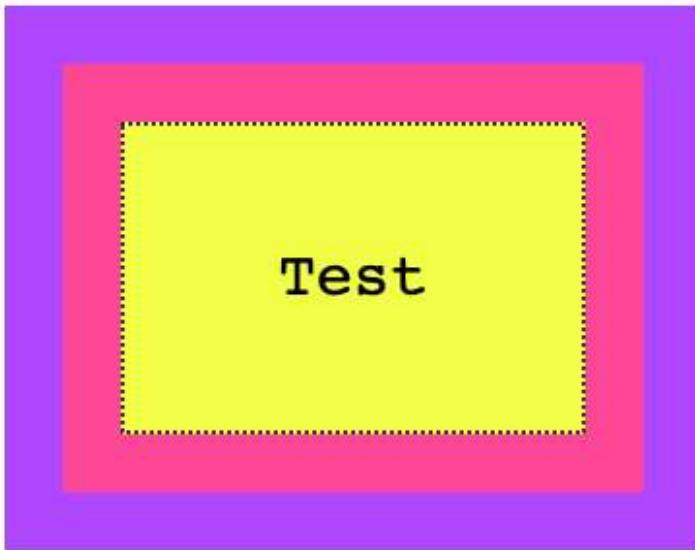
```
.parent {
  background-color: #af47ff;
  padding: 30px;
  width: 300px;
}

.child {
  background-color: #ff4797;
  padding: 30px;
}

.box {
  background-color: #f3ff47;
  padding: 30px;
  border: 2px solid #333;
  border-style: dotted;
```

```
font-family: courier;  
text-align: center;  
font-size: 2rem;  
}
```

here's the result:



You can try and add any of the properties I mentioned before ( , , ) to , and nothing will happen. The position is .toprightbottomleft.boxstatic

Now if we set to the box, at first apparently nothing changes. But the element is now able to move using the , , , properties, and now you can alter the position of it relatively to the element containing it.  
`position: relativetoprightbottomleft`

For example:

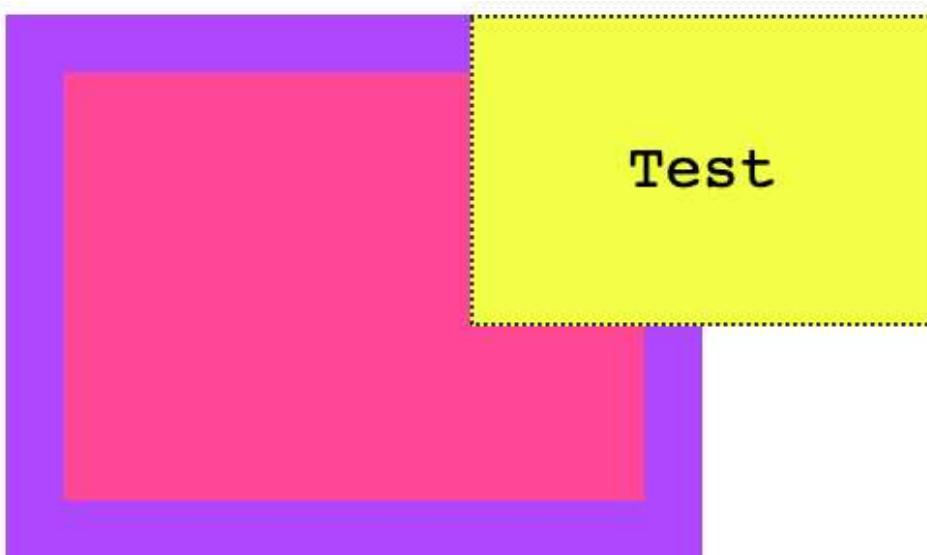
```
.box {  
    /* ... */  
    position: relative;  
    top: -60px;  
}
```



A negative value for `top` will make the box move up relatively to its container.`top`

Ou

```
.box {  
    /* ... */  
    position: relative;  
    top: -60px;  
    left: 180px;  
}
```



Observe como o espaço que é ocupado pela caixa permanece preservado no recipiente, como se ainda estivesse em seu lugar.

Outra propriedade que agora funcionará é alterar o posicionamento do eixo z. Falaremos sobre isso mais adiante.z-index

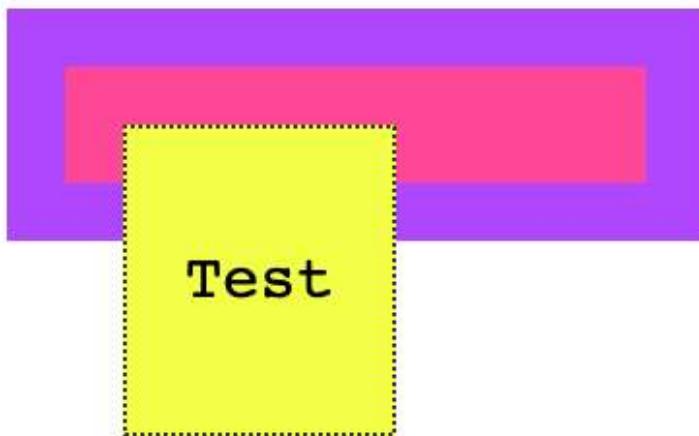
## 28.3. Posicionamento absoluto

A configuração de um elemento o removerá do fluxo do documento e ele não seguirá mais o fluxo de posicionamento da página original.position: absolute

Lembra-se em posicionamento relativo que notamos que o espaço originalmente ocupado por um elemento foi preservado mesmo que fosse movido?

Com o posicionamento absoluto, assim que nos estabelecemos , seu espaço original é agora colapsado, e apenas a origem (coordenadas x, y) permanece a mesma.position: absolute.box

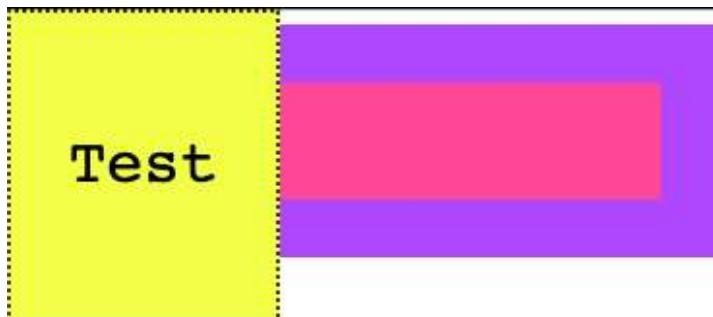
```
.box {  
    /* ... */  
    position: absolute;  
}
```



Agora podemos mover a caixa como quisermos, usando as propriedades , , , :toprightbottomleft

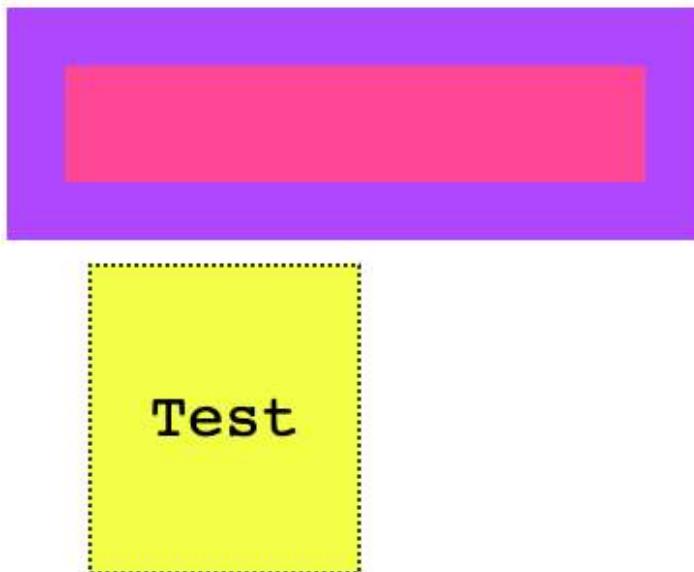
```
.box {  
    /* ... */
```

```
position: absolute;  
top: 0px;  
left: 0px;  
}
```



ou

```
.box {  
/* ... */  
position: absolute;  
top: 140px;  
left: 50px;  
}
```



As coordenadas são relativas ao contêiner mais próximo que não é `.static`

Isso significa que, se adicionarmos ao elemento, e definirmos e para 0, a caixa não será posicionada na margem superior esquerda da *janela*, mas sim nas coordenadas 0, 0 de :position: relative.childtopleft.child

```
.child {  
    /* ... */  
    position: relative;  
}
```

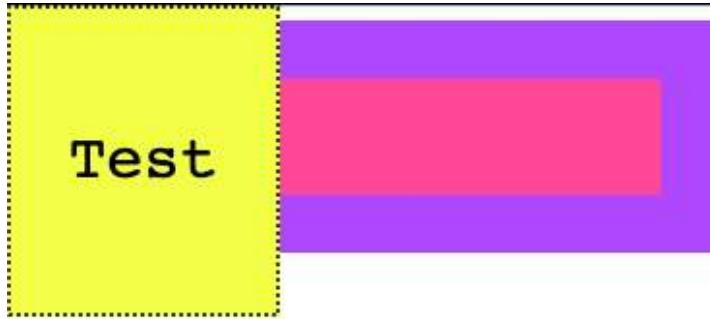
```
.box {  
    /* ... */  
    position: absolute;  
    top: 0px;  
    left: 0px;  
}
```



Veja o que acontece se for estático (o padrão):.child

```
.child {  
    /* ... */  
    position: static;  
}
```

```
.box {  
    /* ... */  
    position: absolute;  
    top: 0px;  
    left: 0px;  
}
```



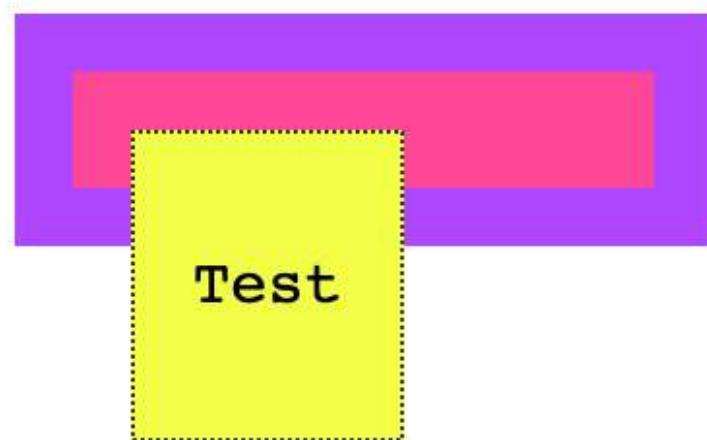
Como para o posicionamento relativo, você pode usar para alterar o posicionamento do eixo z.z-index

## 28.4. Posicionamento fixo

Como no posicionamento absoluto, quando um elemento é atribuído, ele é removido do fluxo da página.`position: fixed`

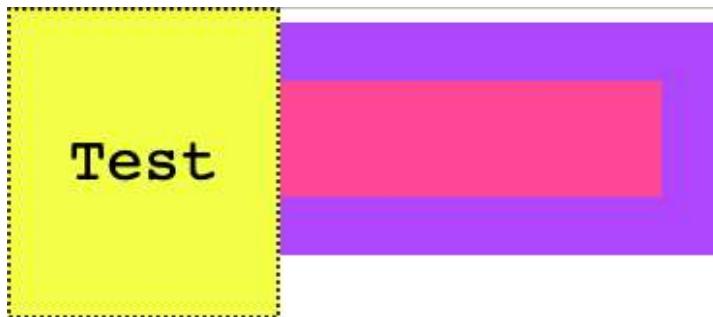
A diferença com o posicionamento absoluto é esta: os elementos agora são sempre posicionados em relação à janela, em vez do primeiro contêiner não estático.

```
.box {  
    /* ... */  
    position: fixed;  
}
```



```
.box {  
    /* ... */  
    position: fixed;
```

```
top: 0;  
left: 0;  
}
```



Outra grande diferença é que os elementos não são afetados pela rolagem. Depois de colocar um elemento pegajoso em algum lugar, rolar a página não o remove da parte visível da página.

## 28.5. Posicionamento pegajoso

Embora os valores acima existam há muito tempo, este foi introduzido recentemente e ainda é relativamente sem suporte ([veja caniuse.com](http://caniuse.com))

O componente UITableView iOS é a coisa que vem à mente quando penso em . Sabe quando você rola na lista de contatos e a primeira letra é colada no topo, para que você saiba que está visualizando os contatos dessa carta em particular?position: sticky

Usamos JavaScript para emular isso, mas essa é a abordagem adotada pelo CSS para permitir isso nativamente.

## 29. Flutuação e limpeza

A flutuação tem sido um tópico muito importante no passado.

Ele foi usado em muitos hacks e usos criativos porque era uma das poucas maneiras, juntamente com tabelas, que poderíamos realmente implementar alguns layouts. No passado, costumávamos flutuar a barra lateral para a esquerda, por exemplo, para mostrá-la no lado esquerdo da tela e adicionamos alguma margem ao conteúdo principal.

Felizmente, os tempos mudaram e hoje temos o Flexbox e o Grid para nos ajudar com o layout, e o float voltou ao seu escopo original: colocar conteúdo em um lado do elemento do contêiner e fazer com que seus irmãos apareçam em torno dele.

A propriedade oferece suporte a 3 valores:`float`

- `left`
- `right`
- `none` (o padrão)

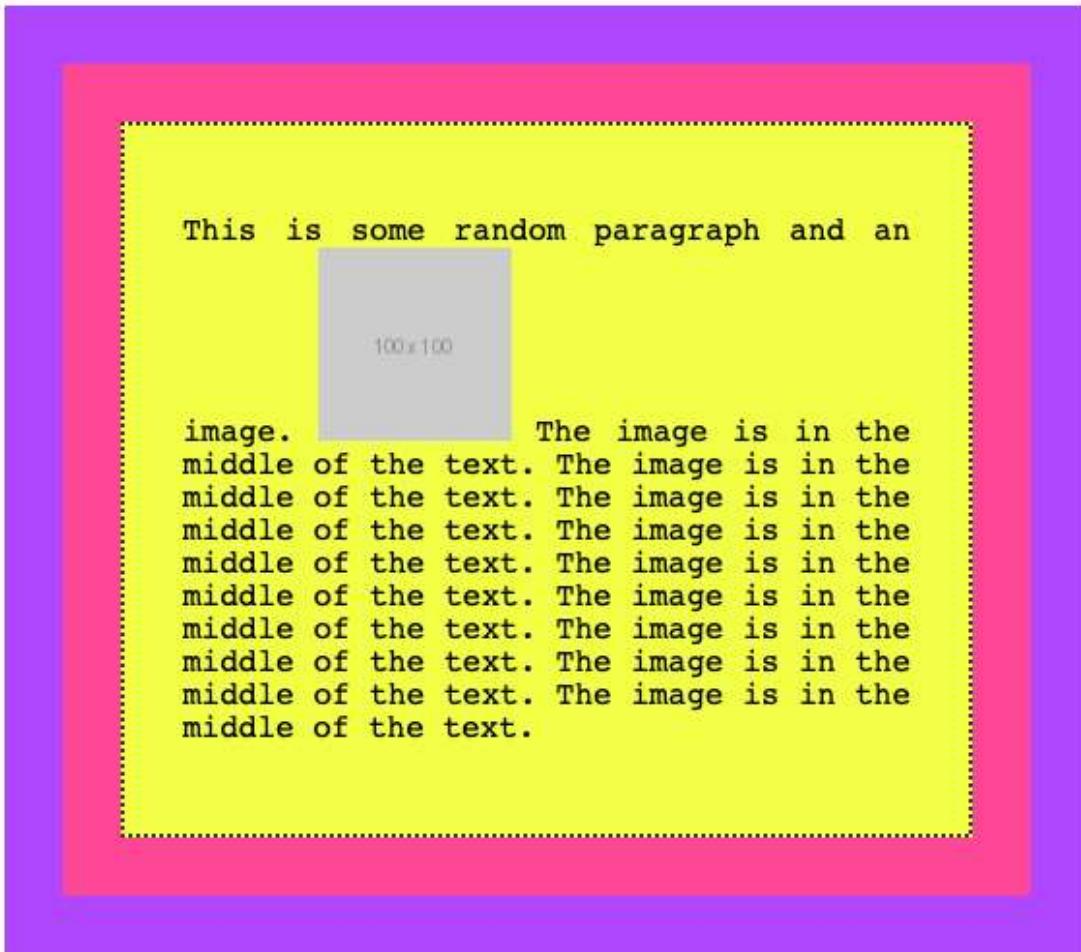
Digamos que temos uma caixa que contém um parágrafo com algum texto, e o parágrafo também contém uma imagem.

Aqui está um pouco de código:

```
<div class="parent">
  <div class="child">
    <div class="box">
      <p>
        This is some random paragraph and an image.
         The image is
        in the
        middle of the text. The image is in the middle of the text. The
        image is
        in the middle of the text. The image is in the middle of the
        text. The
        image is in the middle of the text. The image is in the middle
        of the
        text. The image is in the middle of the text. The image is in
        the middle
        of the text. The image is in the middle of the text.
      </p>
    </div>
  </div>
</div>
```

```
.parent {  
    background-color: #af47ff;  
    padding: 30px;  
    width: 500px;  
}  
  
.child {  
    background-color: #ff4797;  
    padding: 30px;  
}  
  
.box {  
    background-color: #f3ff47;  
    padding: 30px;  
    border: 2px solid #333;  
    border-style: dotted;  
    font-family: courier;  
    text-align: justify;  
    font-size: 1rem;  
}
```

e a aparência visual:

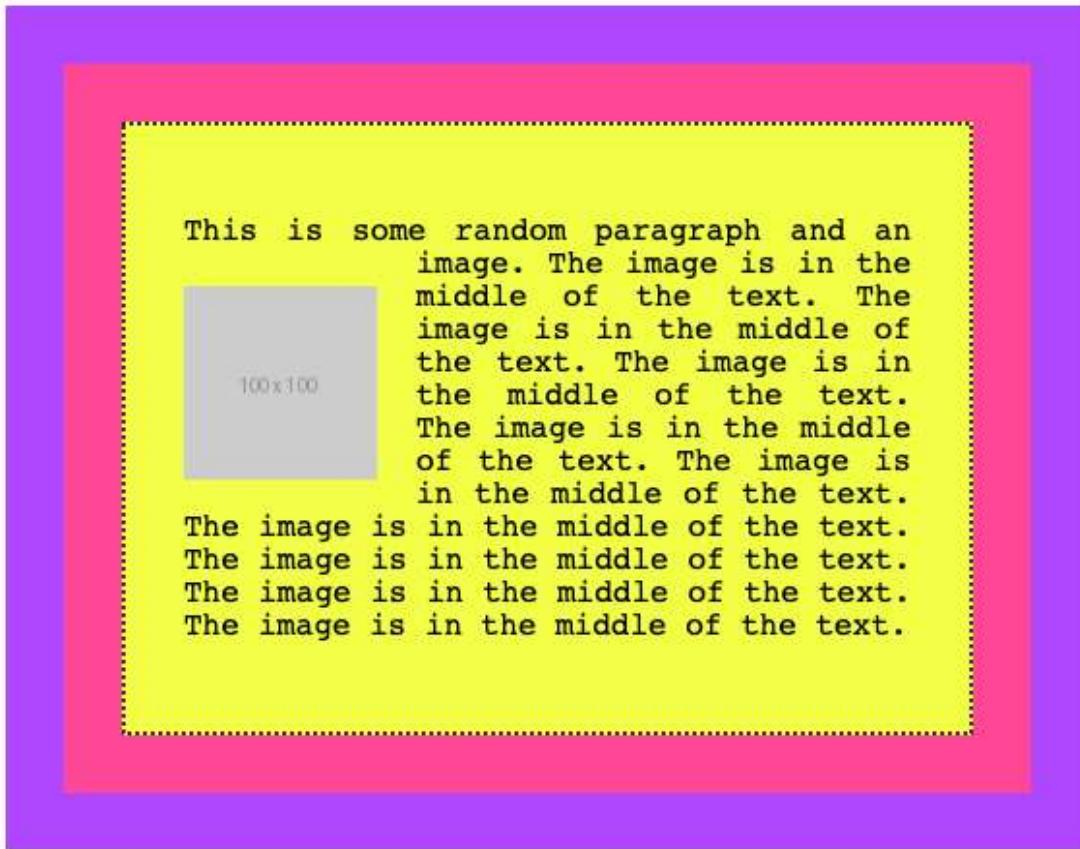


Como você pode ver, o fluxo normal por padrão considera a imagem embutida e abre espaço para ela na própria linha.

