

Manual do CSS: um guia prático de CSS para desenvolvedores



Tradutor: Nielda Karla



Autor: Flávio Copes (em inglês)

THE CSS HANDBOOK



Artigo original: [The CSS Handbook: A Handy Guide to CSS for Developers](#)

Escrevi este artigo para ajudar você a aprender CSS rapidamente e a se familiarizar com os tópicos avançados de CSS.

Muitas vezes, o CSS é rapidamente descartado como uma coisa fácil de aprender pelos desenvolvedores ou algo que você simplesmente aprende quando precisa estilizar rapidamente uma página ou aplicação. Por esse motivo, muitas vezes é aprendido na hora ou aprendemos as coisas isoladamente no momento em que precisamos usá-las. Isso pode ser uma grande fonte de frustração quando descobrimos que a ferramenta simplesmente não faz o que queremos.

Este artigo ajudará você a se familiarizar com o CSS e a obter uma visão geral dos principais recursos modernos que você pode usar para estilizar suas páginas e aplicações.

Espero ajudar você a se familiarizar com o CSS e a estar pronto rapidamente para usar essa ferramenta incrível, que permite criar designs impressionantes na Web.

[Clique aqui para obter uma versão em inglês em PDF/ePub/Mobi deste artigo para lê-lo off-line](#)

CSS é a abreviação de *Cascading Style Sheets*. Ele é uma das bases da construção da web. Sua história remonta aos anos 90 e, junto com o HTML, mudou muito desde o seu começo despretensioso.

Como crio sites desde antes do CSS existir, pude ver sua evolução.

O CSS é uma ferramenta incrível. Nos últimos anos, ele cresceu muito, introduzindo muitos recursos fantásticos como o CSS Grid, o Flexbox e as propriedades personalizadas do CSS.

Este manual destina-se a um público amplo.

Primeiro, o público iniciante. Eu explico CSS do zero de modo sucinto, mas abrangente, para que você possa usar este livro para aprender CSS desde o básico.

Em seguida, o público profissional. O CSS é frequentemente considerado uma coisa secundária para se aprender, especialmente por desenvolvedores de JavaScript. Eles sabem que o CSS não é uma linguagem de programação real. Eles são programadores e, portanto, não devem se preocupar em aprender CSS da maneira certa. Também escrevi este livro para esse público.

Em seguida, a pessoa que conhece CSS há alguns anos, mas ainda não teve a oportunidade de aprender as coisas novas nele. Falaremos bastante sobre os novos recursos do CSS, os 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 viver, saber como o CSS funciona pode ajudar a poupar algumas dores de cabeça quando você precisar entendê-lo de vez em quando, por exemplo, para ajustar uma página da web.

Obrigado por obter este e-book. Meu objetivo com ele é fornecer uma visão geral rápida e abrangente do CSS.

Flávio

Você pode entrar em contato comigo por e-mail, pelo endereço flavio@flaviocopes.com, ou pelo Twitter, [@flaviocopes](https://twitter.com/flaviocopes).

Meu site na web é flaviocopes.com.

Tabela de conteúdo

- [INTRODUÇÃO AO CSS](#)
- [UMA BREVE HISTÓRIA DO CSS](#)
- [ADICIONANDO CSS A UMA PÁGINA HTML](#)
- [SELETORES](#)
- [CASCATA](#)
- [ESPECIFICIDADE](#)
- [HERANÇA](#)
- [IMPORTAR](#)

- [SELETORES DE ATRIBUTO](#)
- [PSEUDOCLASSES](#)
- [PSEUDOELEMENTOS](#)
- [CORES](#)
- [UNIDADES](#)
- [URL](#)
- [CALC](#)
- [BACKGROUNDS](#)
- [COMENTÁRIOS](#)
- [PROPRIEDADES PERSONALIZADAS](#)
- [FONTES](#)
- [TIPOGRAFIA](#)
- [BOX MODEL](#)
- [BORDER](#)
- [PREENCHIMENTO](#)
- [MARGEM](#)
- [DIMENSIONAMENTO DA CAIXA](#)
- [DISPLAY](#)
- [POSICIONAMENTO](#)
- [FLOAT E CLEAR](#)
- [Z-INDEX](#)
- [CSS GRID](#)
- [FLEXBOX](#)
- [TABELAS](#)
- [CENTRALIZANDO](#)
- [LISTAS](#)
- [MEDIA QUERIES E DESIGN RESPONSIVO](#)
- [FEATURE QUERIES](#)
- [FILTROS](#)
- [TRANSFORMAÇÕES](#)
- [TRANSIÇÕES](#)
- [ANIMAÇÕES](#)
- [NORMALIZANDO O CSS](#)
- [LIDANDO COM ERROS](#)
- [PREFIXOS DOS FABRICANTES](#)
- [CSS PARA IMPRESSÃO](#)
- [CONCLUSÕES](#)

INTRODUÇÃO AO CSS

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

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

Arquivos CSS contém várias regras do CSS.

Cada regra é composta por 2 partes:

- O seletor
- O bloco de declaração

O seletor é uma string que identifica um ou mais elementos da página, seguindo uma sintaxe especial sobre a qual falaremos em breve.

O bloco de declaração contém uma ou mais **declarações**, por sua vez compostas por uma **propriedade** e um **valor**.

Essas são todas as coisas que temos em CSS.

Organizar cuidadosamente as propriedades, associando-as a valores e anexando-as a elementos específicos da página usando um seletor é todo o argumento deste artigo.

Qual a aparência do CSS

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

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

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

Várias regras são empilhadas uma após a outra:

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

Um seletor pode ter como destino um ou mais itens:

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

e pode selecionar tags HTML, como acima, ou elementos do HTML que contêm um determinado atributo de classe, usando `.minha-classe` ou elementos HTML que possuem um atributo `id` específico, usando `#meu-id`.

Seletores mais avançados permitem que você escolha itens cujo atributo corresponde a um valor específico, ou também itens que respondem a pseudoclasses (falaremos mais sobre isso depois).

Ponto e vírgula

Cada regra do CSS termina com um ponto e vírgula. O ponto e vírgula **não** é opcional, exceto após a última regra. Sugiro, porém, usá-lo sempre para consistência e para evitar erros caso você adicione outra propriedade e se esqueça de colocar o ponto e vírgula na linha anterior.

Formatação e recuo

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

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

Ele é, porém, muito complicado de se ver. Atenha-se a algumas convenções, como as que você vê nos exemplos acima: coloque os seletores e as chaves de fechamento à esquerda, recue 2 espaços para cada regra, use a chave de abertura na mesma linha do seletor, separando os dois por um espaço.

O uso correto e consistente de espaçamento e recuo é uma ajuda visual para entender seu código.

UMA BREVE HISTÓRIA DO CSS

Antes de prosseguir, quero fazer uma breve recapitulação da história do CSS.

O CSS surgiu da necessidade de estilizar páginas da web. Antes da introdução do CSS, as pessoas queriam uma maneira de estilizar suas páginas da web, que pareciam muito semelhantes e "acadêmicas" naquela época. Você não podia fazer muito em termos de personalização.

O HTML 3.2 introduziu a opção de definir cores incorporadas como atributos de elemento HTML e tags de apresentação, como `center` e `font`, mas isso escalou rapidamente para uma situação longe do 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 se parecer no navegador.

O CSS está evoluindo continuamente. O CSS que você usou há 5 anos pode estar desatualizado, pois novas técnicas idiomáticas do CSS surgiram e os navegadores mudaram.

É difícil imaginar quando o CSS nasceu e como a web era diferente.

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

As páginas eram estilizadas usando HTML, com tags de apresentação especiais, como `bold` (negrito) e atributos especiais, a maioria dos quais, agora, está obsoleta.

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 construía um site especificamente para um dos navegadores, porque cada um introduzia tags diferentes e não padrão para dar mais poder e oportunidades.

Logo, as pessoas perceberam a necessidade de um modo de estilizar as páginas, para que funcionassem em todos os navegadores.

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

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

Desde então, o trabalho começou no nível 3 do CSS. O CSS Working Group decidiu dividir cada recurso e trabalhar neles separadamente, em módulos.

Os navegadores não eram especialmente rápidos na implementação do CSS. Tivemos que esperar até 2002 para que o primeiro navegador implementasse a especificação CSS completa: o Internet Explorer para Mac, conforme bem descrito neste [artigo do CSS Tricks](#) (em inglês).

O Internet Explorer implementou o modelo de caixas incorretamente desde o início, o que fez com que levasse anos de trabalho até se conseguir ter o mesmo estilo aplicado consistentemente em todos os navegadores. Tivemos que usar vários truques e hacks para fazer os navegadores renderizarem as coisas como queríamos.

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

Não temos mais números oficiais de lançamento do CSS agora, mas o CSS Working Group lança um "snapshot" dos módulos que atualmente são considerados estáveis e prontos para serem incluídos nos navegadores. Este é o [snapshot mais recente, de fevereiro de 2023](#).

O CSS Nível 3 é a base para o CSS que escrevemos hoje. Temos muitos outros recursos construídos sobre ele.

ADICIONANDO CSS A UMA PÁGINA HTML

O CSS pode ser anexado a uma página HTML de diferentes maneiras.

1: Usando a tag `link`

A tag `link` é uma maneira de incluir um arquivo CSS. Esta é a maneira preferida de usar o CSS do modo como ele deve ser usado: um arquivo CSS é incluído em todas as páginas do seu site e a alteração de uma linha desse arquivo afeta a apresentação de todas as páginas do site.

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

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

O atributo `rel` também é necessário, pois informa ao navegador que tipo de arquivo estamos vinculando.

2: Usando a tag `style`

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

```
<style>  
...nossa 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" o CSS em um arquivo separado ou adicionar uma linha especial de CSS apenas a um arquivo.

3: Estilos inline

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

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

Exemplo:

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

SELETORES

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

Seletores básicos

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

Podemos segmentar esse elemento usando este seletor `p`, que segmenta todos os elementos usando a tag `p` na página. Uma regra CSS simples para conseguir o que queremos é:

Podemos segmentar (**target**) esse elemento usando este seletor `p`, que marca todos os elementos usando a tag `p` na página. Uma regra CSS simples para conseguir o que queremos é:

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

Cada tag HTML possui 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 possuem 2 atributos que são muito usados dentro do CSS para associar estilo a um elemento específico da página: `class` e `id`.

Há uma grande diferença entre os dois: dentro de um documento HTML, você pode repetir o mesmo valor `class` em vários elementos, mas só pode usar um `id` uma vez. Dessa forma, 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 ids usando o símbolo `#`.

Exemplo usando uma classe:

```
<p class="nome-do-pet">  
    Roger  
</p>  
  
.nome-do-pet {  
    color: yellow;  
}
```

Exemplo usando um id:

```
<p id="nome-do-pet">  
    Roger  
</p>  
  
#nome-do-pet {  
    color: yellow;  
}
```

Combinando seletores

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

Como selecionar um elemento com uma classe ou id

Você pode selecionar um elemento específico que tenha uma classe ou id anexado.

Exemplo usando uma classe:

```
<p class="nome-do-pet">  
  Roger  
</p>  
  
.nome-do-pet {  
  color: yellow;  
}
```

Exemplo usando um id:

```
<p id="nome-do-pet">  
  Roger  
</p>  
  
#nome-do-pet {  
  color: yellow;  
}
```

Por que você faria 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.

Selecionando várias classes

Você pode selecionar um elemento com uma classe específica usando `.nome-da-classe`, como você viu anteriormente. Você pode selecionar um elemento com 2 (ou mais) classes combinando os nomes das classes separados por um ponto, sem espaços.

Exemplo:

```
<p class="nome-do-pet roger">  
  Roger  
</p>  
  
.nome-do-pet.roger {  
  color: yellow;  
}
```

Combinando classes e IDs

Do mesmo modo, você pode combinar uma classe e um id.

Exemplo:

```
<p class="nome-do-pet" id="roger">  
  Roger  
</p>  
  
.nome-do-pet#roger {  
  color: yellow;  
}
```

Agrupando seletores

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

Exemplo:

```
<p>  
  O nome do meu pet é:  
</p>  
<span class="nome-do-pet">  
  Roger  
</span>  
  
p, .nome-do-pet {  
  color: yellow;  
}
```

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

```
p,  
.nome-do-pet {  
  color: yellow;  
}
```

Acompanhe a árvore do documento com seletores

Vimos como selecionar 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 do documento. Por exemplo, se você tiver uma tag `span` aninhada dentro de uma tag `p`, você pode selecionar esse `span` e não aplicar o estilo a outros elementos `span` que não estejam dentro de uma tag `p`:

```
<span>  
  Olá!  
</span>  
<p>
```

```
O nome do meu pet é:  
<span class="nome-do-pet">  
    Roger  
</span>  
</p>  
  
p span {  
    color: yellow;  
}
```

Veja como usamos um espaço entre os dois tokens, `p` e `span`.

Isso funciona mesmo se o elemento à direita tiver vários níveis de profundidade.

Para restringir a dependência ao primeiro nível, você pode usar o símbolo `>` entre os dois tokens:

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

Nesse caso, se um `span` não for o primeiro filho do elemento `p`, ele não terá a nova cor aplicada.

Filhos diretos terão o estilo aplicado:

```
<p>  
    <span>  
        Este span fica amarelo  
    </span>  
    <strong>  
        <span>  
            Este span não ficará.  
        </span>  
    </strong>  
</p>
```

Os seletores de irmãos adjacentes nos permitem estilizar um elemento apenas se for precedido por um elemento específico. Fazemos isso usando o operador `+`:

Exemplo:

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

Isso atribuirá a cor amarela a todos os elementos `span` precedidos por um elemento `p`:

```
<p>Este é um parágrafo</p>  
<span>Este é um span amarelo</span>
```

Temos muito mais seletores que podemos usar:

- seletores de atributo
- seletores de pseudoclasses
- Seletores de pseudoelementos

Aprenderemos tudo sobre eles nas próximas seções.

CASCATA

Cascata é um conceito fundamental do CSS. Afinal, está no próprio nome, o primeiro C de CSS – *Cascading Style Sheets* (que em português, poderíamos traduzir como "folha de estilos em cascata") – deve ser uma coisa importante.

O que 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 definidas em vários lugares.

Isso é feito levando em consideração:

- especificidade
- importância
- herança
- ordem no arquivo

A cascata também resolve conflitos de estilo.

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

Mesmo que você tenha apenas um arquivo CSS carregado em sua página, há outro CSS que fará parte do processo. Temos o CSS do navegador (também chamado de *user agent*). Os navegadores vêm com um conjunto padrão de regras, todas diferentes de um navegador para o outro.

É aí que o CSS entra em jogo.

O navegador aplica as folhas de estilo do usuário, que também podem ser aplicadas por extensões do navegador.

Todas essas regras começam a valer durante a renderização da página.

Veremos agora os conceitos de especificidade e herança.

ESPECIFICIDADE

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

Por exemplo, vamos falar sobre este elemento:

```
<p class="nome-do-pet">  
  Roger  
</p>
```

Podemos ter

```
.nome-do-pet {  
  color: yellow;  
}
```

Outra regra que visa `p`, que define a cor para outro valor:

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

Por fim, uma terceira regra que visa `p.nome-do-pet`. Qual regra terá precedência sobre as outras e por quê?

Aqui, entra 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.

Como calcular a especificidade

A especificidade é calculada usando uma convenção.

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

Da mesma forma que funciona para números no sistema decimal: `1 0 0 0` é maior que `0 1 0 0`.

Slot 1

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

Aumentamos esse valor quando temos um seletor de elemento. Um elemento é um nome de tag. Se você tiver mais de um seletor de elemento na regra, incrementa de acordo o valor armazenado neste 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 */
```

Slot 2

O segundo slot é incrementado por 3 coisas:

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

Sempre que uma regra atende a uma delas, incrementamos o valor da segunda coluna da direita.

Exemplo:

```
.nome {}          /* 0 0 1 0 */
.usuarios .nome {} /* 0 0 2 0 */
[href$='.pdf'] {}   /* 0 0 1 0 */
:hover {}           /* 0 0 1 0 */
```

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

```
div .nome {}      /* 0 0 1 1 */
a[href$='.pdf'] {} /* 0 0 1 1 */
.images img:hover {} /* 0 0 2 1 */
```

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

```
.nome {}          /* 0 0 1 0 */
.nome.nome {}      /* 0 0 2 0 */
.nome.nome.nome {} /* 0 0 3 0 */
```

Slot 3

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

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

Exemplo:

```
#nome {}          /* 0 1 0 0 */
.usuario #nome {} /* 0 1 1 0 */
#nome span {}     /* 0 1 0 1 */
```

Slot 4

O slot 4 é afetado por estilos incorporados aos elementos do HTML diretamente. Qualquer estilo incorporado assim terá precedência sobre as regras definida em um arquivo CSS externo ou dentro da tag `style` no `header` página.

Exemplo:

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

Mesmo que qualquer outra regra no CSS defina a cor, essa regra de estilo *inline* será aplicada. A única exceção é se `!important` for usado, o que preenche um 5º slot.

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 `!important` em uma regra do CSS tornará essa regra a mais importante, de acordo com as regras de especificidade. A única maneira de outra regra ter precedência é ter `!important` também e ter maior especificidade nos outros slots menos importantes.

Dicas

Em geral, você deve usar a quantidade de especificidade necessária, mas não mais do que isso. Desse modo, você pode criar outros seletores para sobrescrever as regras definidas pelas regras anteriores sem enlouquecer.

`!important` é uma ferramenta altamente debatida que o CSS nos oferece. Muitos especialistas em CSS defendem seu uso. Eu me pego usando especialmente quando estou tentando algum estilo e uma regra do CSS tem tanta especificidade que preciso usar `!important` para fazer com que o navegador aplique meu novo CSS.

Geralmente, porém, `!important` não se deve usar em seus arquivos CSS.

Usar o atributo `id` para estilizar o 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 mais poderosas (você pode ter várias classes para um elemento e uma classe pode ser reutilizada várias vezes).

Ferramentas para calcular a especificidade

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

Ele é útil especialmente se você está tentando entender algo, pois pode ser uma boa ferramenta de feedback.

HERANÇA

Quando você define algumas propriedades em um seletor em CSS, elas são herdadas por todos os filhos desse seletor.

Eu disse *algumas*, 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 modo muito mais conciso, já que não precisamos definir explicitamente essa propriedade novamente em cada filho.

Algumas outras propriedades fazem mais sentido *não* sendo herdadas.

Pense nas fontes: você não precisa aplicar a `font-family` a cada tag da sua página. Você define a fonte na tag do `body` e cada filho a herda, juntamente com outras propriedades.

Faz pouco sentido, por outro lado, herdar a propriedade `background-color`.

Propriedades que são herdadas

Aqui está uma lista das propriedades que são herdadas. A lista não é completa, mas essas regras são as mais populares e que você, provavelmente, usará com frequência:

- `border-collapse`
- `border-spacing`
- `caption-side`
- `color`
- `cursor`
- `direction`
- `empty-cells`
- `font-family`
- `font-size`

- font-style
- font-variant
- font-weight
- font-size-adjust
- font-stretch
- font
- letter-spacing
- line-height
- list-style-image
- list-style-position
- list-style-type
- list-style
- orphans
- quotes
- tab-size
- text-align
- text-align-last
- text-decoration-color
- text-indent
- text-justify
- text-shadow
- text-transform
- visibility
- white-space
- widows
- word-break
- word-spacing

Eu peguei essa lista de um [artigo do Sitepoint](#) sobre herança de CSS.

Forçando a herança de propriedades

Se você tiver uma propriedade que não é herdada por padrão e quiser que seja em herdada por um elemento filho, o que pode fazer?

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

Exemplo:

```
body {  
    background-color: yellow;
```

```
}
```

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

Fazendo com que propriedades NÃO sejam herdadas

Pelo contrário, você pode ter uma propriedade que é herdada e desejar que isso não aconteça.

Você pode usar a palavra-chave `revert` para reverter isso. Nesse caso, o valor é revertido para o valor original que o navegador forneceu em sua folha de estilo padrão.

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

Outros valores especiais

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

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

IMPORTAR

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

Aqui está a maneira de usar a importação:

```
@import url(arquivo.css)
```

`url()` pode gerenciar URLs absolutos ou relativos.

Uma coisa importante que você precisa saber é que as diretivas `@import` devem ser colocadas antes de qualquer outra regra de CSS no arquivo. Caso contrário, elas serão ignoradas.

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

```
@import url(arquivo.css) all;
@import url(arquivo-para-tela.css) screen;
@import url(arquivo-para-impressao.css) print;
```

SELETORES DE ATRIBUTOS

Já apresentamos vários seletores CSS básicos: usando seletores de elemento, classe, id, como combiná-los, como selecionar várias classes, como estilizar vários seletores na mesma regra, como seguir a hierarquia da página com seletores de elementos filhos, de filhos diretos e de irmãos adjacentes.

Nesta seção, analisaremos seletores de atributo. Falaremos sobre seletores de pseudoclasse e de pseudoelemento nas próximas 2 seções.

Seletores de presença de atributo

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

Podemos verificar se um elemento possui um atributo usando a sintaxe `[]`. `p[id]` selecionará todas as tags `p` na página que possuem um atributo `id`, independentemente de seu valor:

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

Seletores de valor de atributo exato

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

```
p[id="meu-id"] {  
    /* ... */  
}
```

Corresponder uma porção do valor do atributo

Enquanto `=` nos permite verificar o valor exato, temos outros operadores:

- `*=` verifica se o atributo contém a parcial
- `^=` verifica se o atributo começa com o parcial
- `$=` verifica se o atributo termina com o parcial
- `|=` verifica se o atributo começa com a parcial e vem seguido de um traço (comum em classes, por exemplo), ou contém apenas a parcial
- `~=` verifica se a parcial está contida no atributo, mas separada por espaços do restante

Todas as verificações mencionadas **diferenciam maiúsculas de minúsculas**.