Se adicionarmos à imagem, e algum preenchimento:`float: left`

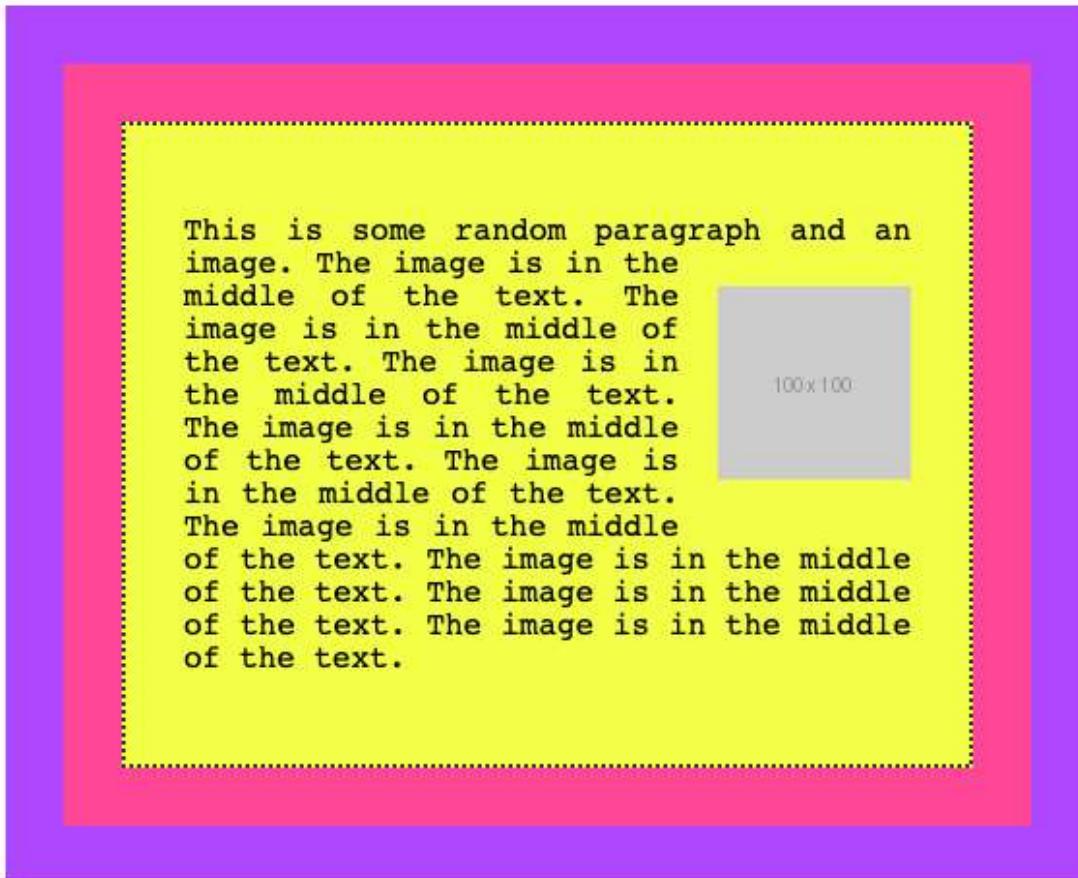
```
img {  
    float: left;  
    padding: 20px 20px 0px 0px;  
}
```

este é o resultado:



e é isso que obtemos aplicando um flutuador: à direita, ajustando o preenchimento de acordo:

```
img {  
    float: right;  
    padding: 20px 0px 20px 20px;  
}
```



Um elemento flutuante é removido do fluxo normal da página e o outro conteúdo flui em torno dele.

[Veja o exemplo no Codepen](#)

Você também não está limitado a imagens flutuantes. Aqui nós trocamos a imagem com um elemento:span

```
<div class="parent">
  <div class="child">
    <div class="box">
      <p>
        This is some random paragraph and an image.
        <span>Some text to float</span> The image is in the middle of
        the text.

        The image is in the middle of the text. The image is in the
        middle of

        the text. The image is in the middle of the text. The image is
        in the

        middle of the text. The image is in the middle of the text. The
```

```
image is
```

```
    in the middle of the text. The image is in the middle of the  
text. The
```

```
        image is in the middle of the text.
```

```
</p>
```

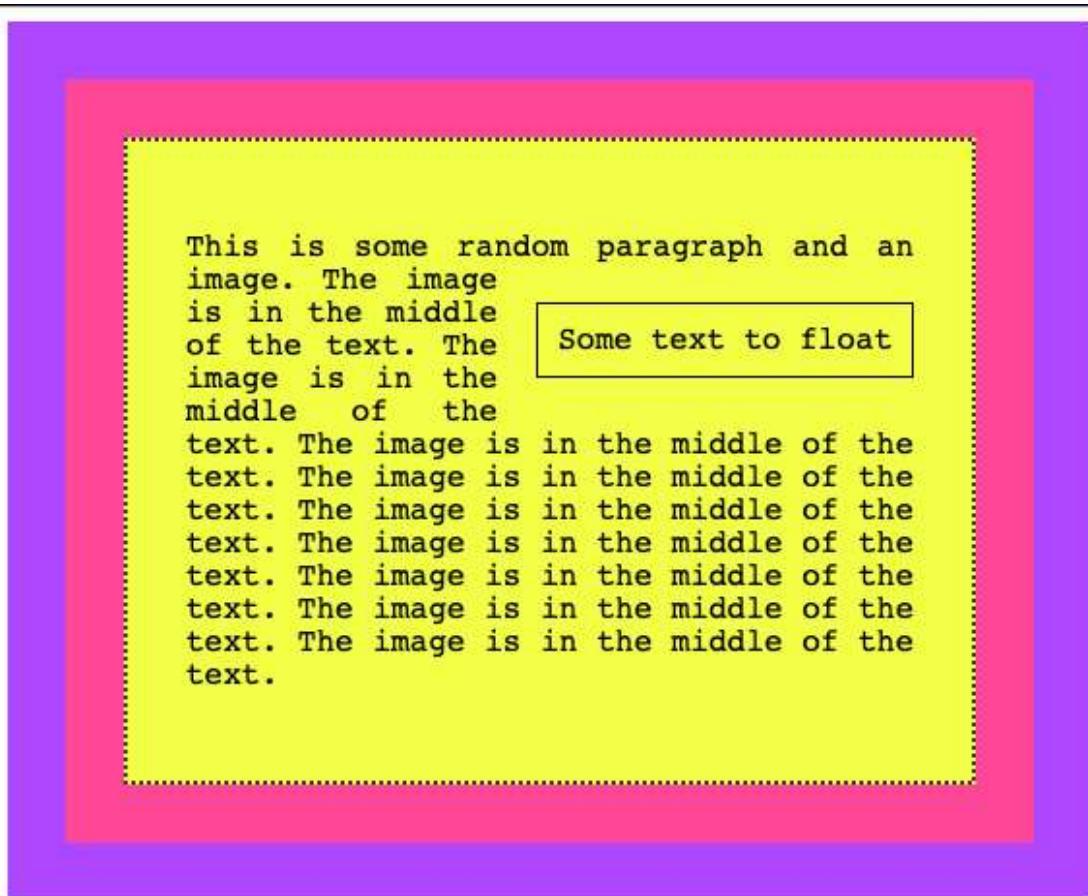
```
</div>
```

```
</div>
```

```
</div>
```

```
span {  
    float: right;  
    margin: 20px 0px 20px 20px;  
    padding: 10px;  
    border: 1px solid black;  
}
```

e este é o resultado:

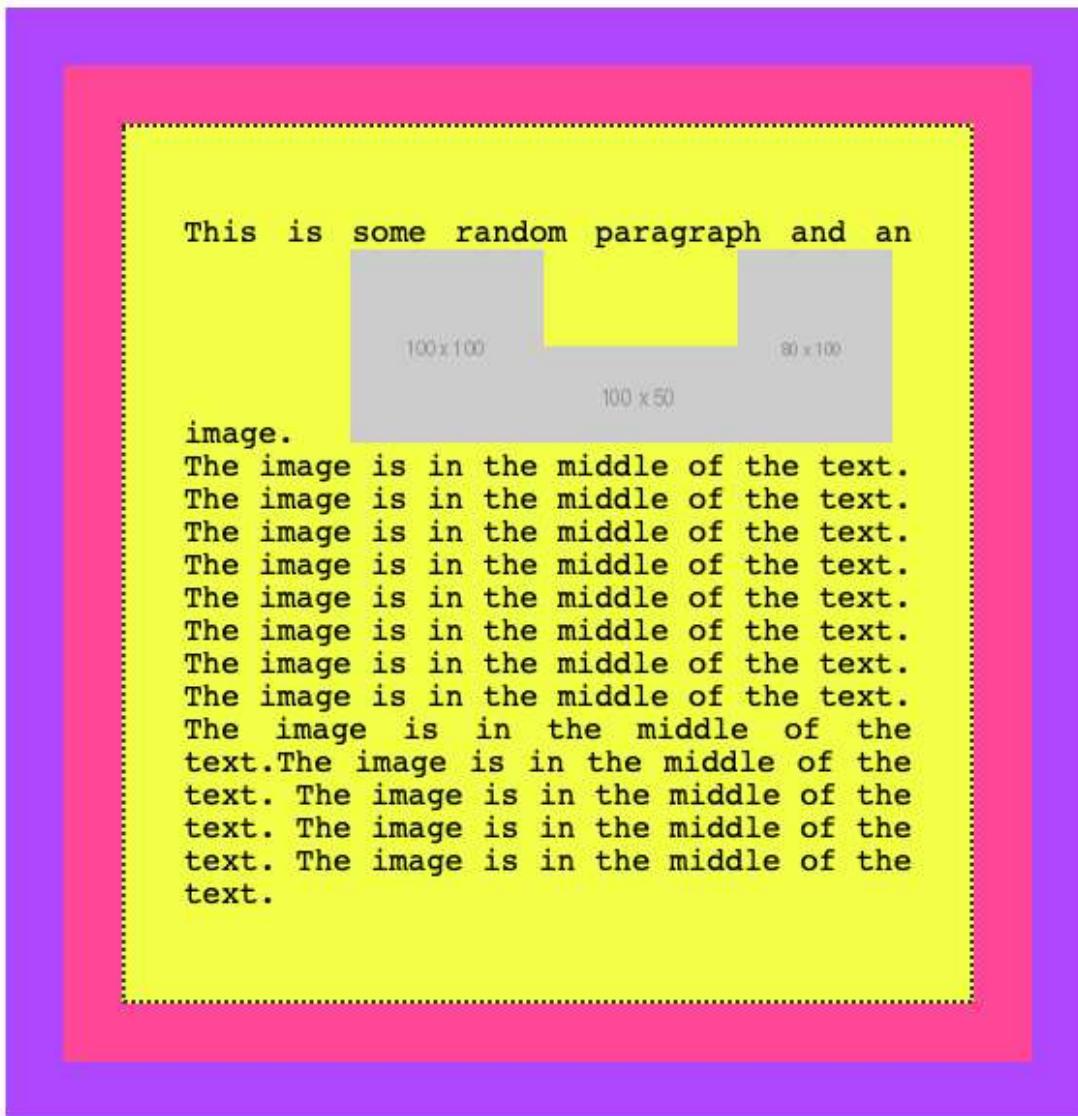


## 29.1. Compensação

O que acontece quando você flutua mais de um elemento?

Se quando flutuados eles encontrarem outra imagem flutuada, por padrão eles são empilhados um ao lado do outro, horizontalmente. Até que não haja espaço, e eles começarão a ser empilhados em uma nova linha.

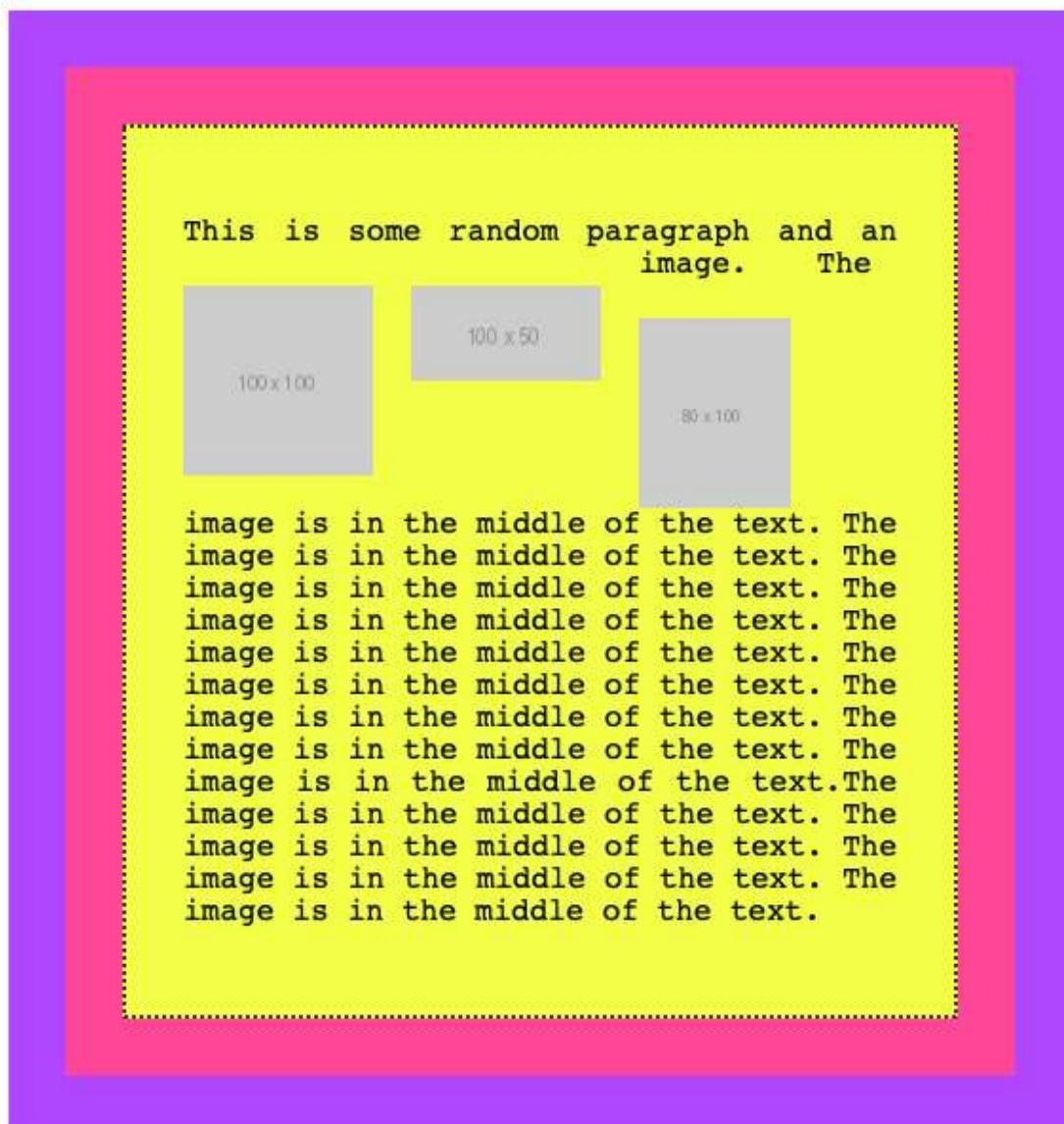
Digamos que tivéssemos 3 imagens embutidas dentro de uma tag:p



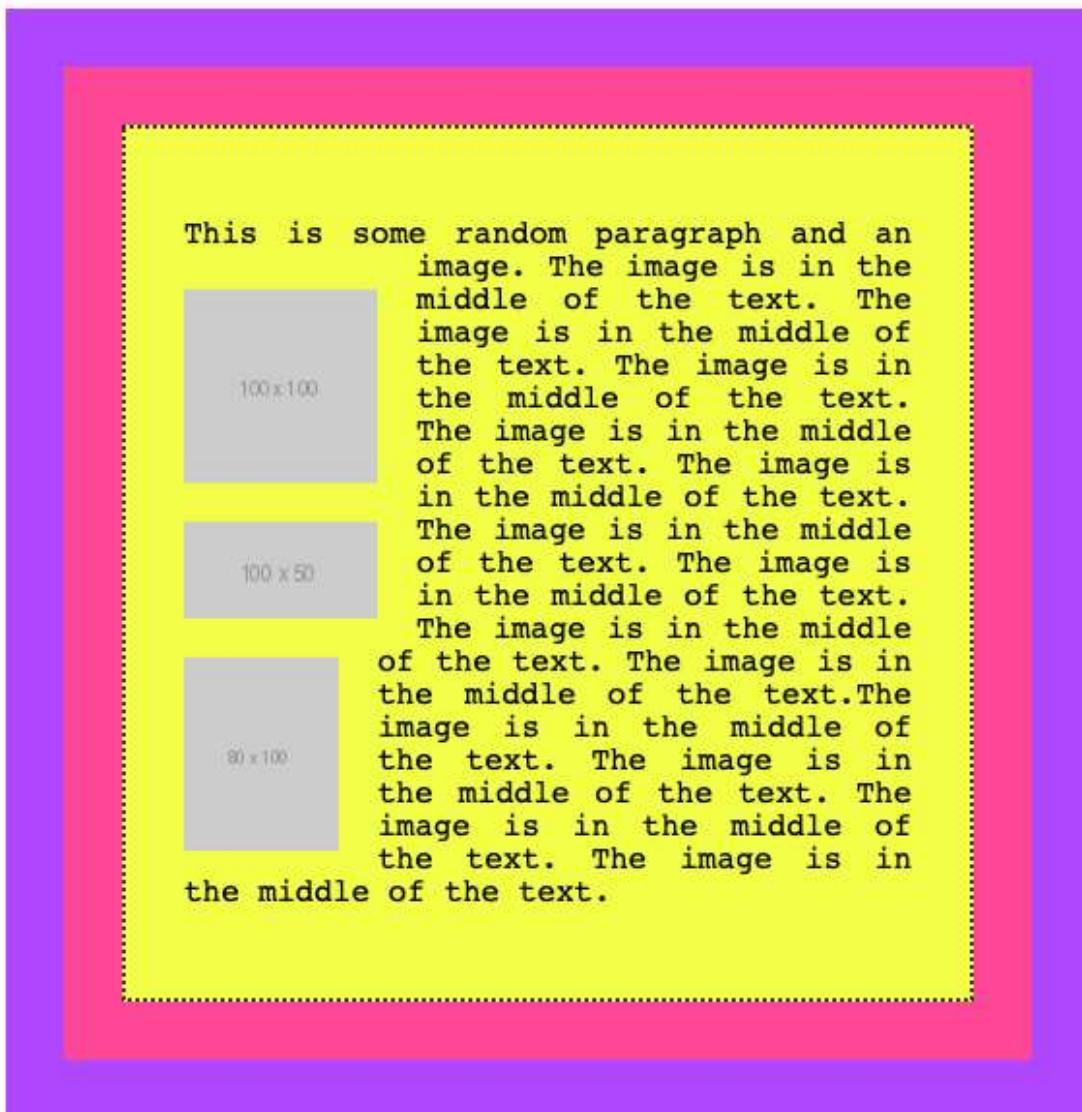
Se acrescentarmos a essas imagens:float: left

```
img {  
  float: left;  
  padding: 20px 20px 0px 0px;  
}
```

é isso que teremos:



se você adicionar às imagens, elas serão empilhadas verticalmente em vez de horizontalmente:  
`clear: left`



Usei o valor para . Permiteleftclear

- left para limpar flutuadores à esquerda
  - right para limpar flutuadores à direita
  - both para limpar os flutuadores esquerdo e direito
  - none (padrão) desabilita a compensação

30. índice z

Quando falamos sobre posicionamento, mencionei que você pode usar a propriedade para controlar o posicionamento do eixo Z dos elementos.`z-index`

É muito útil quando você tem vários elementos que se sobrepõem uns aos outros, e você precisa decidir qual deles é visível, o mais próximo do usuário, e qual deles deve estar escondido atrás dele.