Se você adicionar um `i` logo antes do colchete de fechamento, a verificação não fará distinção entre maiúsculas e minúsculas. É suportado em muitos navegadores, mas não em todos. Verifique <https://caniuse.com/#feat=css-case-insensitive>.

PSEUDOCOCLASSES

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

Elas começam com um único símbolo de dois pontos `:`.

Elas podem ser usadas como parte de um seletor e são muito úteis para estilizar links ativos ou visitados, por exemplo, alterar o estilo ao passar o mouse, focar ou direcionar o primeiro filho ou linhas ímpares. É muito útil em diversos casos.

Estas são as pseudoclasses mais populares que você provavelmente usará:

Pseudo class	Targets
<code>:active</code>	an element being activated by the user (e.g. clicked). Mostly used on links or buttons
<code>:checked</code>	a checkbox, option or radio input types that are enabled
<code>:default</code>	the default in a set of choices (like, option in a select or radio buttons)
<code>:disabled</code>	an element disabled
<code>:empty</code>	an element with no children
<code>:enabled</code>	an element that's enabled (opposite to <code>:disabled</code>)
<code>:first-child</code>	the first child of a group of siblings
<code>:focus</code>	the element with focus
<code>:hover</code>	an element hovered with the mouse
<code>:last-child</code>	the last child of a group of siblings
<code>:link</code>	a link that's not been visited
<code>:not()</code>	any element not matching the selector passed. E.g. <code>:not(span)</code>
<code>:nth-child()</code>	an element matching the specified position
<code>:nth-last-child()</code>	an element matching the specific position, starting from the end
<code>:only-child</code>	an element without any siblings
<code>:required</code>	a form element with the <code>required</code> attribute set
<code>:root</code>	represents the <code>html</code> element. It's like targeting <code>html</code> , but it's more specific. Useful in CSS Variables .
<code>:target</code>	the element matching the current URL fragment (for inner navigation in the page)
<code>:valid</code>	form elements that validated client-side successfully
<code>:visited</code>	a link that's been visited

Vamos mostrar um exemplo. É um exemplo bem comum, na verdade. Você deseja estilizar um link. Então, cria uma regra de CSS para direcionar o elemento `a`:

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

As coisas parecem funcionar bem, até você clicar 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 está azul.

Por que isso acontece?

Porque o link quando clicado muda de estado, indo para o estado `:active`. Quando foi visitado, ele está no estado `:visited` para sempre, até que o usuário limpe o histórico de navegação.

Portanto, para tornar o link amarelo correto 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 direcionar filhos pares ou ímpares com `:nth-child(odd)` e `:nth-child(even)`.

Ele é comumente usado em listas para colorir as linhas ímpares de maneira 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 `:nth-child(-n+3)`. Ou você pode estilizar 1 em cada 5 elementos com `:nth-child(5n)`.

Algumas pseudoclasses são usadas apenas para impressão, como `:first`, `:left`, `:right`, para que você possa direcionar a primeira página, todas as páginas da esquerda e todas as páginas da direita, que geralmente têm estilos ligeiramente diferentes.

PSEUDOELEMENTOS

Pseudoelementos são usados para estilizar uma parte específica de um elemento.

Eles começam com 2 símbolos de dois pontos `::`.

Às vezes, você os verá sendo usados apenas com 1 símbolo de dois pontos, mas essa é apenas uma sintaxe suportada por motivos de compatibilidade com versões anteriores. Você deve usar 2 vezes os dois pontos para distingui-los das pseudoclasses.

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

Aqui está a lista dos pseudoelementos:

Pseudo-element	Targets
::after	creates a pseudo-element after the element
::before	creates a pseudo-element before the element
::first-letter	can be used to style the first letter of a block of text
::first-line	can be used to style the first line of a block of text
::selection	targets the text selected by the user

Vamos dar um exemplo. Digamos que você queira tornar a primeira linha de um parágrafo um pouco maior no tamanho da fonte, algo comum em tipografia:

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

Talvez você queira que a apenas primeira letra seja mais grossa:

```
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.

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

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

.myElement::before {
  content: "Oláááá!";
}
```

CORES

Por padrão, uma página HTML é renderizada pelos navegadores da web com pouquíssimas cores.

Temos o fundo branco, a cor preta de fonte e os links azuis. É isso.

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

Temos estas propriedades:

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

Todas elas aceitam um valor de cor, que pode ser de diferentes formas.

Cores com nome

Primeiro, temos palavras-chave do CSS que definem as cores. O CSS começou com 16, mas hoje existe 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`
- `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`

Além das cores acima, temos `transparent` e `currentColor`. Essa última aponta para a propriedade `color` e, por exemplo, é útil para fazer com que a `border-color` herde o valor daquela propriedade.

Elas são definidas no [módulo de cores do CSS, nível 4](#). Elas não diferenciam maiúsculas de minúsculas.

A Wikipédia tem [uma tabela](#) (em inglês) que permite escolher a cor perfeita pelo nome.

Cores com nomes não são a única opção.

RGB e RGBa

Você pode usar a função `rgb()` para calcular uma cor a partir de sua notação RGB, que define a cor com base em suas tonalidades de vermelho, verde e azul. De 0 a 255:

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

`rgba()` permite adicionar o canal alfa para inserir uma parte transparente. Isso pode ser um número de 0 a 1:

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

Notação hexadecimal

Outra opção é expressar as tonalidades de RGB das cores na notação hexadecimal, que é composta por 3 blocos.

O preto, que é `rgb(0, 0, 0)` é expresso como `#000000` ou `#000` (podemos abbreviar os 2 números para 1 se forem iguais).

Branco, ou `rgb(255, 255, 255)`, pode ser expresso como `#ffffff` ou `fff`.

A notação hexadecimal nos permite expressar um número de 0 a 255 em apenas 2 dígitos, pois podem ir de 0 a "15" (f, nessa notação).

Podemos adicionar o canal alfa adicionando mais 1 ou 2 dígitos no final, por exemplo `#00000033`. Nem todos os navegadores suportam a notação abreviada, então use todos os 6 dígitos para expressar a parte RGB.

HSL e HSLa

Esta é uma adição mais recente ao CSS.

HSL = Hue-Saturation-Lightness (Matiz, saturação e brilho).

Nesta notação, preto é `hsl(0, 0%, 0%)` e branco é `hsl(0, 0%, 100%)`.

Se você está mais familiarizado com HSL do que RGB devido ao seu conhecimento anterior, você pode definitivamente usá-lo.

Você também tem `hsla()`, que adiciona o canal alfa à mistura. Novamente, um número de 0 a 1: `hsl(0, 0%, 0%, 0.5)`

UNIDADES

Uma das coisas que você usará todos os dias em CSS são as unidades. Elas são usadas para definir comprimentos, preenchimentos, margens, alinhar elementos e assim por diante.

As unidades são `px`, `em`, `rem` ou porcentagens.

Elas estão em todo lugar. Existem algumas obscuros também. Analisaremos cada uma delas nesta seção.

Pixels

A unidade de medida mais utilizada. Na verdade, um pixel não se correlaciona com um pixel físico em sua tela, pois isso varia muito de acordo com o dispositivo (pense em dispositivos de alto DPI em comparação com dispositivos sem retina).

Existe uma convenção que faz com que esta unidade funcione consistentemente em todos os dispositivos.

Porcentagens

Outra medida muito útil, as porcentagens permitem especificar valores em porcentagens da propriedade correspondente desse elemento pai.

Exemplo:

```
.parent {  
    width: 400px;  
}  
  
.child {  
    width: 50%; /* = 200px */  
}
```

Unidades de medida do mundo real

Temos aquelas unidades de medida que são traduzidas do mundo exterior. Na tela, elas não têm muita utilidade, mas podem ser úteis para imprimir folhas de estilo. Elas são:

- `cm` centímetro (mapeia para 37,8 pixels)
- `mm` milímetro (0,1 cm)
- `q` quarto de milímetro
- `in` polegada (mapeia para 96 pixels)
- `pt` ponto (1 polegada = 72 pontos)
- `pc` paica (1 paica = 12 pontos)

Unidades relativas

- `em` é o valor atribuído ao `font-size` desse elemento, portanto, seu valor exato muda entre os elementos. Não muda dependendo da fonte utilizada, apenas do tamanho da fonte. Na tipografia, isso mede a largura da letra `m`.
- `rem` é semelhante ao `em`, mas em vez de variar o tamanho da fonte do elemento atual, ele usa o tamanho da fonte do elemento raiz (`html`). Você define o tamanho da fonte uma vez e `rem` será uma medida consistente em toda a página.
- `ex` é como `em`, mas ao invés de medir a largura de `m`, mede a altura da letra `x`. Pode mudar dependendo da fonte usada e do tamanho da fonte.
- `ch` é como `ex`, mas, em vez de medir a altura de `x`, mede a largura de `0` (zero).

Unidades da viewport

- `vw` a unidade de largura da viewport representa uma porcentagem da largura da viewport (área de visualização). `50vw` significa 50% da largura da viewport.
- `vh` a unidade de altura da viewport representa uma porcentagem da altura da viewport. `50vh` significa 50% da altura da viewport.
- `vmin` a unidade mínima da viewport representa o mínimo entre a altura ou a largura em termos de porcentagem. `30vmin` é 30% da largura ou altura atual, dependendo de qual for menor.
- `vmax` a unidade máxima da viewport representa o máximo entre a altura ou a largura em termos de porcentagem. `30vmax` é 30% da largura ou altura atual, dependendo de qual for maior.

Unidades de fração

`fr` são unidades de fração e são usadas no CSS Grid para dividir o espaço em frações. Falaremos sobre elas no contexto do CSS Grid mais adiante.

URL

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

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

```
}
```

Neste caso, utilizei um URL relativo, que busca o arquivo na pasta onde está definido o arquivo do CSS.

Eu poderia voltar um nível:

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

Também poderia entrar em uma pasta:

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

Se quisesse, poderia carregar um arquivo a partir da raiz do domínio onde o CSS está hospedado:

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

Por fim, poderia usar um URL absoluto para carregar um recurso externo:

```
div {  
  background-image: url(https://meu-site.com/teste.png);  
}
```

CALC

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

É assim que funciona:

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

Ele retorna um valor de comprimento, portanto, pode ser usado em qualquer lugar em que você espera um valor de pixel.