Essa propriedade usa um número (sem decimais) e usa esse número para calcular quais elementos aparecem mais próximos do usuário, no eixo Z.

Quanto maior o valor do índice z, mais um elemento é posicionado mais próximo do usuário.

Ao decidir qual elemento deve ser visível e qual deve ser posicionado atrás dele, o navegador faz um cálculo no valor do índice z.

O valor padrão é , uma palavra-chave especial. Usando , a ordem do eixo Z é determinada pela posição do elemento HTML na página - o último irmão aparece primeiro, como é definido por `último.autoauto`

Por padrão, os elementos têm o valor da propriedade. Nesse caso, a propriedade não faz nenhuma diferença - ela deve ser definida como , ou para funcionar.`staticpositionz-indexabsoluterelativefixed`

Exemplo:

```
.my-first-div {  
    position: absolute;  
    top: 0;  
    left: 0;  
    width: 600px;  
    height: 600px;  
    z-index: 10;  
}
```

```
.my-second-div {  
    position: absolute;  
    top: 0;  
    left: 0;  
    width: 500px;  
    height: 500px;  
    z-index: 20;  
}
```

O elemento com classe será exibido e, atrás dele, ..my-second-div .my-first-div

Aqui usamos 10 e 20, mas você pode usar qualquer número. Números negativos também. É comum escolher números não consecutivos, para que você possa posicionar elementos no meio. Se você usar números consecutivos, precisará recalcular o índice-z de cada elemento envolvido no posicionamento.

## 31.CSS Grade

O CSS Grid é o novo garoto da cidade CSS e, embora ainda não seja totalmente suportado por todos os navegadores, será o futuro sistema para layouts.

O CSS Grid é uma abordagem fundamentalmente nova para a criação de layouts usando CSS.

Fique de olho na página Layout de grade CSS no caniuse.com (<https://caniuse.com/#feat=css-grid>) para descobrir quais navegadores atualmente o suportam. No momento em que escrevo, abril de 2019, todos os principais navegadores (exceto o IE, que nunca terá suporte para isso) já estão suportando essa tecnologia, cobrindo 92% de todos os usuários.

O CSS Grid não é um concorrente do Flexbox. Eles interoperam e colaboram em layouts complexos, porque o CSS Grid funciona em 2 dimensões (linhas E colunas), enquanto o Flexbox funciona em uma única dimensão (linhas OU colunas).

A criação de layouts para a Web tem sido tradicionalmente um tópico complicado.

Não vou me aprofundar nas razões para essa complexidade, que é um tópico complexo por si só, mas você pode se pensar como um ser humano muito sortudo, porque hoje em dia você tem 2 ferramentas muito poderosas e bem suportadas à sua disposição:

- **CSS Flexbox**
- **Grade CSS**

Estes 2 são as ferramentas para construir os layouts da Web do futuro.

A menos que você precise suportar navegadores antigos como IE8 e IE9, não há razão para mexer com coisas como:

- Layouts de tabela
- Flutua
- clearfix hacks
- display: table Hacks

Neste guia, há tudo o que você precisa saber sobre como passar de um conhecimento zero do CSS Grid para ser um usuário proficiente.

## 31.1. O básico

O layout CSS Grid é ativado em um elemento de contêiner (que pode ser uma ou qualquer outra tag) definindo .divdisplay: grid

Assim como no flexbox, você pode definir algumas propriedades no contêiner e algumas propriedades em cada item individual na grade.

Essas propriedades combinadas determinarão a aparência final da grade.

As propriedades de contêiner mais básicas são e .grid-template-columnsgrid-template-rows

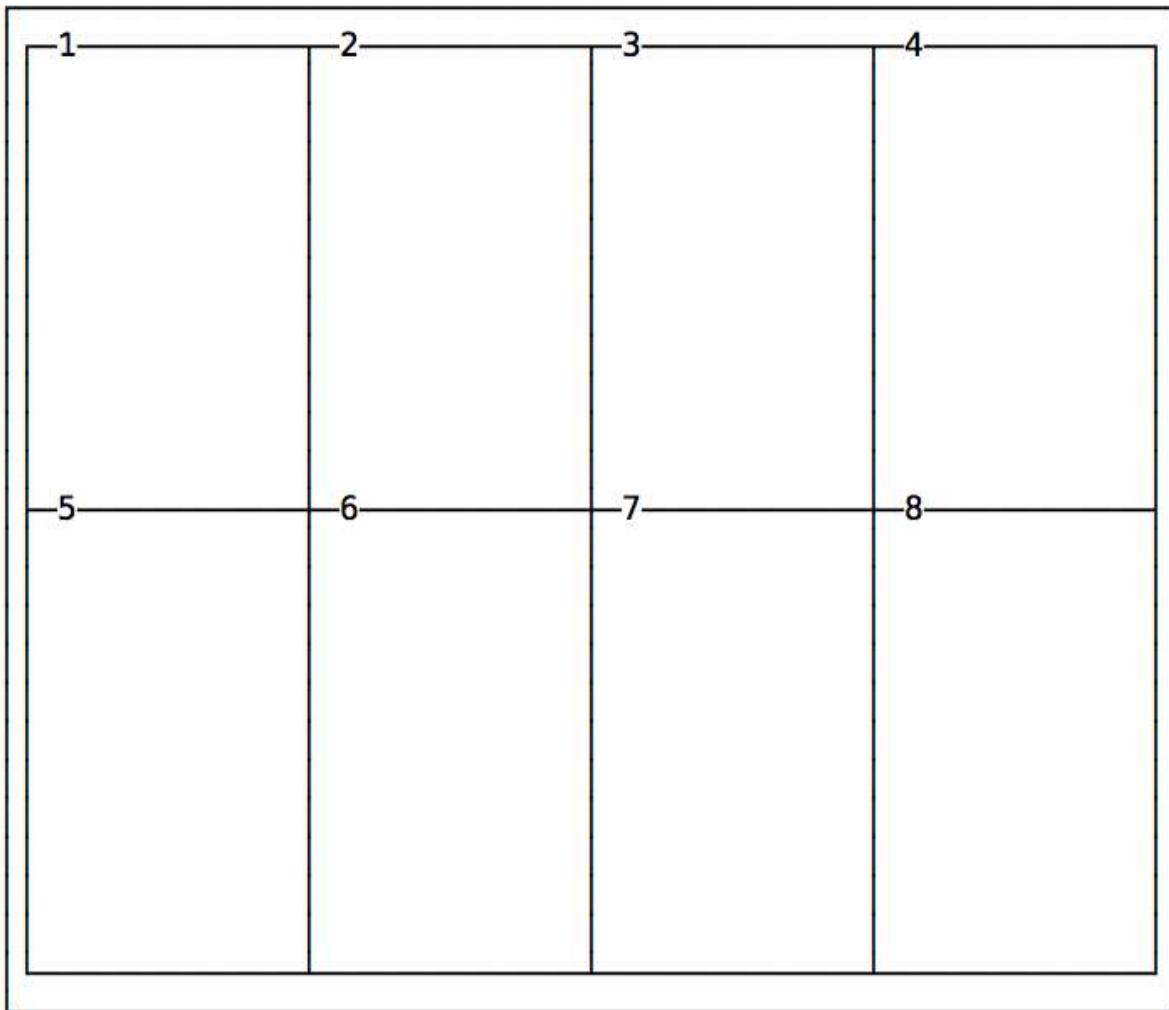
### 31.1.1. grid-template-columns e grid-template-rows

Essas propriedades definem o número de colunas e linhas na grade e também definem a largura de cada coluna/linha.

O trecho a seguir define uma grade com 4 colunas cada uma com 200px de largura e 2 linhas com uma altura de 300px cada.

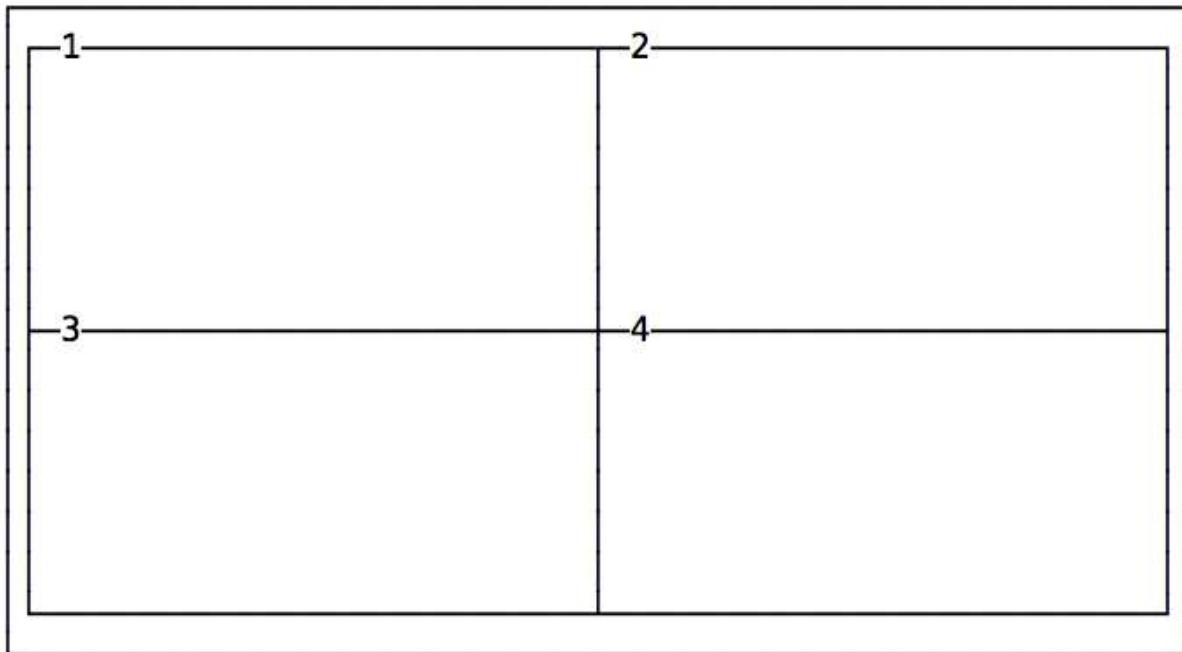
```
.container {  
    display: grid;  
    grid-template-columns: 200px 200px 200px 200px;
```

```
grid-template-rows: 300px 300px;  
}
```



Aqui está outro exemplo de uma grade com 2 colunas e 2 linhas:

```
.container {  
    display: grid;  
    grid-template-columns: 200px 200px;  
    grid-template-rows: 100px 100px;  
}
```



### 31.1.2. Dimensões automáticas

Muitas vezes você pode ter um tamanho de cabeçalho fixo, um tamanho de rodapé fixo e o conteúdo principal que é flexível em altura, dependendo de seu comprimento. Neste caso, você pode usar a palavra-chave: `auto`

```
.container {  
    display: grid;  
    grid-template-rows: 100px auto 100px;  
}
```

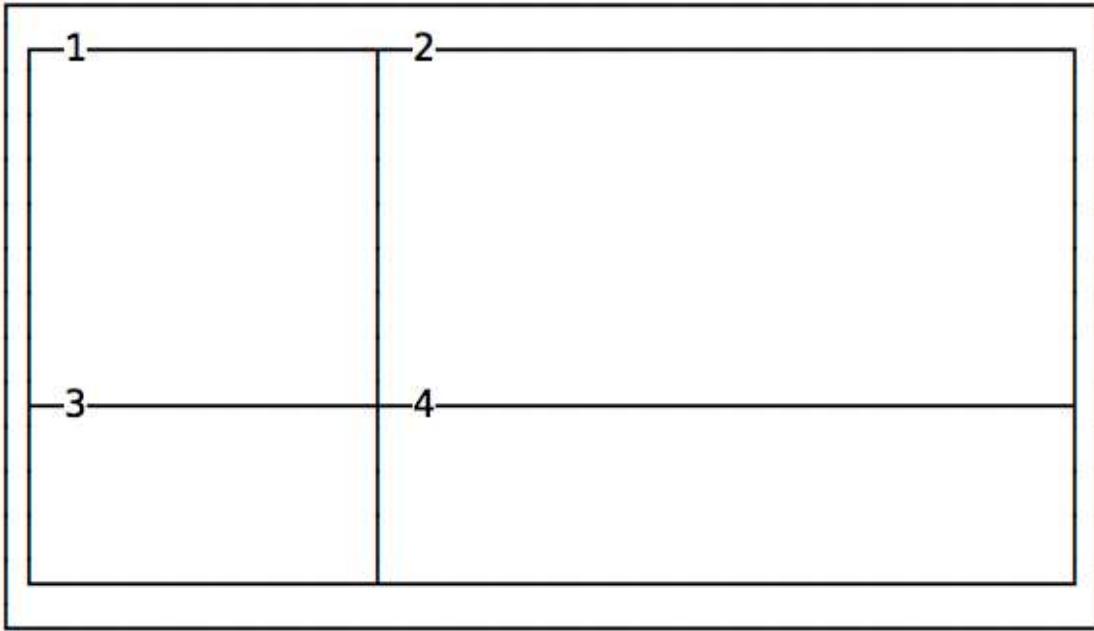
### 31.1.3. Diferentes dimensões de colunas e linhas

Nos exemplos acima, fizemos grades regulares usando os mesmos valores para linhas e os mesmos valores para colunas.

Você pode especificar qualquer valor para cada linha/coluna, para criar muitos designs diferentes:

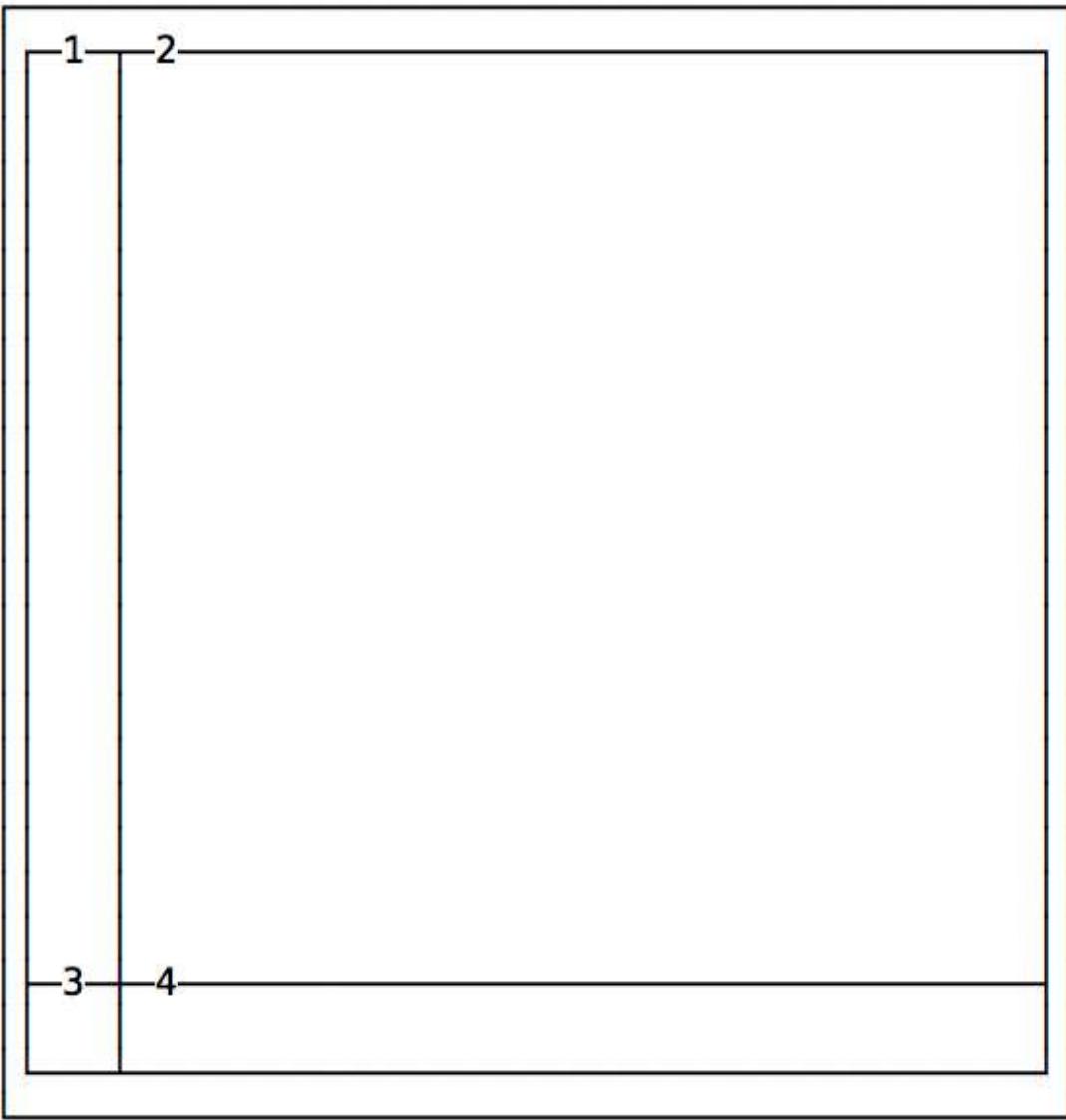
```
.container {  
    display: grid;  
    grid-template-columns: 100px 200px;
```

```
grid-template-rows: 100px 50px;  
}
```



Outro exemplo:

```
.container {  
    display: grid;  
    grid-template-columns: 10px 100px;  
    grid-template-rows: 100px 10px;  
}
```



### 31.1.4. Adicionando espaço entre as células

A menos que especificado, não há espaço entre as células.