Você pode realizar

- adições usando `+`
- subtrações usando `-`
- multiplicações usando `*`
- divisões usando `/`

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

Exemplo:

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

BACKGROUNDS

O plano de fundo (*background*) de um elemento pode ser alterado usando várias propriedades do CSS:

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

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

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

```
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 o URL de localização da imagem:

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

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

Outros valores são

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

Ao usar uma imagem como plano de fundo, você desejará definir a posição da imagem usando a propriedade `background-position: left, right, center` são todos valores válidos para o eixo X (horizontal), enquanto `top` e `bottom` servem para o eixo Y (vertical):

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

Se a imagem for menor que o plano de fundo, você precisa definir o comportamento usando `background-repeat`. Também é necessário definir se ele deve repetir no eixo X, com `repeat-x`, no eixo Y, com `repeat-y` ou repetir, com `repeat`, em ambos os eixos. Esse último é o valor padrão. Outro valor é o sem repetição, ou `no-repeat`.

`background-origin` permite que você escolha onde o plano de fundo deve ser aplicado: a todo o elemento incluindo preenchimento (padrão) usando `padding-box`, a todo o elemento incluindo a borda usando `border-box`, ao elemento sem o preenchimento usando `content-box`.

Com `background-attachment`, podemos anexar o plano de fundo à viewport, para que a rolagem não afete o plano de fundo:

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

Por padrão, o valor é `scroll`. Existe outro valor chamado `local`. A melhor forma de visualizar o comportamento deles é por [este Codepen](#).

A última propriedade de background é `background-size`. Podemos usar 3 palavras-chave: `auto`, `cover` e `contain`. `auto` é o padrão.

`cover` expande a imagem até que todo o fundo preencha todo o elemento.

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

Você também pode especificar um valor de comprimento e, nesse caso, definir a largura da imagem de fundo (a altura é definida automaticamente):

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

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

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

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

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 como alternativa:

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

Você também pode definir um gradiente como plano de fundo:

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

COMENTÁRIOS

O CSS oferece a capacidade de escrever comentários em um arquivo CSS ou na tag de `style` no header da página

O formato é `/* este é um comentário */`. Os comentários são ao estilo dos comentários em C (ou ao estilo do JavaScript, se preferir).

Abaixo, vemos um comentário de várias linhas. Até que você adicione o token `*/` de fechamento, todas as linhas encontradas após a abertura serão comentadas.

Exemplo:

```
#nome { display: block; } /* Boa regra! */

/* #nome { display: block; }

#nome {
    display: block; /*
    color: red;
    */
}
```

CSS não tem comentários em linha, com `//`, como o em C ou o JavaScript.

ATENÇÃO - se você adicionar `//` antes de uma regra, a regra não será aplicada, parecendo que o comentário funcionou. Na verdade, o CSS detectou um erro de sintaxe e, devido ao seu funcionamento, ignorou a linha com o erro e foi direto para a linha seguinte.

Conhecer essa abordagem permite que você escreva comentários em linha propositalmente, embora você deva ter cuidado porque não pode adicionar texto aleatório como em um comentário em bloco.

Por exemplo:

```
// Boa regra!
#nome { display: block; }
```

No caso acima, devido ao funcionamento do CSS, a regra `#nome` é, na verdade, comentada. Você pode encontrar [mais detalhes aqui](#) (texto em inglês) se achar isso interessante. Para evitar fazer algo errado, evite usar comentários em linha e confie nos comentários em bloco.

PROPRIEDADES PERSONALIZADAS

Nos últimos anos, os pré-processadores de CSS tiveram muito sucesso. Era muito comum que os primeiros projetos, sem muitas regras ou restrições, começassem com Less ou Sass. Essa ainda é uma tecnologia muito popular.

Os principais benefícios dessas tecnologias são, na minha opinião:

- Permitirem que você aninhe seletores

- Fornecerem uma funcionalidade de importação fácil
- Permitirem o uso de variáveis

O CSS moderno tem um novo recurso poderoso chamado **CSS Custom Properties**, também comumente conhecido como **variáveis em CSS**.

O CSS não é uma linguagem de programação, como o [JavaScript](#) (texto em inglês), o Python, o PHP, o Ruby ou o Go. Nessas linguagens, as variáveis são essenciais para fazer algo útil. O CSS é muito limitado no que pode fazer e é, principalmente, uma sintaxe declarativa para informar aos navegadores como eles devem exibir uma página HTML.

Uma variável, no entanto, é uma variável: um nome que se refere a um valor. Variáveis em CSS ajudam a reduzir repetições e inconsistências em seu CSS, centralizando a definição de valores.

Além disso, elas apresentam um recurso exclusivo que os pré-processadores de CSS nunca terão: **você pode acessar e alterar o valor de uma variável do CSS usando JavaScript**.

Noções básicas de uso de variáveis

Uma variável CSS é definida com uma sintaxe especial, anexando dois traços a um nome (`--nome-da-variavel`), depois dois pontos e um valor. Assim:

```
:root {  
  --cor-primaria: yellow;  
}
```

(mais sobre `:root` depois)

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

```
p {  
  color: var(--cor-primaria)  
}
```

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

```
:root {  
  --padding-padrao: 30px 30px 20px 20px;  
  --cor-padrao: red;  
  --background-padrao: #fff;  
}
```

Criar variáveis dentro de qualquer elemento

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

```
:root {  
  --cor-padrao: red;  
}  
  
body {  
  --cor-padrao: red;  
}  
  
main {  
  --cor-padrao: red;  
}  
  
p {  
  --cor-padrao: red;  
}  
  
span {  
  --cor-padrao: red;  
}  
  
a:hover {  
  --cor-padrao: red;  
}
```

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

Escopo de variáveis

Adicionar variáveis a um seletor as torna disponíveis para todos os filhos dele.

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

```
:root {  
  --cor-primaria: yellow;  
}
```

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

No contexto de um documento HTML, usar o seletor `:root` aponta para o elemento `html`, exceto que `:root` tem maior especificidade (tem prioridade).

No contexto de uma imagem SVG, `:root` aponta para a tag `svg`.

Adicionar uma propriedade personalizada de CSS a `:root` a torna disponível para todos os elementos da página.

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

```
.container {  
  --cor-secundaria: yellow;  
}
```

Usá-la fora desse elemento não vai funcionar.

As variáveis podem ser reatribuídas:

```
:root {  
  --cor-primaria: yellow;  
}  
  
.container {  
  --cor-primaria: blue;  
}
```

Fora de `.container`, `--cor-primaria` será *amarelo*, mas, dentro, será *azul*.

Você também pode atribuir ou sobrescrever uma variável dentro do HTML usando **estilos inline**:

```
<main style="--cor-primaria: orange;">  
  <!-- ... -->  
</main>
```

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

Interagindo com um valor de variável CSS usando JavaScript

O mais legal das variáveis do CSS é a capacidade de acessá-las e editá-las usando JavaScript.

Veja como definir um valor de variável usando JavaScript simples:

```
const elemento = document.getElementById('meu-elemento')  
elemento.style.setProperty('--nome-da-variavel', 'valor')
```

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

```
const estilos = getComputedStyle(document.documentElement)  
const valor = String(estilos.getPropertyValue('--nome-da-variavel')).trim()
```

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

```
const elemento = document.getElementById('meu-elemento')  
const estilos = getComputedStyle(elemento)  
const valor = String(estilos.getPropertyValue('--nome-da-variavel')).trim()
```

Manipulando 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 `position` ou um valor em `rem` para uma propriedade de cor.

Nesse caso, a linha é considerada inválida e é ignorada.

Suporte dos navegadores

O suporte dos navegadores para variáveis de CSS é muito bom, [de acordo com o Can I Use](#).

As variáveis do CSS vieram para ficar e você pode usá-las hoje se não precisar oferecer suporte ao Internet Explorer e versões antigas de outros navegadores.

Se você precisa oferecer suporte a navegadores mais antigos, pode usar bibliotecas como a [PostCSS](#) ou a [Myth](#), mas perderá a capacidade de interagir com variáveis via JavaScript ou pelas ferramentas de desenvolvedor do navegador, pois elas são transpiladas para o bom e velho CSS sem variáveis (e, como tal, você perde a maior parte do poder das variáveis do CSS).

Variáveis do CSS diferenciam maiúsculas de minúsculas

Esta variável:

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

é diferente desta:

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

Matemática em variáveis CSS

Para fazer contas com variáveis do CSS, você precisa usar `calc()`. Por exemplo:

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

Media queries com variáveis do CSS

Nada de especial aqui. As variáveis do CSS normalmente se aplicam a media queries:

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

Definindo um valor de fallback para var()

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

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

FONTES

No início da web, só havia um punhado de fontes para escolher.

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

O CSS ganhou muitos recursos interessantes ao longo dos anos em relação às fontes.

A propriedade `font` é a abreviação de várias propriedades:

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

Vamos ver cada uma delas e depois veremos `font`.

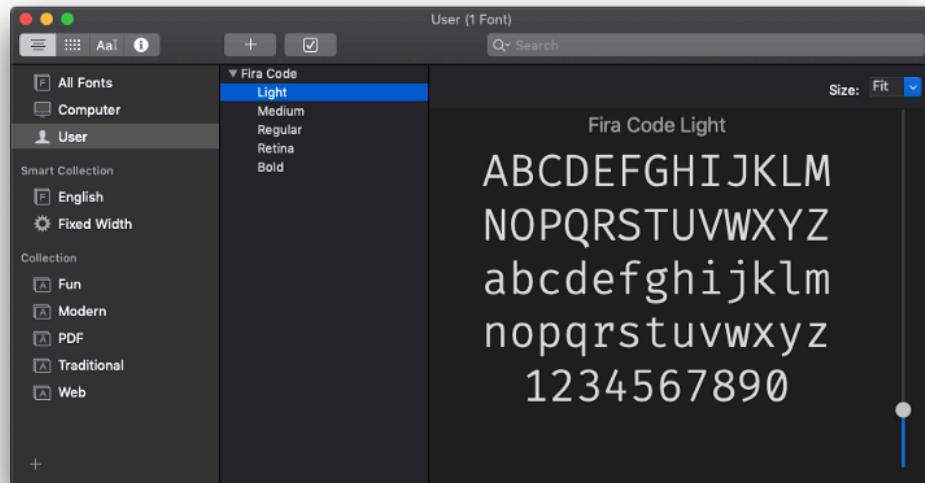
Em seguida, falaremos sobre como carregar fontes personalizadas, usando `@import` ou `@font-face`, ou carregando uma folha de estilo de fontes.

`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, claro...) de que precisamos.

Aqui está um exemplo do aplicativo Mac's Font Book. A família de fontes Fira Code hospeda várias fontes dedicadas abaixo:



Esta propriedade permite selecionar uma fonte específica, por exemplo:

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

Você pode definir vários valores. Então, a segunda opção será usada caso a primeira não possa ser usada por algum motivo (caso não seja encontrada na máquina, ou a conexão de rede para baixar a fonte falhou, por exemplo):

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

Eu usei algumas fontes específicas até agora, aquelas que chamamos de **fontes seguras para a web** (Web Safe fonts), pois são pré-instaladas em diferentes sistemas operacionais.

Nós as dividimos em fontes Serif, Sans-Serif e Monospace. Aqui está uma lista de algumas das mais populares:

Serif

- Georgia
- Palatino
- Times New Roman
- Times

Sans-Serif

- Arial
- Helvetica
- Verdana
- Geneva
- Tahoma
- Lucida Grande
- Impact
- Trebuchet MS
- Arial Black

Monospace

- Courier New
- Courier
- Lucida Console
- Monaco

Você pode usar todas elas como propriedades `font-family`, mas não há garantia de que estejam presentes em todos os sistemas. Existem outras, também, com um nível variável de suporte.

Você também pode usar nomes genéricos:

- `sans-serif` uma fonte sem ligaduras
- `serif` uma fonte com ligaduras
- `monospace` uma fonte especialmente boa para código
- `cursive` usado para simular peças manuscritas
- `fantasy` O nome diz tudo

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

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

font-weight

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

- `normal`

- `bold` negrito
- `bolder` mais negrito (em relação ao elemento pai)
- `lighter` mais leve (em relação ao elemento pai)

Ou usando as palavras-chave numéricas

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

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

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 fique pelo menos tão em negrito quanto o anterior, então você pode ter números que apontam para a mesma fonte.

`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, do mais estreito ao mais largo:

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

`font-style`

Permite aplicar um estilo itálico a uma fonte:

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

Esta propriedade também permite os valores `oblique` e `normal`. Há muito pouca diferença, se é que existe alguma, entre usar `italic` e `oblique`. A primeira é mais fácil para mim, pois o HTML já oferece um elemento `i` que significa itálico.

font-size

Esta propriedade é usada para determinar o tamanho das fontes.

Você pode passar 2 tipos de valores:

1. um valor de comprimento, como `px`, `em`, `rem` e outros, ou uma porcentagem
2. uma palavra-chave de valor predefinido

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

- `xx-small`
- `x-small`
- `small`
- `medium`
- `large`
- `x-large`
- `xx-large`
- `smaller` (menor que - relativo ao elemento pai)
- `larger` (maior que - relativo ao elemento pai)

Uso:

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

font-variant

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

- `normal`

- `inherit`
- `small-caps`

`small-caps` significa que o texto é renderizado em "letras maiúsculas menores" ao lado de suas letras maiúsculas.

font

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

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