Você pode adicionar espaçamento usando essas propriedades:

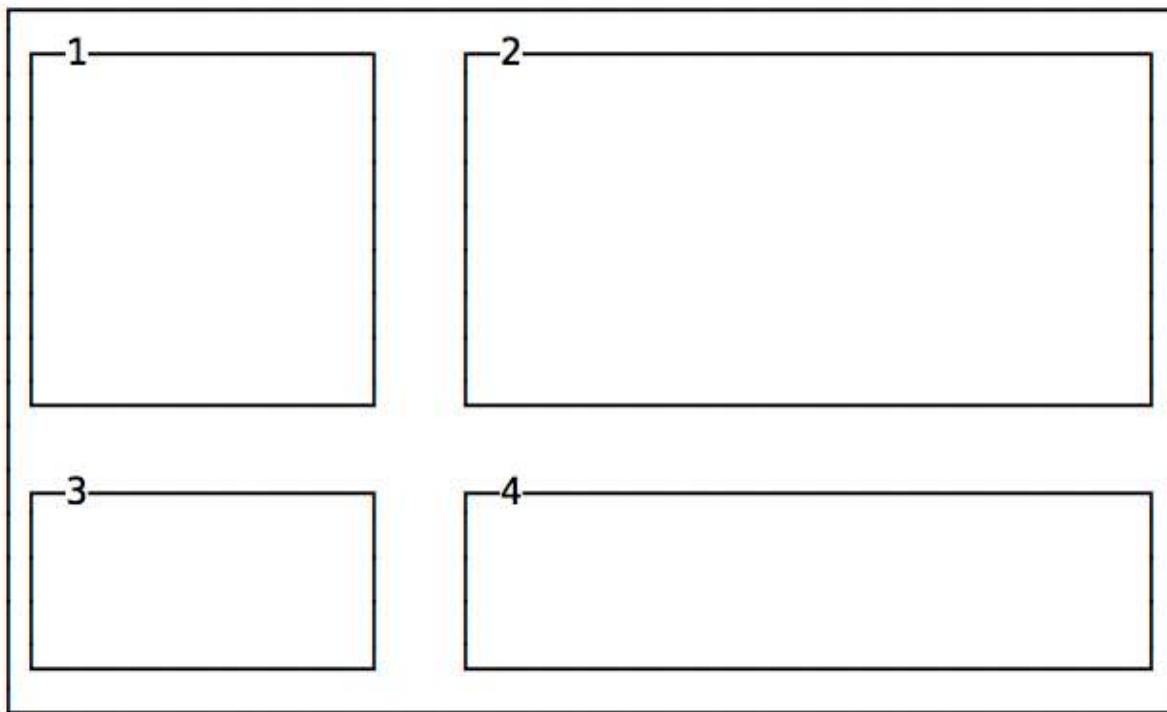
- `grid-column-gap`
- `grid-row-gap`

ou a sintaxe abreviada `.grid-gap`

Exemplo:

```
.container {  
    display: grid;
```

```
grid-template-columns: 100px 200px;  
grid-template-rows: 100px 50px;  
grid-column-gap: 25px;  
grid-row-gap: 25px;  
}
```



O mesmo layout usando a taquigrafia:

```
.container {  
  display: grid;  
  grid-template-columns: 100px 200px;  
  grid-template-rows: 100px 50px;  
  grid-gap: 25px;  
}
```

### 31.1.5. Itens de desova em várias colunas e/ou linhas

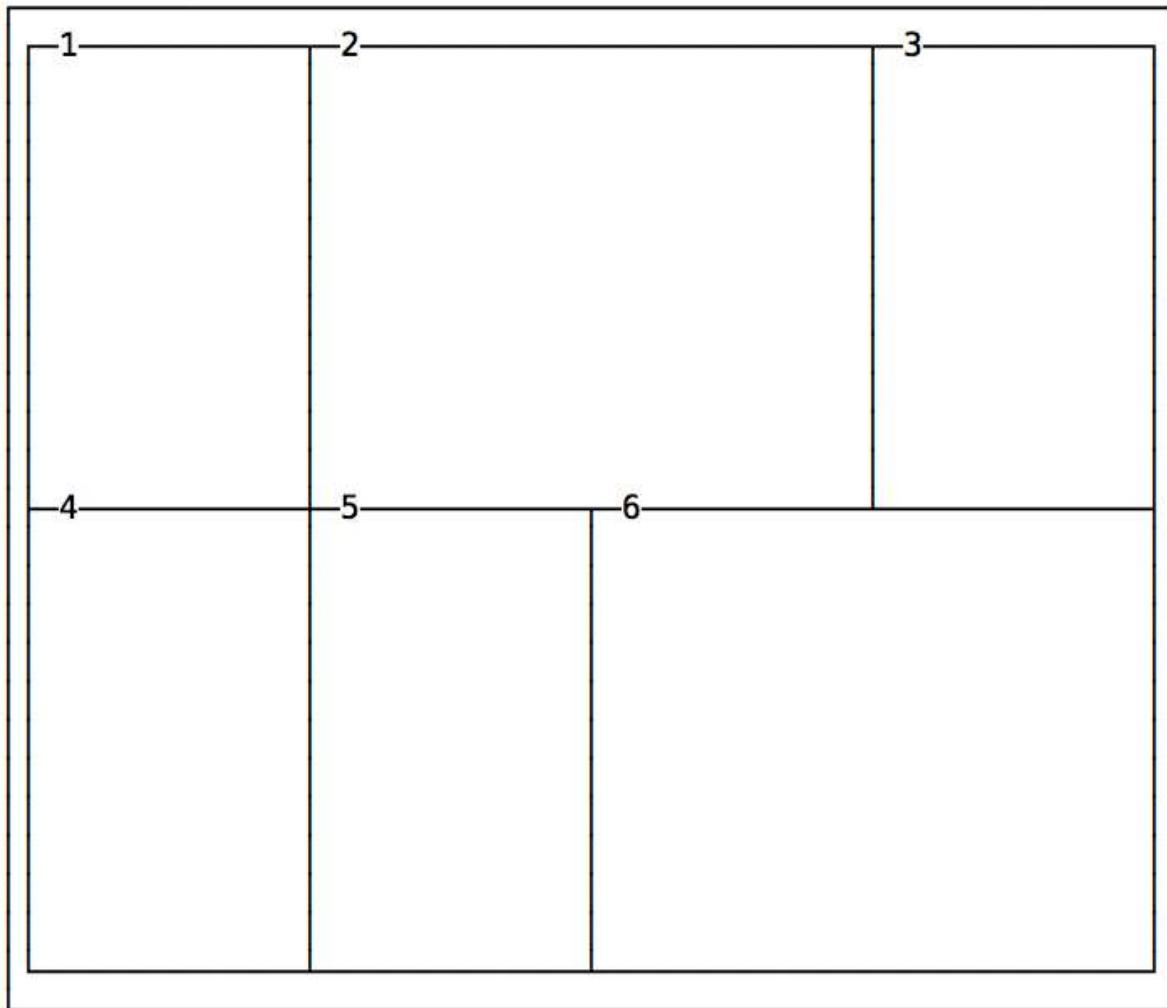
Cada item de célula tem a opção de ocupar mais do que apenas uma caixa na linha e expandir horizontal ou verticalmente para obter mais espaço, respeitando as proporções de grade definidas no contêiner.

Essas são as propriedades que usaremos para isso:

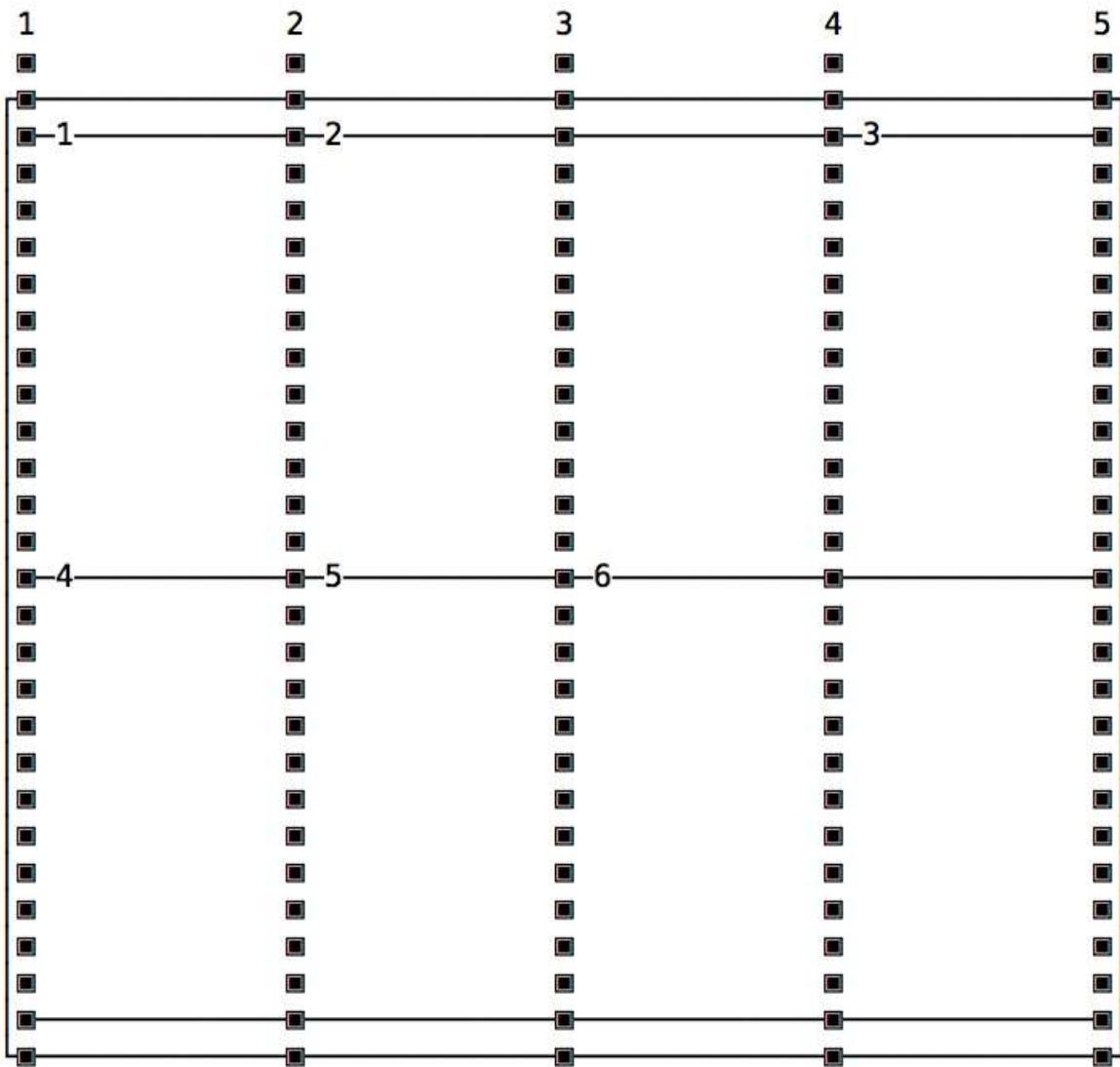
- `grid-column-start`
- `grid-column-end`
- `grid-row-start`
- `grid-row-end`

Exemplo:

```
.container {  
    display: grid;  
    grid-template-columns: 200px 200px 200px 200px;  
    grid-template-rows: 300px 300px;  
}  
.item1 {  
    grid-column-start: 2;  
    grid-column-end: 4;  
}  
.item6 {  
    grid-column-start: 3;  
    grid-column-end: 5;  
}
```



The numbers correspond to the vertical line that separates each column, starting from 1:



The same principle applies to `grid-column-start`, `grid-column-end`, `grid-row-start`, and `grid-row-end`, except this time instead of taking more columns, a cell takes more rows.

### 31.1.5.1. Shorthand syntax

Those properties have a shorthand syntax provided by:

- `grid-column`
- `grid-row`

The usage is simple, here's how to replicate the above layout:

```
.container {  
  display: grid;  
  grid-template-columns: 200px 200px 200px 200px;  
  grid-template-rows: 300px 300px;
```

```
}
```

```
.item1 {
```

```
    grid-column: 2 / 4;
```

```
}
```

```
.item6 {
```

```
    grid-column: 3 / 5;
```

```
}
```

Outra abordagem é definir a coluna/linha inicial e definir quantas elas devem ocupar usando:span

```
.container {
```

```
    display: grid;
```

```
    grid-template-columns: 200px 200px 200px 200px;
```

```
    grid-template-rows: 300px 300px;
```

```
}
```

```
.item1 {
```

```
    grid-column: 2 / span 2;
```

```
}
```

```
.item6 {
```

```
    grid-column: 3 / span 2;
```

```
}
```

## 31.2. Mais configuração de grade

### 31.2.1. Utilização de frações

Especificar a largura exata de cada coluna ou linha não é ideal em todos os casos.

Uma fração é uma unidade de espaço.

O exemplo a seguir divide uma grade em 3 colunas com a mesma largura, 1/3 do espaço disponível cada.

```
.container {
```

```
    grid-template-columns: 1fr 1fr 1fr;
```

}

## 31.2.2. Utilização de percentagens e rem

Você também pode usar porcentagens e misturar e combinar frações, pixels, rem e porcentagens:

```
.container {  
    grid-template-columns: 3rem 15% 1fr 2fr;  
}
```

## 31.2.3. Utilização repeat()

repeat() é uma função especial que usa um número que indica o número de vezes que uma linha/coluna será repetida e o comprimento de cada uma.

Se cada coluna tiver a mesma largura, você poderá especificar o layout usando esta sintaxe:

```
.container {  
    grid-template-columns: repeat(4, 100px);  
}
```

Isso cria 4 colunas com a mesma largura.

Ou usando frações:

```
.container {  
    grid-template-columns: repeat(4, 1fr);  
}
```

## 31.2.4. Especificar uma largura mínima para uma linha

Caso de uso comum: tenha uma barra lateral que nunca recolha mais do que uma certa quantidade de pixels quando você redimensiona a janela.

Aqui está um exemplo em que a barra lateral leva 1/4 da tela e nunca leva menos de 200px:

```
.container {  
    grid-template-columns: minmax(200px, 3fr) 9fr;  
}
```

Você também pode definir apenas um valor máximo usando a palavra-chave: auto

```
.container {  
    grid-template-columns: minmax(auto, 50%) 9fr;  
}
```

ou apenas um valor mínimo:

```
.container {  
    grid-template-columns: minmax(100px, auto) 9fr;  
}
```

### 31.2.5. Elementos de posicionamento utilizando grid-template-areas

Por padrão, os elementos são posicionados na grade usando sua ordem na estrutura HTML.

Usando Você pode definir áreas de modelo para movê-las na grade e também para gerar um item em várias linhas / colunas em vez de usar .grid-template-areasgrid-column

Veja um exemplo:

```
<div class="container">  
    <main>...</main>  
    <aside>...</aside>  
    <header>...</header>
```

```
<footer>...</footer>  
</div>  
  
.container {  
    display: grid;  
    grid-template-columns: 200px 200px 200px 200px;  
    grid-template-rows: 300px 300px;  
    grid-template-areas:  
        'header header header header'  
        'sidebar main main main'  
        'footer footer footer footer';  
}  
  
main {  
    grid-area: main;  
}  
  
aside {  
    grid-area: sidebar;  
}  
  
header {  
    grid-area: header;  
}  
  
footer {  
    grid-area: footer;  
}
```

Apesar de sua ordem original, os itens são colocados onde se definem, dependendo da propriedade associada a eles.`grid-template-areas``grid-area`

### 31.2.5.1. Adicionando células vazias em áreas de modelo

Você pode definir uma célula vazia usando o ponto em vez de um nome de área em: `.grid-template-areas`

```
.container {  
    display: grid;  
    grid-template-columns: 200px 200px 200px 200px;
```

```

grid-template-rows: 300px 300px;
grid-template-areas:
  '. header header .'
  'sidebar . main main'
  '. footer footer .';
}

```

## 31.3. Preencher uma página com uma grade

Você pode fazer uma grade se estender para preencher a página usando:`fr`

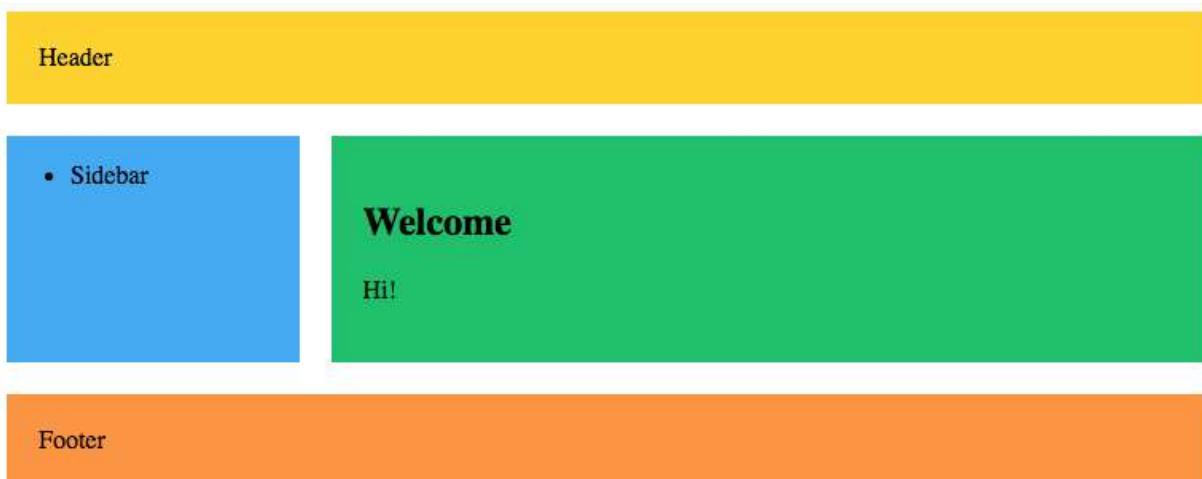
```

.container {
  display: grid;
  height: 100vh;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
}

```

## 31.4. Um exemplo: cabeçalho, barra lateral, conteúdo e rodapé

Aqui está um exemplo simples de como usar o CSS Grid para criar um layout de site que fornece um cabeçalho op top, uma parte principal com barra lateral à esquerda e conteúdo à direita, e um rodapé depois.