```
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;  
}
```

Carregando fontes personalizadas usando `@font-face`

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

Essa fonte será baixada pelo navegador e usada na página. Foi uma mudança muito importante na tipografia na web — agora, podemos usar qualquer fonte que quisermos.

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

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

Uma declaração `@font-face` contém várias propriedades que usamos para definir a fonte, incluindo `src`, o URI (um ou mais URLs) 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).

As fontes geralmente estão nos formatos

- `woff` (Web Open Font Format)
- `woff2` (Web Open Font Format 2.0)
- `eot` (Embedded Open Type)
- `otf` (OpenType Font)
- `ttf` (TrueType Font)

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

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

Uma nota sobre o desempenho

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

TIPOGRAFIA

Já falamos sobre fontes, mas há mais sobre estilo de 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`

`text-transform`

Esta propriedade pode transformar como um elemento é apresentado.

Existem 4 valores válidos:

- `capitalize` colocar em maiúscula a primeira letra de cada palavra
- `uppercase` para aplicar maiúsculas no texto todo
- `lowercase` para aplicar minúsculas no texto todo
- `none` para desabilitar a transformação do texto, usado para evitar herdar a propriedade.

Exemplo:

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

`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 `solid`, `double`, `dotted`, `dashed`, `wavy`.

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;  
}
```

text-align

Por padrão, o alinhamento de texto tem o valor inicial, ou seja, o texto começa no "início", origem 0,0 (zero e zero) da caixa que o contém. Isso significa o canto superior esquerdo, em idiomas da esquerda para a direita, e o canto superior direito, em idiomas da direita para a esquerda.

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

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

vertical-align

Determina como os elementos embutidos são alinhados verticalmente.

Temos vários valores para essa propriedade. Primeiro, podemos atribuir um valor de comprimento ou porcentagem. Eles são usados para alinhar o texto em uma posição mais alta ou mais baixa (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 HTML `sub`

- `super` torna um elemento sobreescrito, simulando o resultado do elemento HTML `sup`
- `top` alinha o topo do elemento ao topo da linha
- `text-top` alinha o topo do elemento com o topo da fonte do elemento pai
- `middle` alinha o meio do elemento ao meio da linha do pai
- `bottom` alinha a parte inferior do elemento com a parte inferior da linha
- `text-bottom` alinha a parte inferior do elemento com a parte inferior da fonte do elemento pai

line-height

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

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

text-indent

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

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

text-align-last

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

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

word-spacing

Modifica o espaçamento entre cada palavra.

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

```
p {  
  word-spacing: 2px;  
}
```

```
span {  
  word-spacing: -0.2em;  
}
```

letter-spacing

Modifica o espaçamento entre cada letra.

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

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

text-shadow

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

Esta 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;  
}
```

white-space

Define como o CSS lida com o espaço em branco, novas linhas e tabulações dentro de um elemento.

Os valores válidos que reduzem o espaço em branco são:

- `normal` reduz o espaço em branco. Adiciona novas linhas quando necessário conforme o texto atinge o final do contêiner

- `nowrap` reduz 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` reduz o espaço em branco. Adiciona novas linhas quando necessário conforme 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 atinge o final do contêiner, mas preserva a quebra de linha adicionada ao texto
- `pre-wrap` preserva o espaço em branco. Adiciona novas linhas quando necessário conforme o texto atinge o final do contêiner.

tab-size

Define a largura do caractere de tabulação. Por padrão, é 8. 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;  
}
```

writing-mode

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

Os valores que você pode usar são

- `horizontal-tb` (padrão)
- `vertical-rl` o conteúdo é disposto verticalmente. As novas linhas são colocadas à esquerda das anteriores
- `vertical-lr` o conteúdo é disposto verticalmente. Novas linhas são colocadas à direita da linha anterior

hyphens

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

Os valores válidos são

- `none` (padrão)
- `manual` adiciona um hífen apenas quando já houver um hífen visível ou um hífen oculto (um caractere especial)
- `auto` adiciona hifens quando determinado que o texto pode ter um hífen.

text-orientation

Quando o `writing-mode` está no modo vertical, determina a orientação do texto.

Os valores válidos são

- `mixed` é o padrão e, se um idioma for 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

direction

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

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

word-break

Esta propriedade especifica como quebrar linhas dentro de palavras.

- `normal` (padrão) significa que o texto é quebrado apenas entre as palavras, não dentro de uma palavra
- `break-all` o navegador pode quebrar uma palavra (mas nenhum hífen é adicionado)
- `keep-all` suprimir a quebra de linha suave. Usado principalmente para texto CJK (chinês/japonês/coreano).

Falando em texto CJK, a propriedade `line-break` é usada para determinar como as linhas de texto são quebradas. Não sou um especialista nesses idiomas, então evitarei abordá-los.

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` (quebra de linha), embora não seja padrão (mas ainda funciona 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 perceber que há uma oportunidade, fará o *soft wrap* em algum lugar anterior. Nenhum hífen é adicionado, em qualquer caso.

Essa propriedade é muito semelhante a `word-break`. Podemos escolher este em idiomas ocidentais, mas `word-break` tem tratamento especial para idiomas não ocidentais.

BOX MODEL

Cada elemento do CSS é essencialmente uma caixa (Box). Cada elemento é uma caixa genérica.

O modelo de caixas explica o dimensionamento dos elementos com base em algumas propriedades do CSS.

De dentro para fora, temos:

- A área de conteúdo
- padding (preenchimento)
- border (borda)
- margin (margem)

A melhor maneira de visualizar o modelo de caixas é abrir as ferramentas de desenvolvedor do navegador e verificar como ele é exibido:

The screenshot shows the 'Box Model' panel in the Chrome DevTools. It displays a visual representation of the box model layers for an element. The layers from inside out are: content box (light blue), padding (purple), border (dark grey), and margin (yellow). The total width is 234.133px and height is 139.167px. Below the visualization, the 'Box Model Properties' section lists the following values:

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

Aqui, você pode ver como o Firefox mostra as propriedades de um elemento de `span` que destaquei. Para isso, cliquei com o botão direito do mouse nele, pressionei Inspect Element e fui para o painel Layout das ferramentas de desenvolvedor.

Veja, o espaço azul claro é a área de conteúdo. Ao redor dele, há o preenchimento, a borda e, finalmente, a margem.

Por padrão, se você definir uma largura (ou altura) no elemento, ela será aplicada à área de conteúdo. Todos os cálculos de preenchimento, borda e margem são feitos fora desse valor. Então, você precisa ter isso em mente ao fazer seus cálculos.

Mais tarde, você verá como pode mudar esse comportamento usando o Box Sizing.

BORDER

A borda é uma camada fina entre o preenchimento (padding) e a margem. Ao editar a borda, você pode fazer com que os elementos desenhem seu perímetro na tela. Você pode trabalhar em bordas usando estas propriedades:

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

A propriedade `border` pode ser usada como um atalho para todas essas propriedades.

`border-radius` é usada para criar cantos arredondados.

Você também pode usar imagens como bordas, uma habilidade dada pela tag `border-image` e suas propriedades separadas específicas:

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

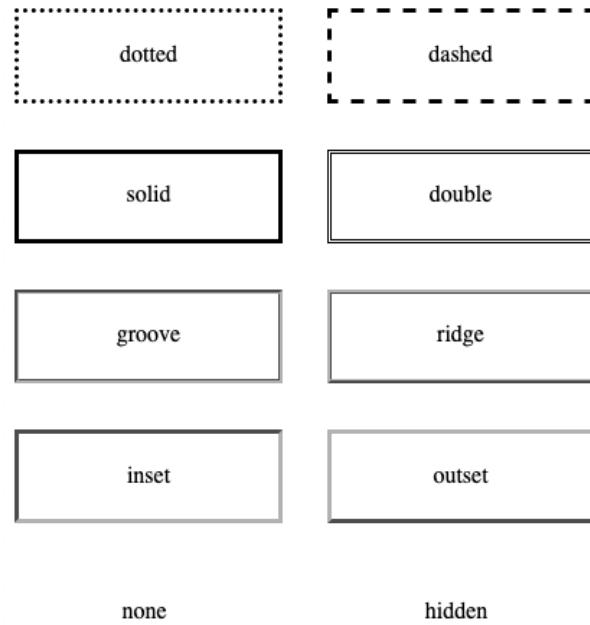
Comecemos com o `border-style`.

The border style

A propriedade `border-style` permite escolher o estilo da borda. As opções que você pode usar são:

- `dotted`
- `dashed`
- `solid`
- `double`
- `groove`

- `ridge`
- `inset`
- `outset`
- `none`
- `hidden`



Dê uma olhada neste [Codepen](#) para um exemplo ao vivo.

O padrão para o estilo é `none`. Portanto, para que a borda apareça, você precisa alterá-la para outra coisa. `solid` é uma boa escolha na maioria das vezes.

Você pode definir um estilo diferente para cada aresta usando as propriedades:

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

Você também pode usar o `border-style` com vários valores para defini-los, usando a ordem normal Superior-Direita-Inferior-Esquerda:

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

A largura da borda

`border-width` é usada para definir a largura da borda.

Você pode usar um dos valores predefinidos:

- `thin` (fina)
- `medium` (valor padrão)
- `thick` (grossa)

ou expressar um valor em pixels, em, rem ou qualquer outro valor de comprimento válido.