Aqui está a marcação:

```
<div class="wrapper">
  <header>Header</header>
  <article>
    <h1>Welcome</h1>
    <p>Hi!</p>
  </article>
  <aside>
    <ul>
      <li>Sidebar</li>
    </ul>
  </aside>
  <footer>Footer</footer>
</div>
```

e aqui está o CSS:

```
header {
  grid-area: header;
  background-color: #fed330;
  padding: 20px;
}

article {
  grid-area: content;
  background-color: #20bf6b;
  padding: 20px;
}

aside {
  grid-area: sidebar;
  background-color: #45aaf2;
}

footer {
  padding: 20px;
  grid-area: footer;
  background-color: #fd9644;
}

.wrapper {
  display: grid;
  grid-gap: 20px;
```

```
grid-template-columns: 1fr 3fr;  
grid-template-areas:  
  'header header'  
  'sidebar content'  
  'footer footer';  
}
```

Eu adicionei algumas cores para torná-lo mais bonito, mas basicamente ele atribui a cada tag diferente um nome, que é usado na propriedade em `.grid-area` `grid-template-areas` `.wrapper`

Quando o layout é menor, podemos colocar a barra lateral abaixo do conteúdo usando uma consulta de mídia:

```
@media (max-width: 500px) {  
  .wrapper {  
    grid-template-columns: 4fr;  
    grid-template-areas:  
      'header'  
      'content'  
      'sidebar'  
      'footer';  
  }  
}
```

[Veja no CodePen](#)

## 31.5. Conclusão

Estes são os conceitos básicos do CSS Grid. Há muitas coisas que eu não incluí nesta introdução, mas eu queria torná-lo muito simples, para começar a usar este novo sistema de layout sem fazê-lo parecer esmagador.

## 32. Caixa flexível

O Flexbox, também chamado de Flexible Box Module, é um dos dois sistemas de layouts modernos, juntamente com o CSS Grid.

Em comparação com o CSS Grid (que é bidimensional), o flexbox é um **modelo de layout unidimensional**. Ele controlará o layout com base em uma linha ou em uma coluna, mas não juntos ao mesmo tempo.

O principal objetivo do flexbox é permitir que os itens preencham todo o espaço oferecido pelo seu recipiente, dependendo de algumas regras que você definir.

A menos que você precise suportar navegadores antigos como IE8 e IE9, o Flexbox é a ferramenta que permite que você esqueça o uso

- Layouts de tabela
- Flutua
- clearfix hacks
- display: table Hacks

Vamos mergulhar no flexbox e nos tornar um mestre dele em um tempo muito curto.

## 32.1. Suporte do navegador

No momento da redação deste artigo (fevereiro de 2018), ele é suportado por 97,66% dos usuários, todos os navegadores mais importantes o implementam desde anos, portanto, mesmo os navegadores mais antigos (incluindo o IE10 +) são cobertos:



Embora devamos esperar alguns anos para que os usuários acompanhem o CSS Grid, o Flexbox é uma tecnologia mais antiga e pode ser usada agora.

## 32.2. Ativar o Flexbox

Um layout flexbox é aplicado a um contêiner, definindo

```
display: flex;
```

ou

```
display: inline-flex;
```

o conteúdo **dentro do contêiner** será alinhado usando o flexbox.

## 32.3. Propriedades do contêiner

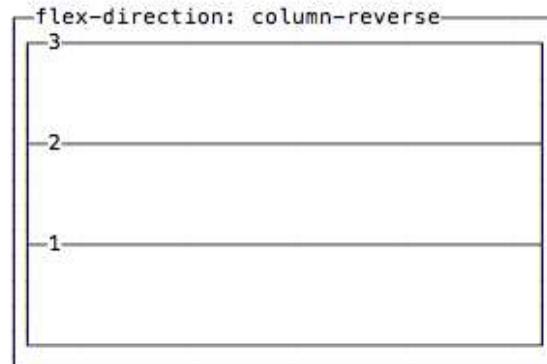
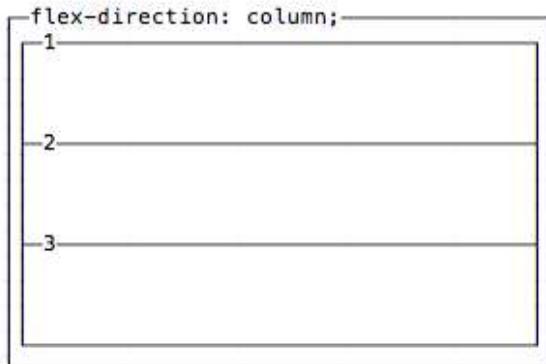
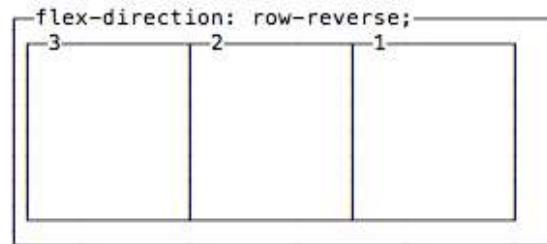
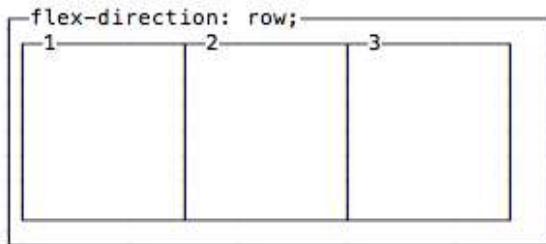
Algumas propriedades do flexbox se aplicam ao contêiner, que define as regras gerais para seus itens. Eles são

- `flex-direction`
- `justify-content`
- `align-items`
- `flex-wrap`
- `flex-flow`

### 32.3.1. Alinhar linhas ou colunas

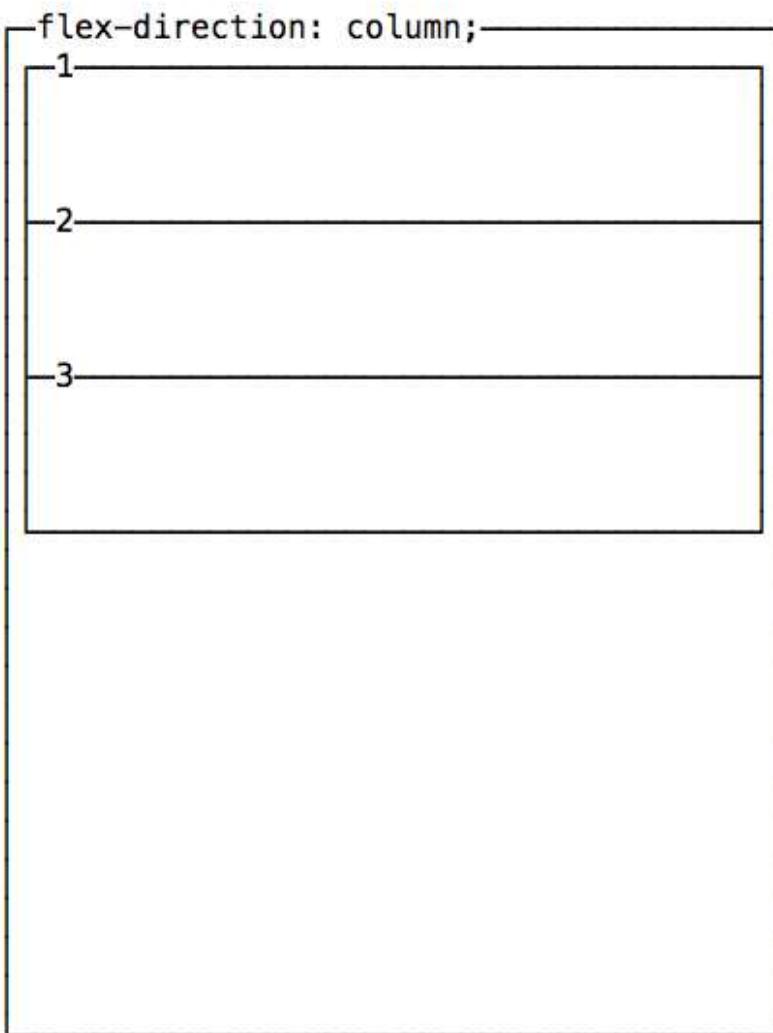
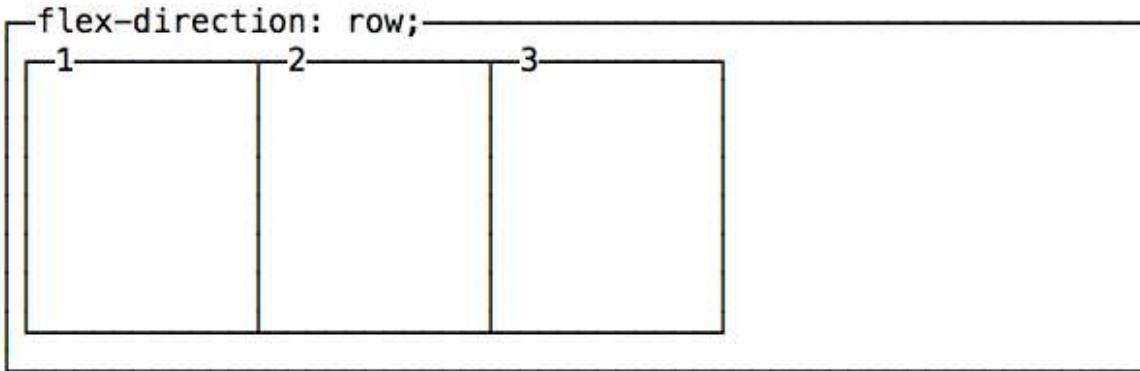
A primeira propriedade que vemos, `flex-direction`, determina se o contêiner deve alinhar seus itens como linhas ou colunas:

- `flex-direction: row` coloca itens como uma **linha**, na direção do texto (da esquerda para a direita para países ocidentais)
- `flex-direction: row-reverse` coloca itens como mas na direção oposta row
- `flex-direction: column` coloca itens em uma **coluna**, ordenando de cima para baixo
- `flex-direction: column-reverse` coloca itens em uma coluna, assim como, mas na direção oposta column
- 



### 32.3.2. Alinhamento vertical e horizontal

Por padrão, os itens começam a partir da esquerda, se for linha, e da parte superior, se for coluna.  
`flex-direction`

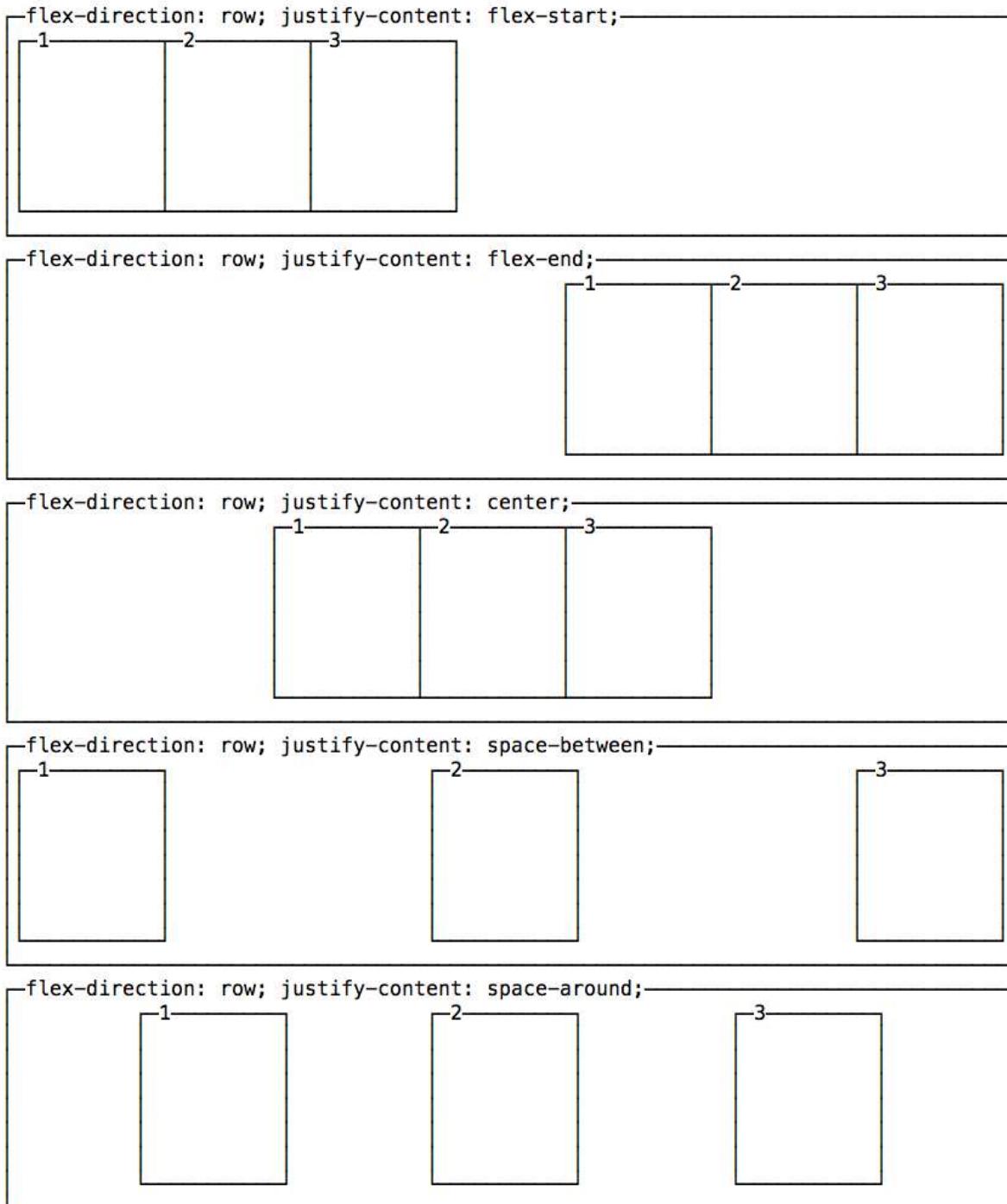


You can change this behavior using to change the horizontal alignment, and to change the vertical alignment.  
`justify-content`  
`align-items`

### 32.3.2.1. Change the horizontal alignment

`justify-content` has 5 possible values:

- **flex-start:** align to the left side of the container.
- **flex-end:** align to the right side of the container.
- **center:** align at the center of the container.
- **space-between:** display with equal spacing between them.
- **space-around:** display with equal spacing around them



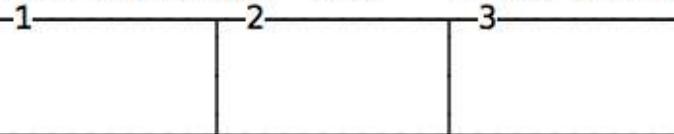
### 32.3.2.2. Change the vertical alignment

**align-items** has 5 possible values:

- **flex-start:** align to the top of the container.

- **flex-end**: align to the bottom of the container.
- **center**: align at the vertical center of the container.
- **baseline**: display at the baseline of the container.
- **stretch**: items are stretched to fit the container.

```
-flex-direction: row;— align-items: flex-start;—
```



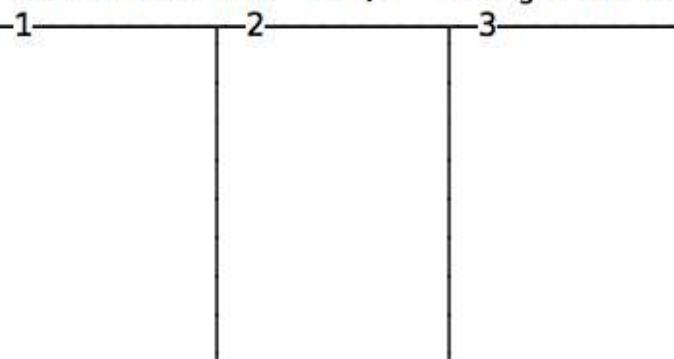
```
-flex-direction: row;— align-items: flex-end;—
```



```
-flex-direction: row;— align-items: baseline;—
```



```
-flex-direction: row;— align-items: stretch;—
```



```
-flex-direction: row;— align-items: center;—
```



### 32.3.2.3. A note on baseline

baseline looks similar to in this example, due to my boxes being too simple. Check out [this Codepen](#) to have a more useful example, which I forked from a Pen originally created by [Martin Michálek](#). As you can see there, items dimensions are aligned.`flex-start`

### 32.3.3. Wrap

By default items in a flexbox container are kept on a single line, shrinking them to fit in the container.

To force the items to spread across multiple lines, use `.flex-wrap`. This will distribute the items according to the order set in `.flex-direction`. Use `.flex-wrap: wrap-reverse` to reverse this order.

A shorthand property called `flex-wrap` allows you to specify `wrap` and `wrap-reverse` in a single line, by adding the value first, followed by value, for example: `.flex-wrap: wrap` or `.flex-wrap: wrap-reverse`.

## 32.4. Properties that apply to each single item

Since now, we've seen the properties you can apply to the container.

Single items can have a certain amount of independence and flexibility, and you can alter their appearance using those properties:

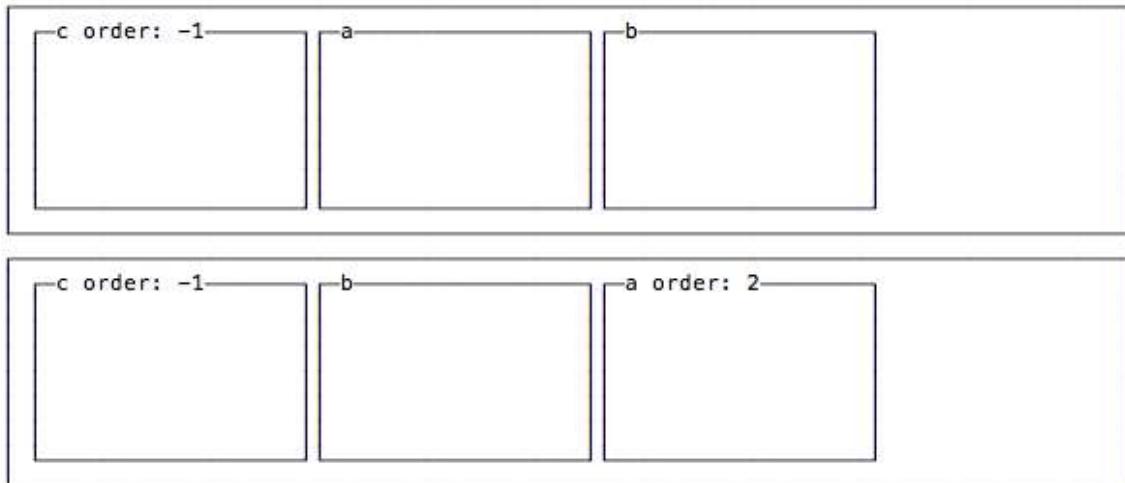
- `order`
- `align-self`
- `flex-grow`
- `flex-shrink`
- `flex-basis`
- `flex`

Vamos vê-los em detalhes.

### 32.4.1. Movendo itens antes / depois de outro usando a ordem

Os itens são encomendados com base em um pedido que lhes é atribuído. Por padrão, cada item tem ordem e a aparência no HTML determina a ordem final.<sup>0</sup>

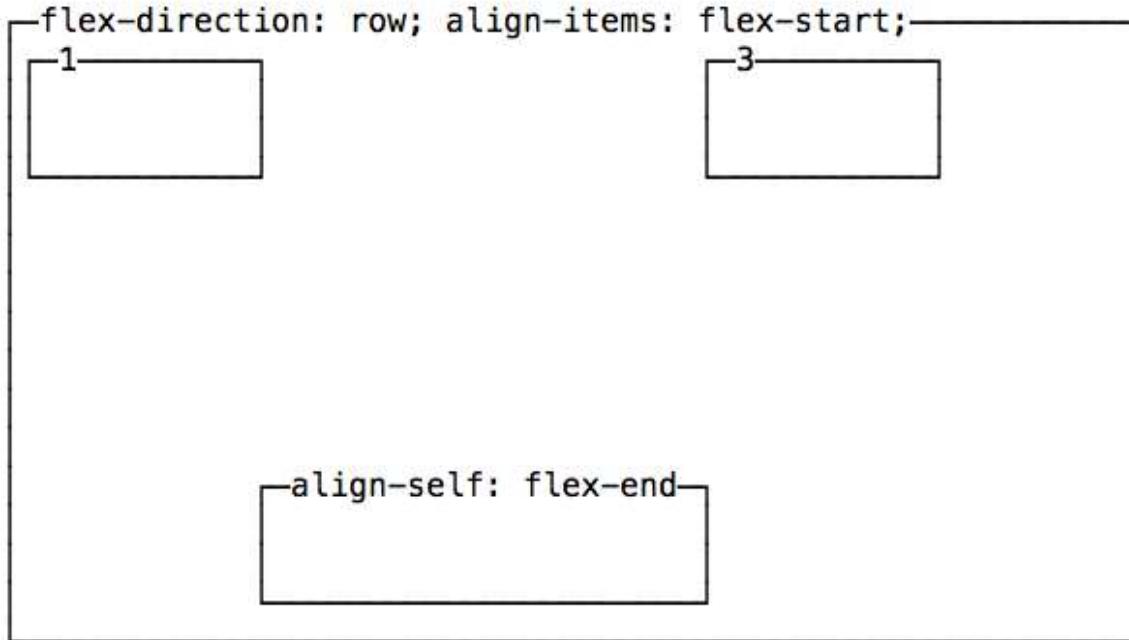
Você pode substituir essa propriedade usando em cada item separado. Essa é uma propriedade definida no item, não no contêiner. Você pode fazer com que um item apareça antes de todos os outros definindo um valor negativo.`order`



### 32.4.2. Alinhamento vertical utilizando align-self

Um item pode optar por **substituir** a configuração do contêiner, usando **align-self**, que tem os mesmos 5 valores possíveis de `align-items`

- `flex-start`: alinhe à parte superior do recipiente.
- `flex-end`: alinhe ao fundo do recipiente.
- `center`: alinhar no centro vertical do recipiente.
- `baseline`: exibir na linha de base do contêiner.
- `stretch`: os itens são esticados para caber no recipiente.



### 32.4.3. Aumentar ou encolher um item, se necessário

#### 32.4.3.1. `flex-grow`

O default para qualquer item é 0.

Se todos os itens forem definidos como 1 e um for definido como 2, o elemento maior ocupará o espaço de dois itens "1".

#### 32.4.3.2. `flex-shrink`

O default para qualquer item é 1.

Se todos os itens forem definidos como 1 e um for definido como 3, o elemento maior encolherá 3x os outros. Quando há menos espaço disponível, será necessário 3x menos espaço.

#### 32.4.3.3. base flexível

Se definido como , ele dimensiona um item de acordo com sua largura ou altura e adiciona espaço extra com base na propriedade `autoflex-grow`

Se definido como 0, ele não adicionará nenhum espaço extra para o item ao calcular o layout.

Se você especificar um valor de número de pixel, ele o usará como o valor de comprimento (largura ou altura depende se for uma linha ou um item de coluna)

### 32.4.3.4. flex

Esta propriedade combina as 3 propriedades acima:

- `flex-grow`
- `flex-shrink`
- `flex-basis`

e fornece uma sintaxe abreviada: `flex: 0 1 auto`

## 33. Tabelas

As tabelas no passado eram muito usadas em excesso no CSS, pois eram uma das únicas maneiras de criar um layout de página sofisticado.

Hoje, com o Grid e o Flexbox, podemos mover as tabelas de volta ao trabalho que elas deveriam fazer: estilizar tabelas.

Vamos começar a partir do HTML. Esta é uma tabela básica:

```
<table>
  <thead>
    <tr>
      <th scope="col">Name</th>
      <th scope="col">Age</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">Flavio</th>
      <td>36</td>
```

```
</tr>
<tr>
  <th scope="row">Roger</th>
  <td>7</td>
</tr>
</tbody>
</table>
```

Por padrão, não é muito atraente. O navegador fornece alguns estilos padrão, e é isso:

Name	Age
Flavio	36
Roger	7
Syd	6

Podemos usar CSS para estilizar todos os elementos da tabela, é claro.

Comecemos pela fronteira. Uma boa fronteira pode percorrer um longo caminho.

Podemos aplicá-lo no elemento, e nos elementos internos também, como e :table th td

```
table,
th,
td {
  border: 1px solid #333;
}
```

Se combinarmos com alguma margem, obteremos um bom resultado:

Name	Age
Flavio	36
Roger	7
Syd	6

Uma coisa comum com tabelas é a capacidade de adicionar uma cor a uma linha e uma cor diferente a outra linha. Isso é possível usando o seletor ou::nth-child(odd):nth-child(even)

```
tbody tr:nth-child(odd) {  
    background-color: #af47ff;  
}
```

Isso nos dá:

Name	Age
Flavio	36
Roger	7
Syd	6

Se você adicionar ao elemento table, todas as bordas serão recolhidas em  
`uma:border-collapse: collapse;`

Name	Age
Flavio	36
Roger	7
Syd	6

## 34. Centralização

Centralizar as coisas em CSS é uma tarefa muito diferente se você precisar centralizar horizontal ou verticalmente.

Neste post explico os cenários mais comuns e como resolvê-los. Se uma nova solução é fornecida pelo Flexbox, eu ignoro as técnicas antigas porque precisamos avançar, e o Flexbox é suportado por navegadores há anos, incluindo o IE10.

### 34.1. Centralizar horizontalmente

#### 34.1.1. Texto

O texto é muito simples de centralizar horizontalmente usando a propriedade definida como :text-align:center

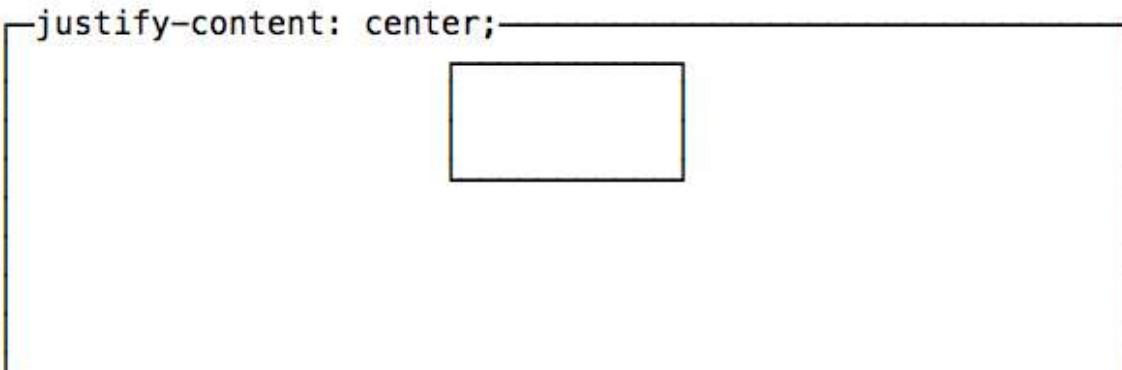
```
p {  
    text-align: center;  
}
```

### 34.1.2. Blocos

A maneira moderna de centralizar qualquer coisa que não seja texto é usar o Flexbox:

```
#mysection {  
    display: flex;  
    justify-content: center;  
}
```

qualquer elemento dentro será centralizado horizontalmente.



Aqui está a abordagem alternativa se você não quiser usar o Flexbox.

Qualquer coisa que não seja texto pode ser centralizada aplicando uma margem automática à esquerda e à direita e definindo a largura do elemento:

```
section {  
    margin: 0 auto;
```

```
    width: 50%;  
}
```

o acima é uma abreviação de:  
`margin: 0 auto;`

```
section {  
    margin-top: 0;  
    margin-bottom: 0;  
    margin-left: auto;  
    margin-right: auto;  
}
```

Lembre-se de definir o item como se ele for um elemento  
embutido.  
`display: block`

## 34.2. Centralizar verticalmente

Tradicionalmente, esta sempre foi uma tarefa difícil. O Flexbox agora nos fornece uma ótima maneira de fazer isso da maneira mais simples possível:

```
#mysection {  
    display: flex;  
    align-items: center;  
}
```

qualquer elemento dentro será centralizado verticalmente.  
`#mysection`

```
align-items: center;
```

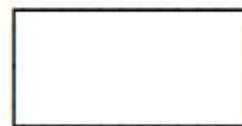


### 34.3. Centralize vertical e horizontalmente

As técnicas de flexbox para centralizar vertical e horizontalmente podem ser combinadas para centralizar completamente um elemento na página.

```
#mysection {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
}
```

```
align-items: center; justify-content: center;
```



O mesmo pode ser feito usando [o CSS Grid](#):

```
body {  
    display: grid;  
    place-items: center;  
    height: 100vh;  
}
```

## 35. Listas

As listas são uma parte muito importante de muitas páginas da web.

CSS pode estilizá-los usando várias propriedades.

`list-style-type` é usado para definir um marcador predefinido a ser usado pela lista:

```
li {  
    list-style-type: square;  
}
```

Temos muitos valores possíveis, que você pode ver aqui <https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-type> com exemplos de sua aparência. Alguns dos mais populares são , , e .disc ccirclesquare none

`list-style-image` é usado para usar um marcador personalizado quando um marcador predefinido não é apropriado:

```
li {  
    list-style-image: url(list-image.png);  
}
```

`list-style-position` permite que você adicione o marcador (o padrão) ou do conteúdo da lista, no fluxo da página, em vez de fora dela outside inside

```
li {  
    list-style-position: inside;
```

}

A propriedade abreviada nos permite especificar todas essas propriedades na mesma linha:`list-style`

```
li {  
    list-style: url(list-image.png) inside;  
}
```

## 36. Consultas de mídia e design responsivo

Nesta seção, vamos primeiro apresentar tipos de mídia e descritores de recursos de mídia e, em seguida, explicaremos as consultas de mídia.

### 36.1. Tipos de mídia

Usados em consultas de mídia e declarações de `@import`, os tipos de mídia nos permitem determinar em qual mídia um arquivo CSS ou uma parte do CSS é carregado.

Temos os seguintes **tipos de mídia**

- `all` significa todos os meios de comunicação
- `print` usado ao imprimir
- `screen` usado quando a página é apresentada em uma tela
- `speech` usado para leitores de tela

`screen` é o padrão.

No passado, tínhamos mais deles, mas a maioria é preterida, pois provaram não ser uma maneira eficaz de determinar as necessidades do dispositivo.

Podemos usá-los em `@import` declarações como esta:

```
@import url(myfile.css) screen;  
@import url(myfile-print.css) print;
```

Podemos carregar um arquivo CSS em vários tipos de mídia separando cada um com uma vírgula:

```
@import url(myfile.css) screen, print;
```

O mesmo funciona para a tag em HTML:`link`

```
<link rel="stylesheet" type="text/css" href="myfile.css" media="screen"  
/>  
<link  
    rel="stylesheet"  
    type="text/css"  
    href="another.css"  
    media="screen, print"  
/>
```

Não estamos limitados a apenas usar tipos de mídia no atributo e na declaração. Há mais `media@import`

## 36.2. Descritores de recursos de mídia

Primeiro, vamos introduzir **descritores de recursos de mídia**. São palavras-chave adicionais que podemos adicionar ao atributo de ou à declaração, para expressar mais condicionais sobre o carregamento do CSS.

Aqui está a lista deles:

- `width`
- `height`
- `device-width`
- `device-height`
- `aspect-ratio`

- device-aspect-ratio
- color
- color-index
- monochrome
- resolution
- orientation
- scan
- grid

Cada um deles tem um min- e max- correspondente, por exemplo:

- min-width, max-width
- min-device-width, max-device-width

e assim por diante.

Alguns deles aceitam um valor de comprimento que pode ser expresso em px ou rem ou qualquer valor de comprimento. É o caso de , , ,  
.pxremwidthheightdevice-widthdevice-height

Por exemplo:

```
@import url(myfile.css) screen and (max-width: 800px);
```

Observe que encapsulamos cada bloco usando descritores de recursos de mídia entre parênteses.

Alguns aceitam um valor fixo. , usado para detectar a orientação do dispositivo, aceita ou .orientationportraitlandscape

Exemplo:

```
<link rel="stylesheet" type="text/css" href="myfile.css" media="screen  
and (orientation: portrait)" />
```

scan, usado para determinar o tipo de tela, aceita (para monitores modernos) ou (para dispositivos CRT mais antigos)progressiveinterlace

Alguns outros querem um inteiro.

Como o que inspeciona o número de bits por componente de cor usado pelo dispositivo. Muito baixo nível, mas você só precisa saber que está lá para o seu uso (como , , ).colorgridcolor-indexmonochrome

aspect-ratio e aceite um valor de proporção que represente a relação largura/altura do visor, que é expressa como uma fração.device-aspect-ratio

Exemplo:

```
@import url(myfile.css) screen and (aspect-ratio: 4/3);
```

resolution representa a densidade de pixels do dispositivo, expressa em um tipo de dados de resolução como .dpi

Exemplo:

```
@import url(myfile.css) screen and (min-resolution: 100dpi);
```

### 36.3. Operadores lógicos

We can combine rules using :and

```
<link rel="stylesheet" type="text/css" href="myfile.css" media="screen  
and (max-width: 800px)" />
```

We can perform an "or" type of logic operation using commas, which combines multiple media queries:

```
@import url(myfile.css) screen, print;
```

We can use to negate a media query:not

```
@import url(myfile.css) not screen;
```

Important: can only be used to negate an entire media query, so it must be placed at the beginning of it (or after a comma)not

## 36.4. Media queries

All those above rules we saw applied to @import or the the HTML tag can be applied inside the CSS, too.[link](#)

You need to wrap them in a structure.

```
@media () {}
```

Example:

```
@media screen and (max-width: 800px) {  
    /* enter some CSS */  
}
```

and this is the foundation for **responsive design**.

Media queries can be quite complex. This example applies the CSS only if it's a screen device, the width is between 600 and 800 pixels, and the orientation is landscape:

```
@media screen and (max-width: 800px) and (min-width: 600px) and  
(orientation: landscape) {  
    /* enter some CSS */  
}
```

## 37. Feature Queries

Feature queries are a recent and relatively unknown ability of CSS, but a well supported one.

We can use it to check if a feature is supported by the browser using the keyword.

```
@supports
```

For example I think this is especially useful, at the time of writing, for checking if a browser supports CSS grid, for example, which can be done using:

```
@supports (display: grid) {  
    /* apply this CSS */  
}
```

We check if the browser supports the value for the property `grid` `display`

We can use for any CSS property, to check any value `@supports`

We can also use the logical operators `and`, `or`, `not` and to build complex feature queries: `and` or `not`

```
@supports (display: grid) and (display: flex) {  
    /* apply this CSS */  
}
```

## 38. Filters

Filters allow us to perform operations on elements.

Things you normally do with Photoshop or other photo editing software, like changing the opacity or the brightness, and more.

You use the `filter` property. Here's an example of it applied on an image, but this property can be used on *any* element: `filter`

```
img {  
    filter: <something>;  
}
```

You can use various values here:

- `blur()`
- `brightness()`

- contrast()
- drop-shadow()
- grayscale()
- hue-rotate()
- invert()
- opacity()
- sepia()
- saturate()
- url()

Notice the parentheses after each option, because they all require a parameter.

For example:

```
img {  
  filter: opacity(0.5);  
}
```

means the image will be 50% transparent, because takes one value from 0 to 1, or a percentage.`opacity()`

You can also apply multiple filters at once:

```
img {  
  filter: opacity(0.5) blur(2px);  
}
```

Let's now talk about each filter in details.

## 38.0.1. blur()

Desfoca o conteúdo de um elemento. Você passa um valor, expresso em ou que será usado para determinar o raio de desfoque.`px`

Exemplo:

```
img {
  filter: blur(4px);
}
```

## 38.0.2. opacity()

`opacity()` usa um valor de 0 a 1, ou uma porcentagem, e determina a transparência da imagem com base nele.

0, ou , significa totalmente transparente. , ou , ou superior, significa totalmente visível.  
0% 100%

Exemplo:

```
img {
  filter: opacity(0.5);
}
```

CSS também tem uma propriedade. no entanto, pode ser acelerado por hardware, dependendo da implementação, por isso este deve ser o método preferido.`opacityfilter`

## 38.0.3. drop-shadow()

`drop-shadow()` mostra uma sombra atrás do elemento, que segue o canal alfa. Isso significa que, se você tiver uma imagem transparente, obterá uma sombra aplicada à forma da imagem, não à caixa de imagem. Se a imagem não tiver um canal alfa, a sombra será aplicada a toda a caixa de imagem.

Aceita um mínimo de 2 parâmetros, até 5:

- *offset-x* define o deslocamento horizontal. Pode ser negativo.
- *offset-y* define o deslocamento vertical. Pode ser negativo.
- *raio de desfoque*, opcional, define o *raio de desfoque* para a sombra. O padrão é 0, sem desfoque.
- *o raio de propagação*, opcional, define o *raio de propagação*. Expresso em , ou pxremem

- *color*, opcional, define a cor da sombra.

Você pode definir a cor sem definir o raio de propagação ou o raio de desfoque. CSS entende que o valor é uma cor e não um valor de comprimento.

Exemplo:

```
img {  
    filter: drop-shadow(10px 10px 5px orange);  
}
```

```
img {  
    filter: drop-shadow(10px 10px orange);  
}
```

```
img {  
    filter: drop-shadow(10px 10px 5px 5px #333);  
}
```

## 38.0.4. grayscale()

Faça com que o elemento tenha uma cor cinza.

Você passa um valor de 0 para 1, ou de 0% para 100%, onde 1 e 100% significam completamente cinza, e 0 ou 0% significam que a imagem não é tocada e as cores originais permanecem.

Exemplo:

```
img {  
    filter: grayscale(50%);  
}
```

## 38.0.5. sepia()

Faça com que o elemento tenha uma cor sépia.

Você passa um valor de 0 para 1, ou de 0% para 100%, onde 1 e 100% significam completamente sépia, e 0 ou 0% significam que a imagem não é tocada e as cores originais permanecem.