Exemplo:

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

Você pode definir a largura de cada borda (Superior-Direita-Inferior-Esquerda) separadamente usando 4 valores:

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

Você também pode usar as propriedades de borda específicas `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`.

A cor da borda

`border-color` é usada para definir a cor da borda.

Se você não definir uma cor, a borda por padrão será colorida usando a cor do texto no elemento.

Você pode passar qualquer valor de cor válido para `border-color`.

Exemplo:

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

Você pode definir a cor de cada borda (Superior-Direita-Inferior-Esquerda) separadamente usando 4 valores:

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

```
}
```

Você também pode usar as propriedades de borda específicas `border-top-color`, `border-right-color`, `border-bottom-color`, `border-left-color`.

Propriedade abreviada da borda

As 3 propriedades mencionadas, `border-width`, `border-style` e `border-color` podem ser definidas usando a propriedade abreviada `border`.

Exemplo:

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

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

Exemplo:

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

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-radius`, `border-top-right-radius`, `border-bottom-left-radius`, `border-bottom-right-radius`.

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 bordas.

Temos 5 propriedades:

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

Também temos a abreviação, `border-image`. Não vou entrar em muitos detalhes aqui, pois imagens como bordas precisariam de uma cobertura mais aprofundada do que posso fazer neste pequeno capítulo. Eu recomendo ler a entrada do [almanaque CSS Tricks](#) (em inglês) sobre border-image para mais informações.

Preenchimento

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

Lembrete:

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

Propriedades específicas de preenchimento

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

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

O uso delas é muito simples e não pode ser confundido. Por exemplo:

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

Usando a abreviação `padding`

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

1 valor

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

```
padding: 20px;
```

2 valores

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

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

3 valores

O uso de 3 valores aplica o primeiro ao **topo**, o segundo à **esquerda e à direita** e o terceiro à **parte inferior**.

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

4 valores

O uso de 4 valores aplica o primeiro ao **topo**, o segundo à **direita**, o terceiro à **parte inferior** e o quarto à **esquerda**.

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

Portanto, a ordem é superior-direita-inferior-esquerda.

Valores aceitos

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

MARGEM

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

Lembre-se:

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

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 delas é muito simples e não pode ser confundido. Por exemplo:

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

Usando a abreviação `margin`

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

1 valor

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

```
margin: 20px;
```

2 valores

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

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

3 valores

O uso de 3 valores aplica o primeiro ao **topo**, o segundo à **esquerda** e à **direita** e o terceiro à parte **inferior**.

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

4 valores

O uso de 4 valores aplica o primeiro ao **topo**, o segundo à **direita**, o terceiro à parte **inferior** e o quarto à **esquerda**.

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

Portanto, a ordem é superior-direita-inferior-esquerda.

Valores aceitos

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

Também aceita valores percentuais e o valor especial `auto`.

Usando `auto` para centralizar elementos

`auto` pode ser usado para dizer ao navegador para selecionar automaticamente uma margem, e é mais comumente usado para centralizar um elemento desta forma:

```
margin: 0 auto;
```

Como dito acima, usar 2 valores aplica o primeiro à parte **inferior e superior** e o segundo à **esquerda e à direita**.

A maneira moderna de centralizar elementos é usar o [Flexbox](#) e sua diretiva `justify-content: center;`

É claro que os navegadores mais antigos não implementam o Flexbox e, se você precisar dar suporte a eles, `margin: 0 auto;` ainda é uma boa escolha.

Usando 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 mova sobre os elementos anteriores a ele e, dado um valor negativo suficiente, ele se moverá para fora da página.

A margem inferior negativa move os elementos depois dela.

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

A margem esquerda negativa move o elemento à esquerda sobre os elementos que a precedem e, dado um valor negativo suficiente, ele se moverá para fora da página.

Dimensionamento da caixa (Box sizing)

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 nenhum preenchimento, borda e margem.

Essa abordagem provou ser bastante complicada de se trabalhar.

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

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

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

`content-box` é o padrão, aquele que tivemos por muito tempo antes `box-sizing` se tornar conhecido.

`border-box` é a novidade e o achado que estamos procurando. Ao definir isso em um elemento:

```
.minha-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, pois, em nossa mente, também costumamos ver isso como algo separado: a margem está fora da caixa.

Esta propriedade é uma pequena mudança, mas tem um grande impacto. O CSS Tricks até declarou um [dia internacional de conscientização do box-sizing](#) (texto em inglês), e é recomendável aplicá-lo a todos os elementos da página, fora da caixa, com isto:

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

DISPLAY

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

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 todas `table-*`)
- `flex`
- `grid`
- `list-item`

- `inline-block`
- `inline-table`
- `inline-flex`
- `inline-grid`
- `inline-list-item`

Também há outros que você provavelmente não usará, como `ruby`.

A escolha de 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 `table`, `flex` e `grid`.

`inline`

É o valor de exibição padrão para cada elemento em CSS.

Todas as tags HTML são exibidas em linha ao natural, exceto alguns elementos, como `div`, `p` e `section`, que são definidos como `block` pelo agente do usuário (o navegador).

Os elementos em linha não têm nenhuma margem ou preenchimento aplicado.

A mesma coisa vale para altura e largura.

Você *pode* adicioná-los, mas a aparência da página não mudará — eles são calculados e aplicados automaticamente pelo navegador.

`inline-block`

Semelhante ao `inline`, mas, com `inline-block`, `width` e `height` são aplicadas conforme você especificar.

`block`

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

- `div`
- `p`
- `section`

- `ul`

que são definidos como `block` pelo navegador.

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

Os valores atribuídos às propriedades `width` e `height` são respeitados, se você as definir, juntamente com `margin` e `padding`.

`none`

Usar `display: none` faz um elemento desaparecer. Ele ainda está lá no HTML, mas não é visível no navegador.

POSICIONAMENTO

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

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

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

Temos uma propriedade CSS principal: `position`.

Ela pode ter estes 5 valores:

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

Posicionamento estático

Este é o valor padrão para um elemento. Os elementos posicionados estaticamente são exibidos no fluxo normal da página.

Posicionamento relativo

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

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

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

Veja este [exemplo \(em inglês\) 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>
```

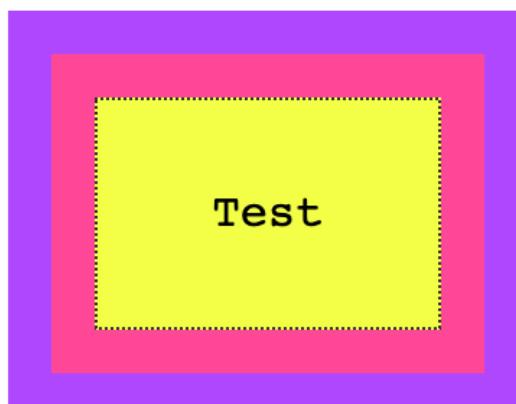
Depois, adiciono um pouco de CSS para dar algumas cores e preenchimento, mas sem afetar o posicionamento:

```
.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;
}
```

aqui está o resultado:

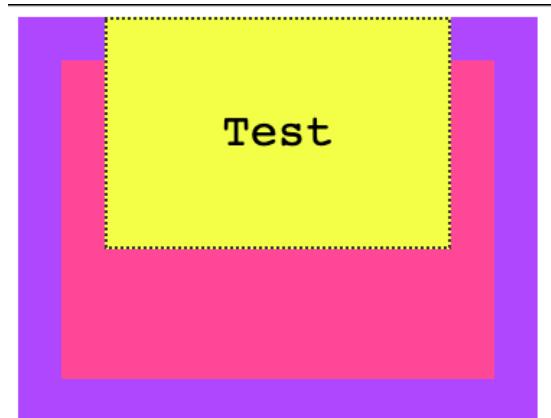


Você pode tentar adicionar qualquer uma das propriedades que mencionei antes (`top`, `right`, `bottom`, `left`) ao `.box`, e nada acontecerá. A posição é `static`.

Agora, se definirmos `position: relative` à caixa, a princípio aparentemente nada muda. Mas o elemento agora pode se mover usando as propriedades `top`, `right`, `bottom` e `left`, e agora você pode alterar a posição dele em relação ao elemento que o contém.

Por exemplo:

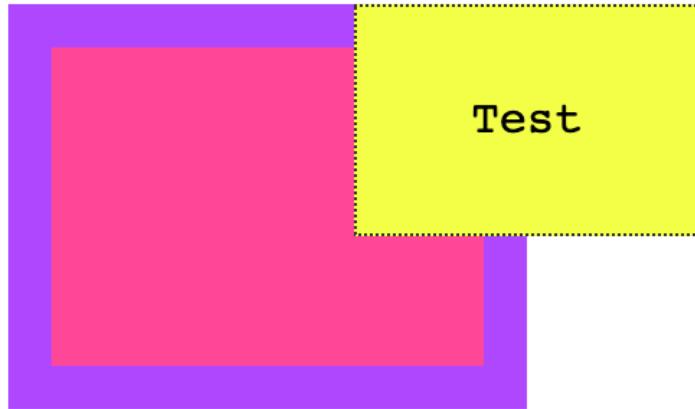
```
.box {  
    /* ... */  
    position: relative;  
    top: -60px;  
}
```



Um valor negativo para o `top` fará com que a caixa se mova para cima em relação ao seu contêiner.

Como alternativa,

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



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

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

Posicionamento absoluto

Definir `position: absolute` em um elemento o removerá do fluxo do documento.

Lembra no posicionamento relativo que notamos que o espaço originalmente ocupado por um elemento era preservado mesmo que ele fosse movido?