Exemplo:

```
img {  
    filter: sepia(50%);  
}
```

### 38.0.6. invert()

Inverter as cores de um elemento. Inverter uma cor significa procurar o oposto de uma cor na roda de cores HSL. Basta pesquisar "roda de cores" no Google se você não tem ideia do que isso significa. Por exemplo, o oposto do amarelo é azul, o oposto do vermelho é ciano. Cada cor tem um oposto.

Você passa um número, de 0 a 1 ou de 0% a 100%, que determina a quantidade de inversão. 1 ou 100% significa inversão total, 0 ou 0% significa nenhuma inversão.

0,5 ou 50% sempre renderizará uma cor cinza de 50%, porque você sempre acaba no meio da roda.

Exemplo:

```
img {  
    filter: invert(50%);  
}
```

### 38.0.7. hue-rotate()

A roda de cores HSL é representada em graus. Usando você pode girar a cor usando uma rotação positiva ou negativa.`hue-rotate()`

A função aceita um valor.deg

Exemplo:

```
img {  
  filter: hue-rotate(90deg);  
}
```

## 38.0.8. brightness()

Altera o brilho de um elemento.

0 ou 0% dá um elemento preto total. 1 ou 100% dá uma imagem inalterada

Valores superiores a 1 ou 100% tornam a imagem mais brilhante para atingir um elemento branco total.

Exemplo:

```
img {  
  filter: brightness(50%);  
}
```

## 38.0.9. contrast()

Altera o contraste de um elemento.

0 ou 0% dá um elemento cinza total. 1 ou 100% dá uma imagem inalterada

Valores superiores a 1 ou 100% dão mais contraste.

Exemplo:

```
img {  
  filter: contrast(150%);  
}
```

## 38.0.10. saturate()

Altera a saturação de um elemento.

0 ou 0% dá um elemento de escala de cinza total (com menos saturação). 1 ou 100% dá uma imagem inalterada

Valores superiores a 1 ou 100% dão mais saturação.

Exemplo:

```
img {  
  filter: saturate();  
}
```

## 38.0.11. url()

Este filtro permite aplicar um filtro definido em um arquivo SVG. Aponte para o local do arquivo SVG.

Exemplo:

```
img {  
  filter: url(filter.svg);  
}
```

Os filtros SVG estão fora do escopo deste livro, mas você pode ler mais neste post da Smashing Magazine:

<https://www.smashingmagazine.com/2015/05/why-the-svg-filter-is-awesome/>

# 39. Transforma

As transformações permitem traduzir, girar, dimensionar e distorcer elementos no espaço 2D ou 3D. Eles são um recurso CSS muito legal, especialmente quando combinados com animações.

## 39.1. Transformações 2D

A propriedade aceita essas funções:`transform`

- `translate()` para mover elementos
- `rotate()` para girar elementos
- `scale()` para dimensionar elementos em tamanho
- `skew()` para torcer ou inclinar um elemento
- `matrix()` uma maneira de executar qualquer uma das operações acima usando uma matriz de 6 elementos, uma sintaxe menos amigável, mas menos detalhada

Também temos funções específicas do eixo:

- `translateX()` para mover elementos no eixo X
- `translateY()` para mover elementos no eixo Y
- `scaleX()` para dimensionar elementos em tamanho no eixo X
- `scaleY()` para dimensionar elementos em tamanho no eixo Y
- `skewX()` para torcer ou inclinar um elemento no eixo X
- `skewY()` para torcer ou inclinar um elemento no eixo Y

Aqui está um exemplo de uma transformação que altera a largura do elemento em 2 (duplicando-o) e a altura em 0,5 (reduzindo-o à metade):`.box`

```
.box {  
    transform: scale(2, 0.5);  
}
```

A origem da transformação nos permite definir a origem (as coordenadas) para a transformação, deixando-nos mudar o centro de rotação. $(0, 0)$

## 39.2. Combinando múltiplas transformações

Você pode combinar várias transformações separando cada função com um espaço.

Por exemplo:

```
transform: rotateY(20deg) scaleX(3) translateY(100px);
```

## 39.3. Transformações 3D

Podemos dar um passo adiante e mover nossos elementos em um espaço 3D em vez de em um espaço 2D. Com o 3D, estamos adicionando outro eixo, Z, que adiciona profundidade aos visuais.

Usando a propriedade, você pode especificar a que distância o objeto 3D está do visualizador.`perspective`

Exemplo:

```
.3delement {  
    perspective: 100px;  
}
```

`perspective-origin` determina a aparência da posição do espectador, como estamos olhando para ele nos eixos X e Y.

Agora podemos usar funções adicionais que controlam o eixo Z, que se soma às outras transformações dos eixos X e Y:

- `translateZ()`
- `rotateZ()`
- `scaleZ()`

e os atalhos correspondentes, e como abreviaturas para usar o , e funções e assim por diante.`translate3d()``rotate3d()``scale3d()``translateX()``translateY()``translateZ()`

As transformações 3D são um pouco avançadas demais para este manual, mas um ótimo tópico para explorar por conta própria.

## 40. Transições

As transições CSS são a maneira mais simples de criar uma animação em CSS.

Em uma transição, você altera o valor de uma propriedade e diz ao CSS para alterá-la lentamente de acordo com alguns parâmetros, em direção a um estado final.

As transições CSS são definidas por estas propriedades:

Propriedade	Descrição
transition-property	a propriedade CSS que deve fazer a transição
transition-duration	a duração da transição
transition-timing-function	a função de temporização usada pela animação (valores comuns: linear, ease). Padrão: facilidade
transition-delay	número opcional de segundos a aguardar antes de iniciar a animação

A propriedade é uma taquigrafia útil: `transition`

```
.container {  
    transition: property duration timing-function delay;  
}
```

## 40.1. Exemplo de transição CSS

Este código implementa uma transição CSS:

```
.one,  
.three {  
    background: rgba(142, 92, 205, 0.75);  
    transition: background 1s ease-in;  
}  
  
.two,  
.four {
```

```
background: rgba(236, 252, 100, 0.75);  
}  
  
.circle:hover {  
    background: rgba(142, 92, 205, 0.25); /* lighter */  
}
```

Veja o exemplo em Glitch <https://flavio-css-transitions-example.glitch.me>

Ao passar o mouse sobre os elementos e, os círculos roxos, há uma animação de transição que facilita a mudança de fundo, enquanto os círculos amarelos não, porque eles não têm a propriedade definida..one.threetransition

## 40.2. Valores da função de temporização de transição

transition-timing-function permite especificar a curva de aceleração da transição.

Existem alguns valores simples que você pode usar:

- linear
- ease
- ease-in
- ease-out
- ease-in-out

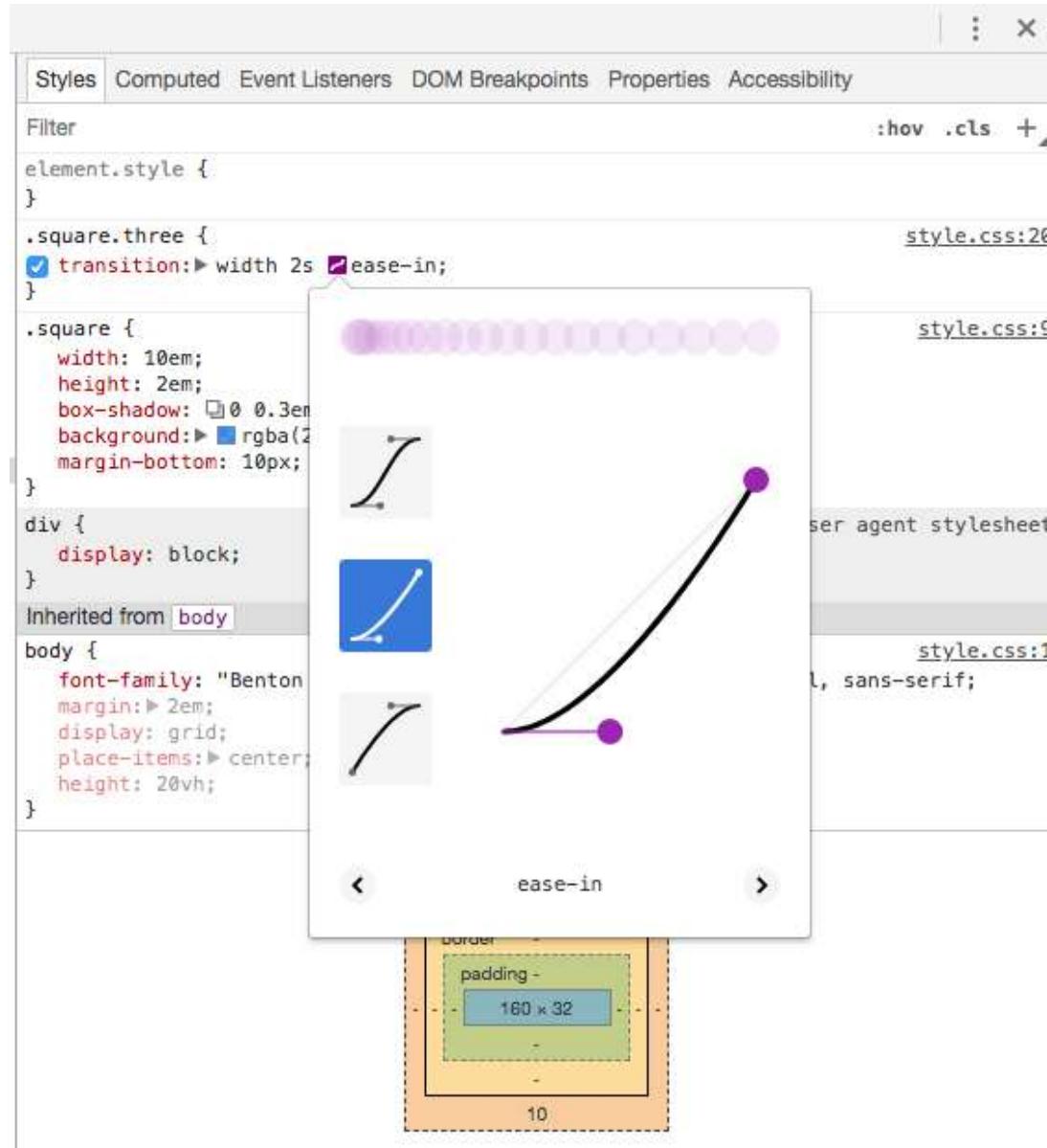
Esta falha mostra como eles funcionam na prática.

Você pode criar uma função de temporização completamente personalizada usando curvas de bezier cúbicas. Isso é bastante avançado, mas basicamente qualquer uma dessas funções acima é construída usando curvas de bezier. Temos nomes úteis, pois são comuns.

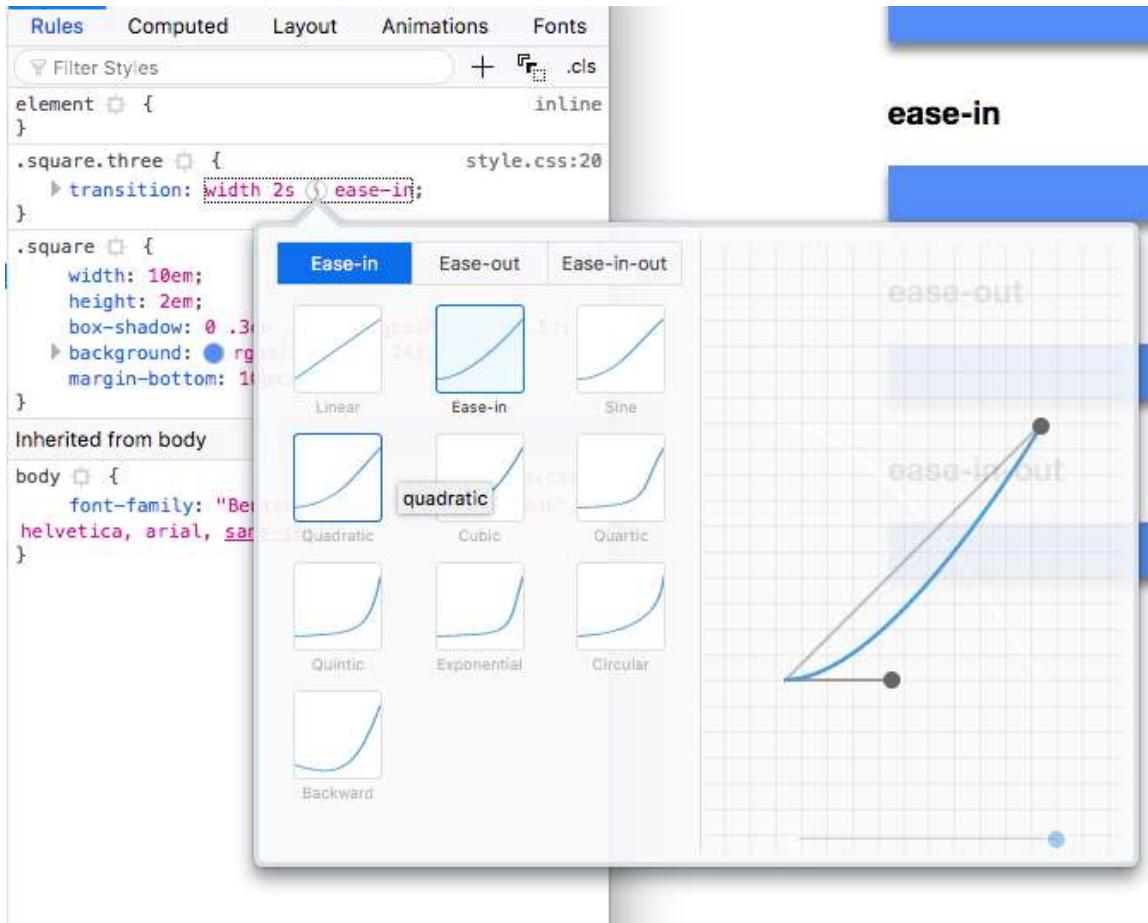
## 40.3.CSS Transições no DevTools do navegador

O Browser DevTools oferece uma ótima maneira de visualizar transições.

Este é o Chrome:



Este é o Firefox:



A partir desses painéis, você pode editar ao vivo a transição e experimentar na página diretamente sem recarregar seu código.

## 40.4. Quais propriedades você pode animar usando animações CSS

Muito! Eles são os mesmos que você pode animar usando CSS Transitions, também.

Aqui está a lista completa:

- background
- background-color
- background-position
- background-size
- border
- border-color
- border-width
- border-bottom
- border-bottom-color

- border-bottom-left-radius
- border-bottom-right-radius
- border-bottom-width
- border-left
- border-left-color
- border-left-width
- border-radius
- border-right
- border-right-color
- border-right-width
- border-spacing
- border-top
- border-top-color
- border-top-left-radius
- border-top-right-radius
- border-top-width
- bottom
- box-shadow
- caret-color
- clip
- color
- column-count
- column-gap
- column-rule
- column-rule-color
- column-rule-width
- column-width
- columns
- content
- filter
- flex
- flex-basis
- flex-grow
- flex-shrink
- font
- font-size

- font-size-adjust
- font-stretch
- font-weight
- grid-area
- grid-auto-columns
- grid-auto-flow
- grid-auto-rows
- grid-column-end
- grid-column-gap
- grid-column-start
- grid-column
- grid-gap
- grid-row-end
- grid-row-gap
- grid-row-start
- grid-row
- grid-template-areas
- grid-template-columns
- grid-template-rows
- grid-template
- grid
- height
- left
- letter-spacing
- line-height
- margin
- margin-bottom
- margin-left
- margin-right
- margin-top
- max-height
- max-width
- min-height
- min-width
- opacity
- order

- outline
- outline-color
- outline-offset
- outline-width
- padding
- padding-bottom
- padding-left
- padding-right
- padding-top
- perspective
- perspective-origin
- quotes
- right
- tab-size
- text-decoration
- text-decoration-color
- text-indent
- text-shadow
- top
- transform.
- vertical-align
- visibility
- width
- word-spacing
- z-index

## 41. Animações

As animações CSS são uma ótima maneira de criar animações visuais, não limitadas a um único movimento como as transições CSS, mas muito mais articuladas.

Uma animação é aplicada a um elemento usando a propriedade `animation`

```
.container {  
    animation: spin 10s linear infinite;
```

}

spin é o nome da animação, que precisamos definir separadamente. Também dizemos ao CSS para fazer a animação durar 10 segundos, executá-la de forma linear (sem aceleração ou qualquer diferença em sua velocidade) e repeti-la infinitamente.

Você deve **definir como sua animação funciona** usando **quadros-chave**. Exemplo de uma animação que gira um item:

```
@keyframes spin {  
 0% {  
   transform: rotateZ(0);  
 }  
 100% {  
   transform: rotateZ(360deg);  
 }  
}
```

Dentro da definição, você pode ter quantos waypoints intermediários quiser.`@keyframes`

Neste caso, instruímos o CSS a fazer com que a propriedade transform gire o eixo Z de 0 a 360 graus, completando o loop completo.

Você pode usar qualquer transformação CSS aqui.

Observe como isso não dita nada sobre o intervalo temporal que a animação deve tomar. Isso é definido quando você o usa via `.animation`

## 41.1. Um exemplo de animações CSS

Quero desenhar quatro círculos, todos com um ponto de partida em comum, todos a 90 graus distantes um do outro.

```
<div class="container">  
  <div class="circle one"></div>  
  <div class="circle two"></div>
```

```
<div class="circle three"></div>
<div class="circle four"></div>
</div>

body {
    display: grid;
    place-items: center;
    height: 100vh;
}

.circle {
    border-radius: 50%;
    left: calc(50% - 6.25em);
    top: calc(50% - 12.5em);
    transform-origin: 50% 12.5em;
    width: 12.5em;
    height: 12.5em;
    position: absolute;
    box-shadow: 0 1em 2em rgba(0, 0, 0, 0.5);
}

.one,
.three {
    background: rgba(142, 92, 205, 0.75);
}

.two,
.four {
    background: rgba(236, 252, 100, 0.75);
}

.one {
    transform: rotateZ(0);
}

.two {
    transform: rotateZ(90deg);
}
```

```
.three {  
    transform: rotateZ(180deg);  
}  
  
.four {  
    transform: rotateZ(-90deg);  
}
```

Você pode vê-los neste Glitch: <https://flavio-css-circles.glitch.me>

Vamos fazer essa estrutura (todos os círculos juntos) girar. Para fazer isso, aplicamos uma animação no contêiner e definimos essa animação como uma rotação de 360 graus:

```
@keyframes spin {  
    0% {  
        transform: rotateZ(0);  
    }  
    100% {  
        transform: rotateZ(360deg);  
    }  
}  
  
.container {  
    animation: spin 10s linear infinite;  
}
```

Veja no <https://flavio-css-animations-tutorial.glitch.me>

Você pode adicionar mais quadros-chave para ter animações mais engraçadas:

```
@keyframes spin {  
    0% {  
        transform: rotateZ(0);  
    }  
    25% {
```

```

    transform: rotateZ(30deg);
}
50% {
    transform: rotateZ(270deg);
}
75% {
    transform: rotateZ(180deg);
}
100% {
    transform: rotateZ(360deg);
}
}

```

Veja o exemplo no <https://flavio-css-animations-four-steps.glitch.me>

## 41.2. As propriedades de animação CSS

As animações CSS oferecem muitos parâmetros diferentes que você pode ajustar:

Propriedade	Descrição
animation-name	o nome da animação, ele faz referência a uma animação criada usando @keyframes
animation-duration	quanto tempo a animação deve durar, em segundos
animation-timing-function	a função de temporização usada pela animação (valores comuns: , ). Inadimplênci: lineareasee ease
animation-delay	número opcional de segundos a aguardar antes de iniciar a animação
animation-iteration-count	quantas vezes a animação deve ser executada. Espera um número, ou . Padrão: 1infinite
animation-direction	a direção da animação. Pode ser , , ou . Nos últimos 2, alterna para frente e depois para trásnormalreversealternatealternate-reverse

Propriedade	Descrição
animation-fill-mode	define como estilizar o elemento quando a animação termina, depois que ela termina seu número de contagem de iteração. ou volte para os primeiros estilos de quadro-chave. e use o estilo definido no último quadro-chave nonebackwardsforwardsboth
animation-play-state	se definido como , ele pausa a animação. O padrão é pausedrunning

A propriedade é uma abreviação para todas essas propriedades, nesta ordem:animation

```
.container {
  animation: name duration timing-function delay iteration-count
  direction
  fill-mode play-state;
}
```

Este é o exemplo que usamos acima:

```
.container {
  animation: spin 10s linear infinite;
}
```

## 41.3. JavaScript events for CSS Animations

Using JavaScript you can listen for the following events:

- animationstart
- animationend
- animationiteration

Be careful with , because if the animation starts on page load, your JavaScript code is always executed after the CSS has been processed, so the animation is already started and you cannot intercept the event.animationstart

```
const container = document.querySelector('.container')
```

```
container.addEventListener(  
  'animationstart',  
  (e) => {  
    //do something  
  },  
  false  
)
```

```
container.addEventListener(  
  'animationend',  
  (e) => {  
    //do something  
  },  
  false  
)
```

```
container.addEventListener(  
  'animationiteration',  
  (e) => {  
    //do something  
  },  
  false  
)
```

## 41.4. Which Properties You Can Animate using CSS Animations

A lot! They are the same you can animate using CSS Transitions, too.