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

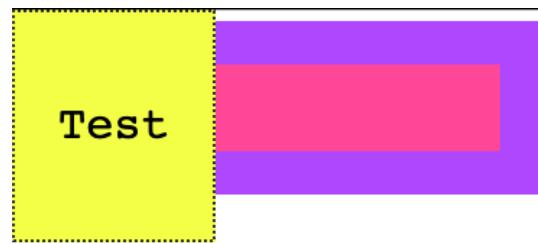
```
.box {  
    /* ... */  
    position: absolute;  
}
```



Agora, podemos mover a caixa como quisermos, usando as propriedades `top`, `right`, `bottom`, `left`:

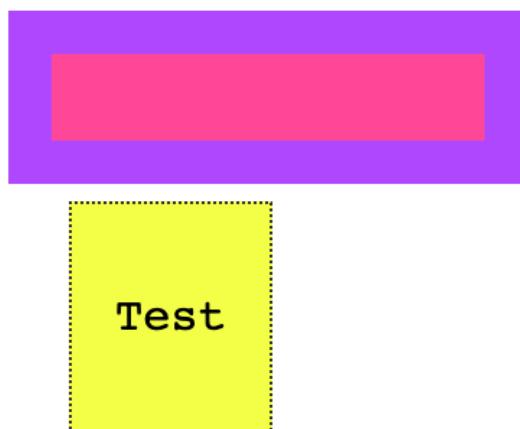
```
.box {  
    /* ... */  
    position: absolute;
```

```
    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 `position: relative` ao elemento `.child` e definirmos `top` e `left` como 0, a caixa não será posicionada na margem superior esquerda da *janela*, mas nas coordenadas 0, 0 de `.child`:

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

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



Já vimos que `.child` é estático (o padrão):

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



Como no posicionamento relativo, você pode usar o `z-index` para alterar o posicionamento do eixo z.

Posicionamento fixo

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

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 fixo em algum lugar, rolar a página não o remove da parte visível da página.

Posicionamento "sticky"

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

O componente UITableView do iOS é o que vem à mente quando penso em `position: sticky`. Você sabe quando você rola na lista de contatos e a primeira letra está presa no topo, para que você saiba que está visualizando os contatos dessa letra em particular?

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

FLOAT E CLEAR

`float` foi um tópico muito importante no passado.

Ele foi usado em muitos hacks e usos criativos porque era uma das poucas maneiras, junto com as tabelas, de realmente implementar alguns layouts. No passado, costumávamos usar o `float` para lançar a barra lateral para a esquerda, por exemplo, para mostrá-la no lado esquerdo da tela e adicionar alguma margem ao conteúdo principal.

Felizmente, os tempos mudaram e hoje temos o Flexbox e o Grid para nos ajudar no layout. Float, agora, voltou ao seu escopo original: colocar o conteúdo de um lado do elemento de contêiner e fazer com que seus irmãos apareçam ao seu redor.

A propriedade `float` dá suporte a 3 valores:

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

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

Aqui está o código:

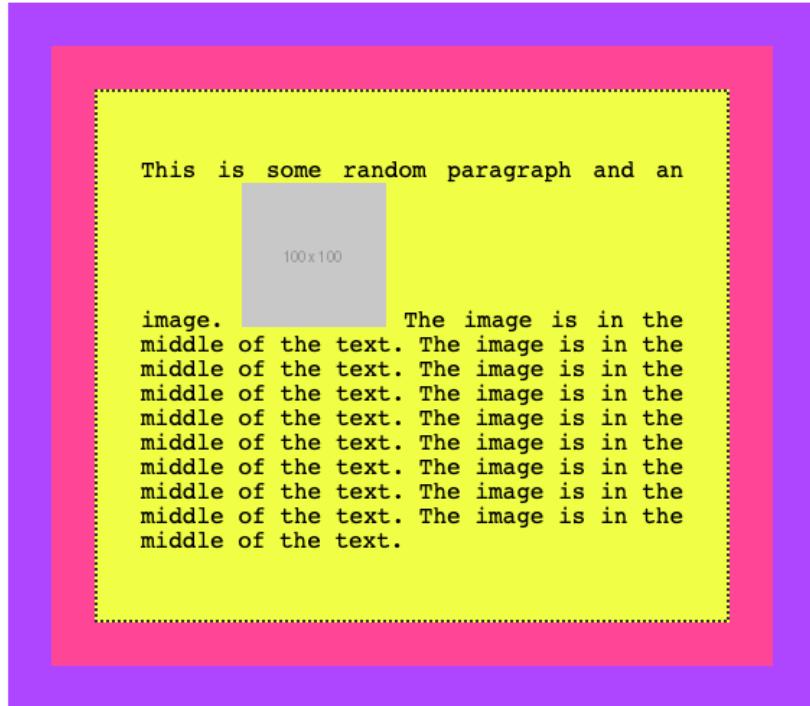
```
<div class="parent">
  <div class="child">
    <div class="box">
      <p>This is some random paragraph and an image.  The image
      </p>
    </div>
  </div>
</div>

.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;
}
```

Seu aspecto visual é:

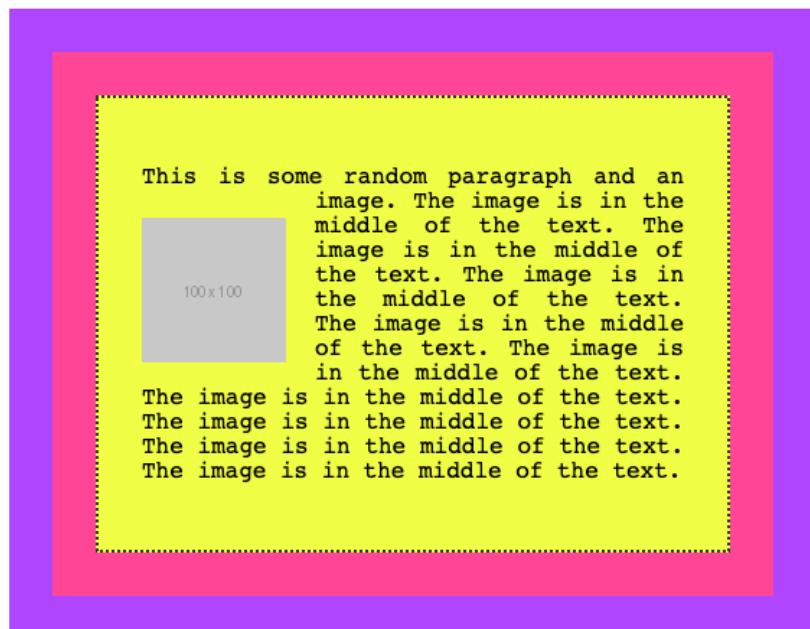


Como você pode ver, o fluxo normal, por padrão, considera a imagem *inline* e abre espaço para ela na própria linha.

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

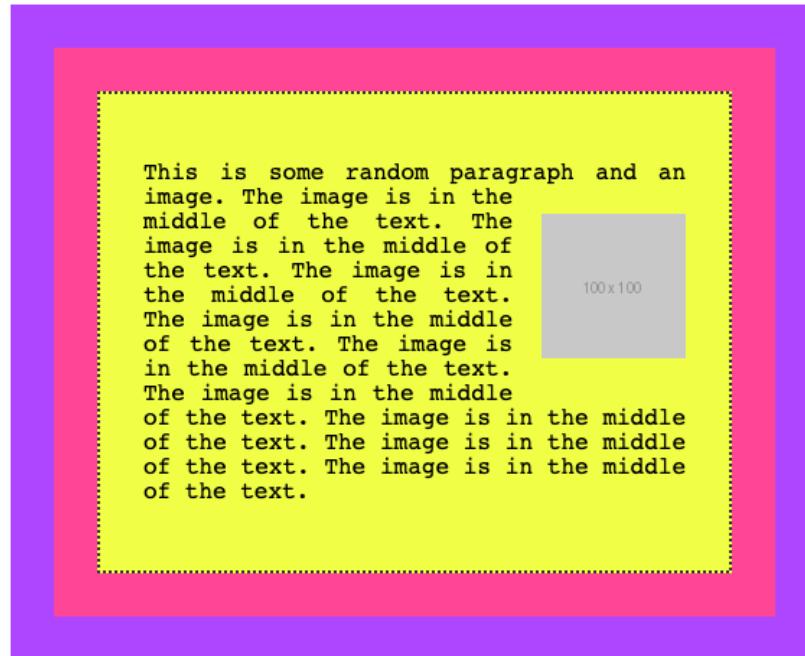
```
img {  
  float: left;  
  padding: 20px 20px 0px 0px;  
}
```

este é o resultado:



Abaixo vemos o que se obtém aplicando um `float: right` e ajustando `padding` de acordo:

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



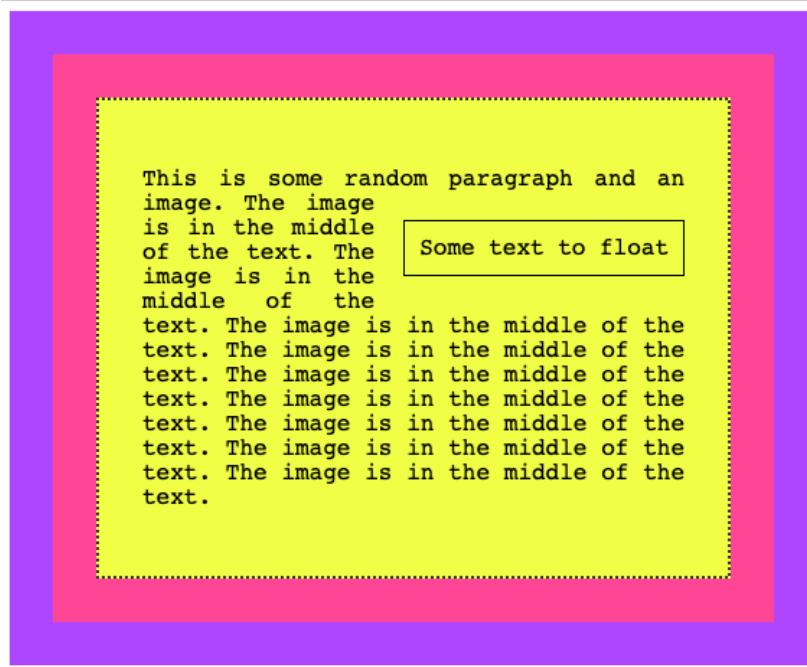
Um elemento flutuante é removido do fluxo normal da página e o outro conteúdo flui ao redor dele.

[Veja o exemplo em inglês no Codepen](#)

Você também não está limitado a imagens flutuantes. Aqui, trocamos a imagem por 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.  
    </p>  
    </div>  
  </div>  
</div>  
  
span {  
  float: right;  
  margin: 20px 0px 20px 20px;  
  padding: 10px;  
  border: 1px solid black  
}
```

Este é o resultado:

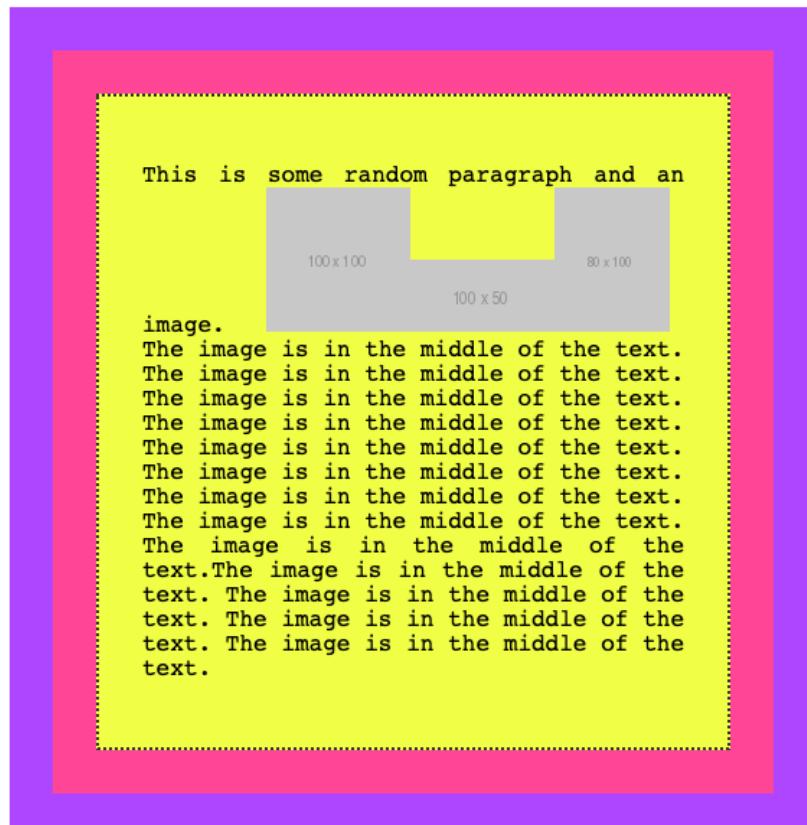


Clear

O que acontece quando você coloca `float` em mais de um elemento?