Here's the full list:

- background
- background-color
- background-position
- background-size

- border
- border-color
- border-width
- border-bottom
- border-bottom-color
- border-bottom-left-radius
- border-bottom-right-radius
- border-bottom-width
- border-left
- border-left-color
- border-left-width
- border-radius
- border-right
- border-right-color
- border-right-width
- border-spacing
- border-top
- border-top-color
- border-top-left-radius
- border-top-right-radius
- border-top-width
- bottom
- box-shadow
- caret-color
- clip
- color
- column-count
- column-gap
- column-rule
- column-rule-color
- column-rule-width
- column-width
- columns
- content
- filter
- flex

- flex-basis
- flex-grow
- flex-shrink
- font
- font-size
- font-size-adjust
- font-stretch
- font-weight
- grid-area
- grid-auto-columns
- grid-auto-flow
- grid-auto-rows
- grid-column-end
- grid-column-gap
- grid-column-start
- grid-column
- grid-gap
- grid-row-end
- grid-row-gap
- grid-row-start
- grid-row
- grid-template-areas
- grid-template-columns
- grid-template-rows
- grid-template
- grid
- height
- left
- letter-spacing
- line-height
- margin
- margin-bottom
- margin-left
- margin-right
- margin-top
- max-height

- max-width
- min-height
- min-width
- opacity
- order
- outline
- outline-color
- outline-offset
- outline-width
- padding
- padding-bottom
- padding-left
- padding-right
- padding-top
- perspective
- perspective-origin
- quotes
- right
- tab-size
- text-decoration
- text-decoration-color
- text-indent
- text-shadow
- top
- transform.
- vertical-align
- visibility
- width
- word-spacing
- z-index

## 42. Normalizando o CSS

A folha de estilo padrão do navegador é o conjunto de regras que o navegador tem para aplicar algum estilo mínimo aos elementos.

Na maioria das vezes, esses estilos são muito úteis.

Como cada navegador tem seu próprio conjunto, é comum encontrar um terreno comum.

Em vez de remover todos os padrões, como uma das abordagens de **redefinição de CSS**, o processo de normalização remove as inconsistências do navegador, mantendo um conjunto básico de regras nas quais você pode confiar.

Normalize.css <http://necolas.github.io/normalize.css> é a solução mais comumente usada para esse problema.

Você deve carregar o arquivo CSS de normalização antes de qualquer outro CSS.

## 43. Tratamento de erros

O CSS é resiliente. Quando ele encontra um erro, ele não age como o JavaScript, que empacota todas as suas coisas e desaparece completamente, encerrando toda a execução do script depois que o erro é encontrado.

CSS se esforça muito para fazer o que você quer.

Se uma linha tiver um erro, ele a ignorará e pulará para a próxima linha sem nenhum erro.

Se você esquecer o ponto-e-vírgula em uma linha:

```
p {  
    font-size: 20px  
    color: black;  
    border: 1px solid black;  
}
```

a linha com o erro E a próxima **não** será aplicada, mas a terceira regra será aplicada com êxito na página. Basicamente, ele verifica tudo até encontrar

um ponto-e-vírgula, mas quando o alcança, a regra agora é , o que é inválido, então ele a ignora.`font-size: 20px; color: black;`

Às vezes é complicado perceber que há um erro em algum lugar e onde esse erro está, porque o navegador não nos diz.

É por isso que existem ferramentas como [o CSS Lint](#).

## 44. Prefixos do fornecedor

Os prefixos de fornecedor são uma maneira que os navegadores usam para nos dar aos desenvolvedores de CSS acesso a recursos mais recentes ainda não considerados estáveis.

Antes de continuar, tenha em mente que essa abordagem está diminuindo em popularidade, em favor do uso de **sinalizadores experimentais**, que devem ser ativados explicitamente no navegador do usuário.

Por que? Como os desenvolvedores, em vez de considerar os prefixos de fornecedor como uma maneira de visualizar recursos, eles os enviaram em produção - algo considerado prejudicial pelo CSS Working Group.

Principalmente porque uma vez que você adiciona um sinalizador e os desenvolvedores começam a usá-lo em produção, os navegadores estão em uma posição ruim se perceberem que algo deve mudar. Com bandeiras, você não pode enviar um recurso, a menos que você possa empurrar todos os seus visitantes para ativar essa bandeira em seu navegador (apenas brincando, não tente).

Dito isso, vamos ver quais são os prefixos do fornecedor.

Lembro-me especificamente deles por trabalharem com CSS Transitions no passado. Em vez de apenas usar a propriedade, você tinha que fazer o seguinte:`transition`

```
.myClass {  
    -webkit-transition: all 1s linear;  
    -moz-transition: all 1s linear;  
    -ms-transition: all 1s linear;
```

```
-o-transition: all 1s linear;  
transition: all 1s linear;  
}
```

Now you just use

```
.myClass {  
  transition: all 1s linear;  
}
```

since the property is now well supported by all modern browsers.

The prefixes used are:

- -webkit- (Chrome, Safari, iOS Safari / iOS WebView, Android)
- -moz- (Safari)
- -ms- (Edge, Internet Explorer)
  
- -o- (Opera, Opera Mini)

Como o Opera é baseado no Chromium e o Edge em breve também será, e provavelmente sairá de moda em breve. Mas, como dissemos, os prefixos de fornecedores como um todo também estão saindo de moda.-o--ms-

Escrever prefixos é difícil, principalmente por causa da incerteza. Você realmente precisa de um prefixo para uma propriedade? Vários recursos online também estão desatualizados, o que torna ainda mais difícil fazer o certo. Projetos como o [Autoprefixer](#) podem automatizar o processo em sua totalidade sem que precisemos descobrir se um prefixo é mais necessário, ou se o recurso agora está estável e o prefixo deve ser descartado. Ele usa dados do caniuse.com, um site de referência muito bom para todas as coisas relacionadas ao suporte do navegador.

Se você usa React ou Vue, projetos como o Vue CLI, duas maneiras comuns de começar a construir um aplicativo, use fora da caixa, para que você nem precise se preocupar com isso.create-react-app autoprefixer

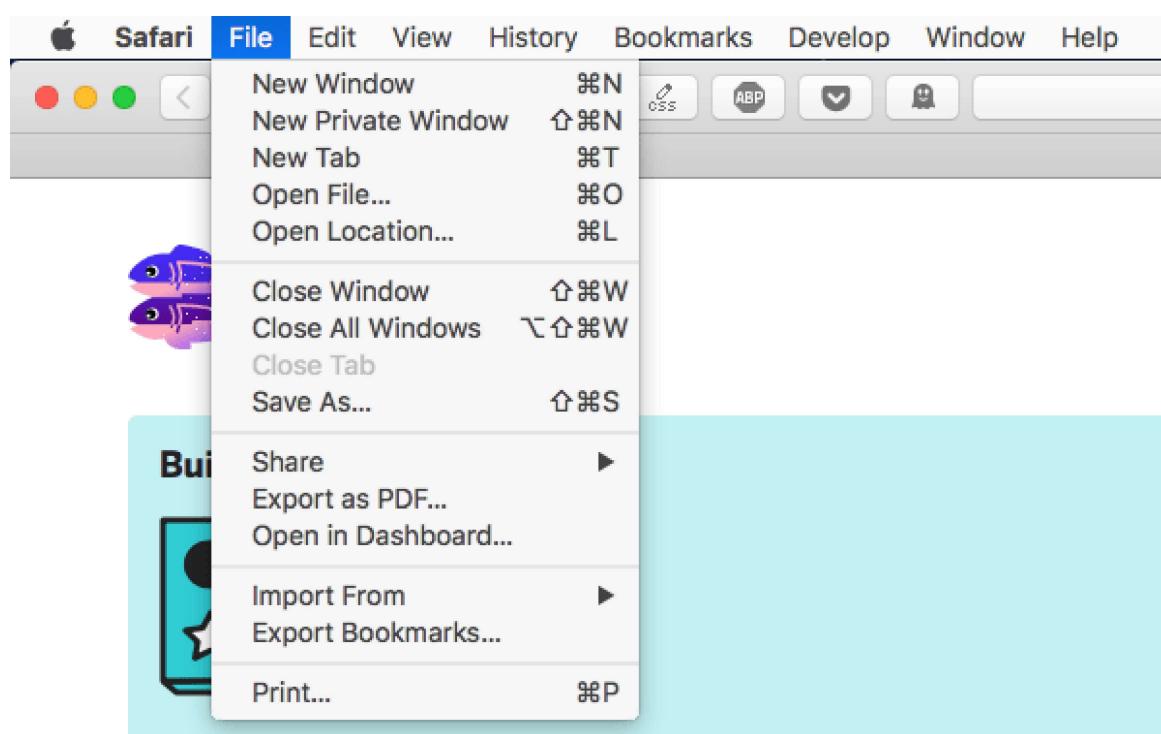
## 45.CSS para impressão

Mesmo que cada vez mais olhemos para nossas telas, a impressão ainda é uma coisa.

Mesmo com postagens de blog. Lembro-me de uma vez em 2009 que conheci uma pessoa que me disse que fazia seu assistente pessoal imprimir cada postagem de blog que eu publicava (sim, eu olhava fixamente por um pouco). Definitivamente inesperado.

Meu principal caso de uso para olhar para a impressão geralmente é imprimir em um PDF. Posso criar algo dentro do navegador e quero disponibilizá-lo como PDF.

Os navegadores tornam isso muito fácil, com o Chrome usando como padrão "Salvar" ao tentar imprimir um documento e uma impressora não está disponível, e o Safari tem um botão dedicado na barra de menus:



## 45.1. Imprimir CSS

Algumas coisas comuns que você pode querer fazer ao imprimir é ocultar algumas partes do documento, talvez o rodapé, algo no cabeçalho, na barra lateral.

Talvez você queira usar uma fonte diferente para impressão, o que é totalmente legítimo.

Se você tiver um CSS grande para impressão, é melhor usar um arquivo separado para ele. Os navegadores só o baixarão ao imprimir:

```
<link rel="stylesheet" src="print.css" type="text/css" media="print" />
```

## 45.2.CSS @media impressão

Uma alternativa à abordagem anterior são as consultas de mídia. Qualquer coisa que você adicionar dentro deste bloco:

```
@media print {  
  /* ... */  
}
```

será aplicado apenas a documentos impressos.

## 45.3. Ligações

HTML é ótimo por causa de links. É chamado de HyperText por um bom motivo. Ao imprimir, podemos perder muita informação, dependendo do conteúdo.

O CSS oferece uma ótima maneira de resolver esse problema editando o conteúdo, anexando o link após o texto da tag, usando:<a>

```
@media print {  
  a[href*='//']:after  
  {  
    content: ' (' attr(href) ')';  
    color: $primary;  
  }  
}
```

I target to only do this for external links. I might have internal links for navigation and internal indexing purposes, which would be useless in most of my use cases. If you also want internal links to be printed, just do:  
a[href\*='//']

```
@media print {
  a:after {
    content: ' (' attr(href) ')';
    color: $primary;
  }
}
```

## 45.4. Page margins

You can add margins to every single page. or is a good unit for paper printing.cm in

```
@page {
  margin-top: 2cm;
  margin-bottom: 2cm;
  margin-left: 2cm;
  margin-right: 2cm;
}
```

@page can also be used to only target the first page, using , or only the left and right pages using and .@page :first@page :left@page: right

## 45.5. Page breaks

You might want to add a page break after some elements, or before them. Use and :page-break-afterpage-break-before

```
.book-date {
  page-break-after: always;
}

.post-content {
  page-break-before: always;
}
```

Those properties accept a wide variety of values.

## 45.6. Avoid breaking images in the middle

I experienced this with Firefox: images by default are cut in the middle, and continue on the next page. It might also happen to text.

Use

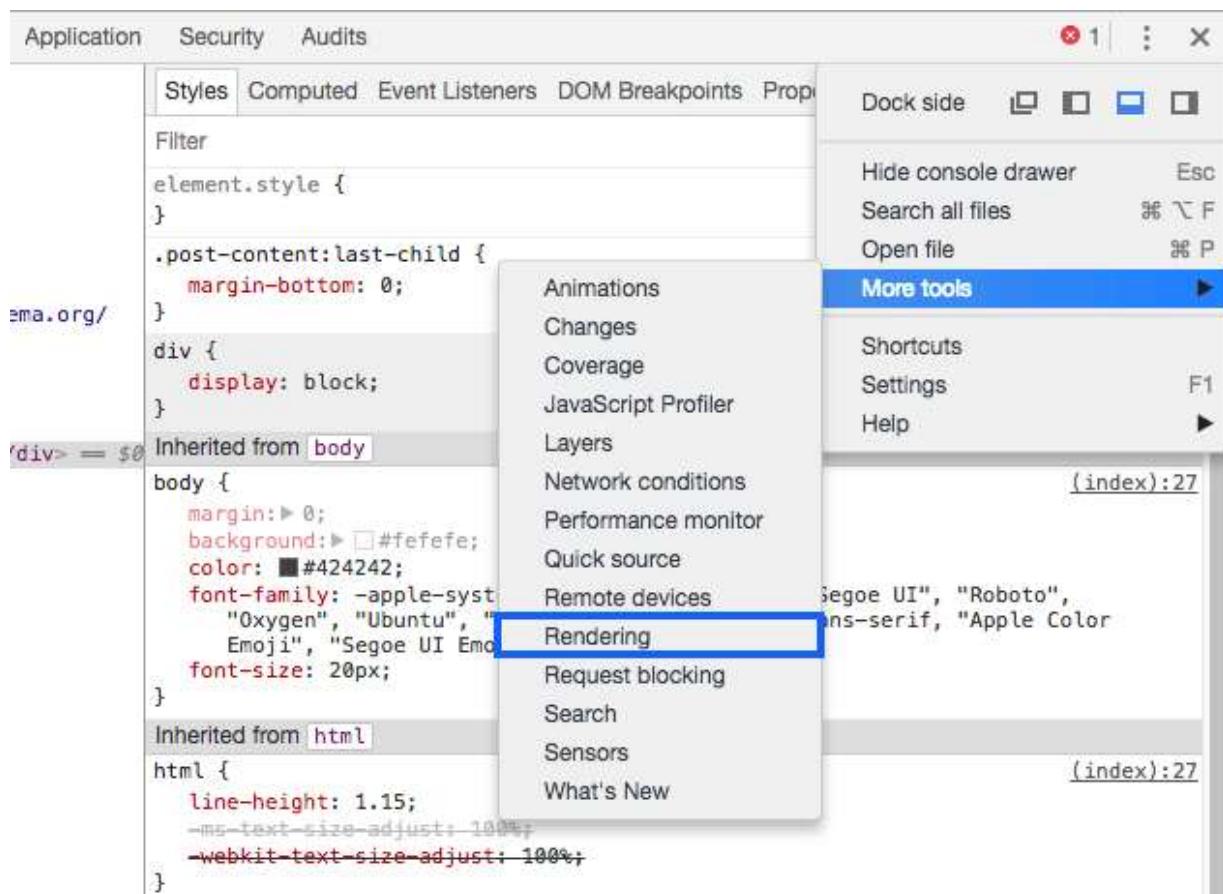
```
p {  
    page-break-inside: avoid;  
}
```

and wrap your images in a tag. Targeting directly didn't work in my tests.pimg

This applies to other content as well, not just images. If you notice something is cut when you don't want, use this property.

## 45.7. Debug the printing presentation

The Chrome DevTools offer ways to emulate the print layout:



Quando o painel for aberto, altere a emulação de renderização para:print

The screenshot shows the Chrome DevTools interface with the 'Rendering' tab selected. Below it is a list of emulation options with checkboxes:

- Paint flashing  
Highlights areas of the page (green) that need to be repainted
- Layer borders  
Shows layer borders (orange/olive) and tiles (cyan)
- FPS meter  
Plots frames per second, frame rate distribution, and GPU memory
- Scrolling performance issues  
Highlights elements (teal) that can slow down scrolling, including touch & wheel event handlers

Below these is a section titled 'Emulate CSS media' with the subtext 'Forces media type for testing print and screen styles'. Underneath is a dropdown menu with three items:

- No emulation (selected, indicated by a checkmark)
- print (highlighted with a blue background)
- screen

🐦 → Você pode me seguir no [Twitter](#)

🅱️ → Todos os anos organizo um [BOOTCAMP](#) de codificação para lhe ensinar JS, Tailwind, React e Next.js (próxima edição Q1 2024)

⌚ → Quer aprender JavaScript AGORA MESMO com um currículo bem organizado? [O CURSO JS](#) é o lugar

🌴 → Vá para [o SOLO LAB](#) se você estiver em iniciar e administrar um negócio na Internet como uma pessoa solo (NOVO CURSO EM BREVE, junte-se à lista de espera para ficar por dentro)

📚 → Leia meus outros ebooks O Manual C O Manual da Linha de Comando O Manual CSS O Manual do Expresso O Manual do Go O Manual HTML O Manual do JS O Próximo.js Manual O Nó.js Manual O Manual do PHP O Manual do Python O Manual do [React](#)