Se um elemento com float encontra outra imagem com float, por padrão, elas são empilhadas uma ao lado da outra, horizontalmente. Quando não houver mais espaço, elas começarão a ser empilhadas em uma nova linha.

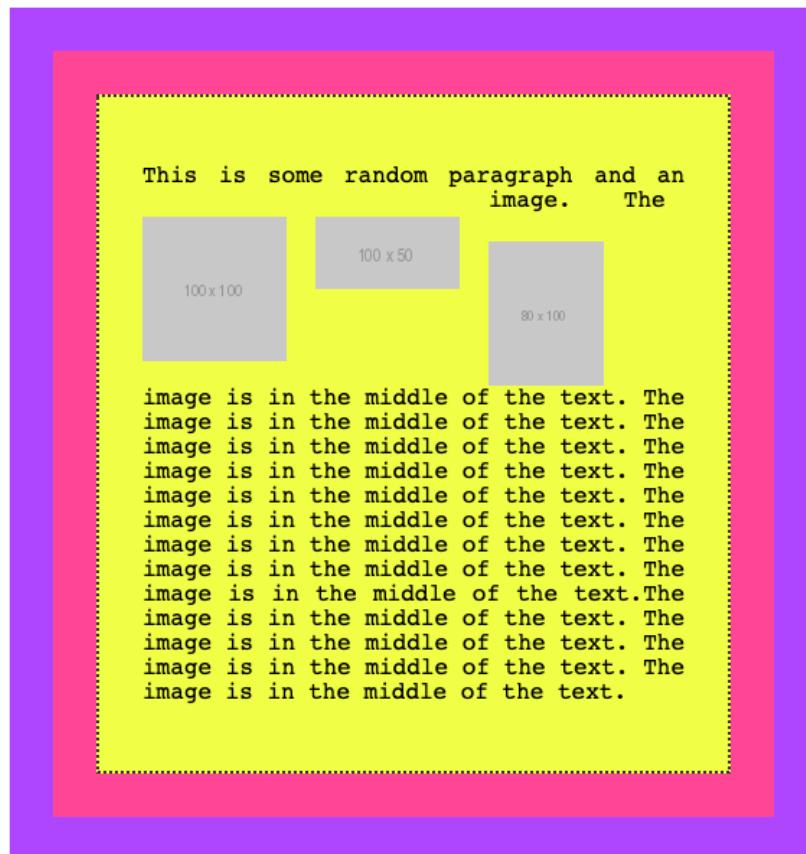
Imagine que temos 3 imagens em linha dentro de uma tag `p`:



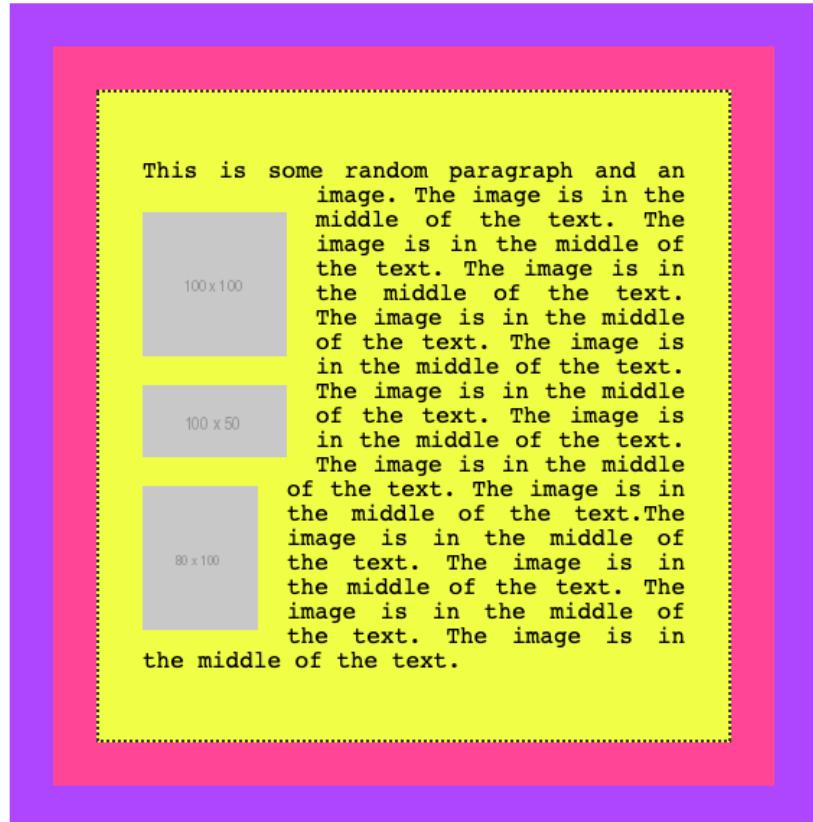
Se adicionarmos `float: left` a essas imagens:

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

Isto é o que teremos:



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



Eu usei o valor `left` para `clear`. Isso permite que:

- `left` limpe `float` à esquerda
 - `right` limpe `float` à direita
 - `both` limpe `float` à esquerda e à direita
 - `none` (padrão) desativa a "limpeza" dos `float`

Z-INDEX

Quando falamos sobre posicionamento, mencionei que você pode usar a propriedade `z-index` para controlar o posicionamento dos elementos no eixo Z.

É muito útil quando você tem vários elementos que se sobrepõem e precisa decidir quais deles estão visíveis, mais próximos do usuário e quais devem ficar ocultos atrás dos elementos visíveis.

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

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

Ao decidir qual elemento deve estar visível e qual deve ser posicionado atrás dele, o navegador faz um cálculo sobre o valor do z-index.

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

Por padrão, os elementos têm o valor `static` para a propriedade `position`. Nesse caso, a propriedade `z-index` não faz diferença – ela deve ser definida como `absolute`, `relative` ou `fixed` para funcionar.

Exemplo:

```
.minha-primeira-div {  
    position: absolute;  
    top: 0;  
    left: 0;  
    width: 600px;  
    height: 600px;  
    z-index: 10;  
}  
  
.minha-segunda-div {  
    position: absolute;  
    top: 0;  
    left: 0;  
    width: 500px;  
    height: 500px;  
    z-index: 20;  
}
```

O elemento com a classe `.minha-segunda-div` será exibido e, atrás dele, `.minha-primeira-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 os elementos no meio. Se você usar números consecutivos, precisará recalcular o `z-index` de cada elemento envolvido no posicionamento.

CSS GRID

CSS Grid é o novo membro da família 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 construir layouts usando o CSS.

Fique de olho na página do CSS Grid Layout em caniuse.com (<https://caniuse.com/#feat=css-grid>) para descobrir quais navegadores o suportam atualmente. Atualmente, todos os principais navegadores (exceto o IE, que nunca terá suporte para ele) já oferecem suporte a essa tecnologia, cobrindo 95,74% 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).

Construir layouts para a web tem sido tradicionalmente um tópico complicado.

Não vou me aprofundar nas razões dessa complexidade, que é um tópico complexo por si só. Você, porém, pode se considerar um humano de muita sorte, pois hoje em dia você tem 2 ferramentas muito poderosas e bem suportadas à sua disposição:

- **CSS Flexbox**
- **CSS Grid**

Essas 2 são as ferramentas para construir os layouts da web do futuro.

Caso você não precise oferecer suporte a navegadores antigos como IE8 e IE9, não há razão para mexer com coisas como:

- Layouts de tabela
- Floats
- Hacks com clearfix
- Hacks com `display: table`

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

O básico

O layout do CSS Grid é ativado em um elemento contêiner (que pode ser uma `div` ou qualquer outra tag) definindo `display: 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.

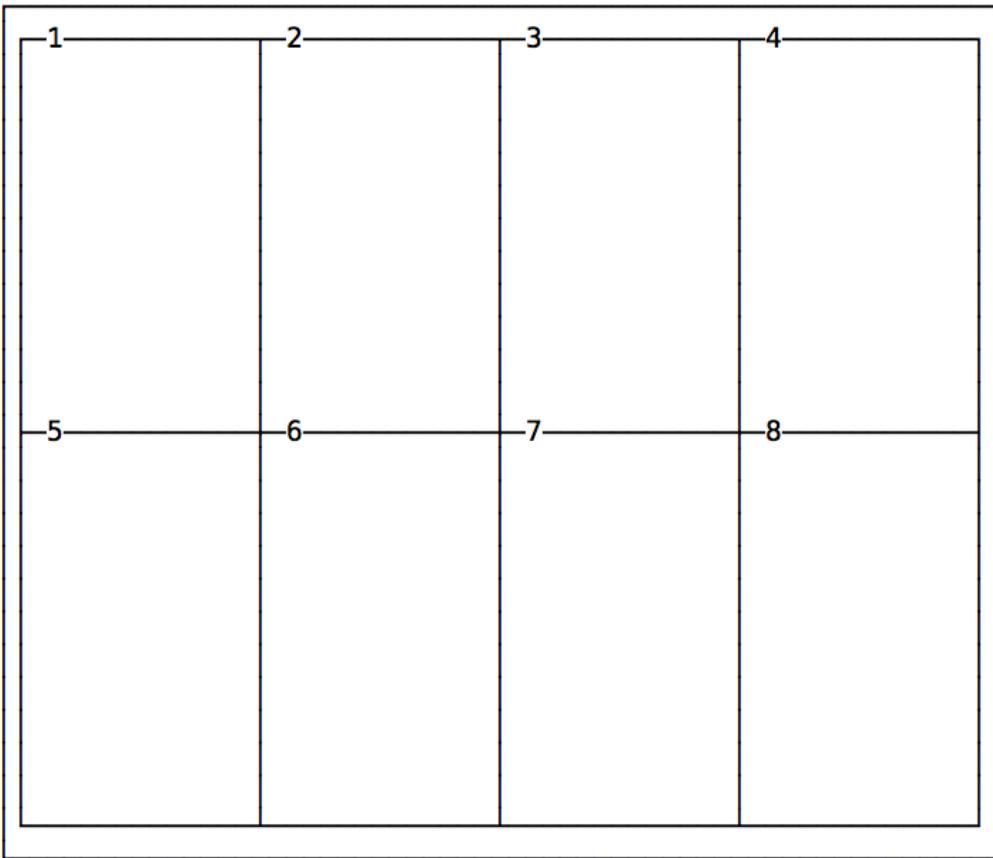
As propriedades do contêiner mais básicas são `grid-template-columns` e `grid-template-rows`.

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.

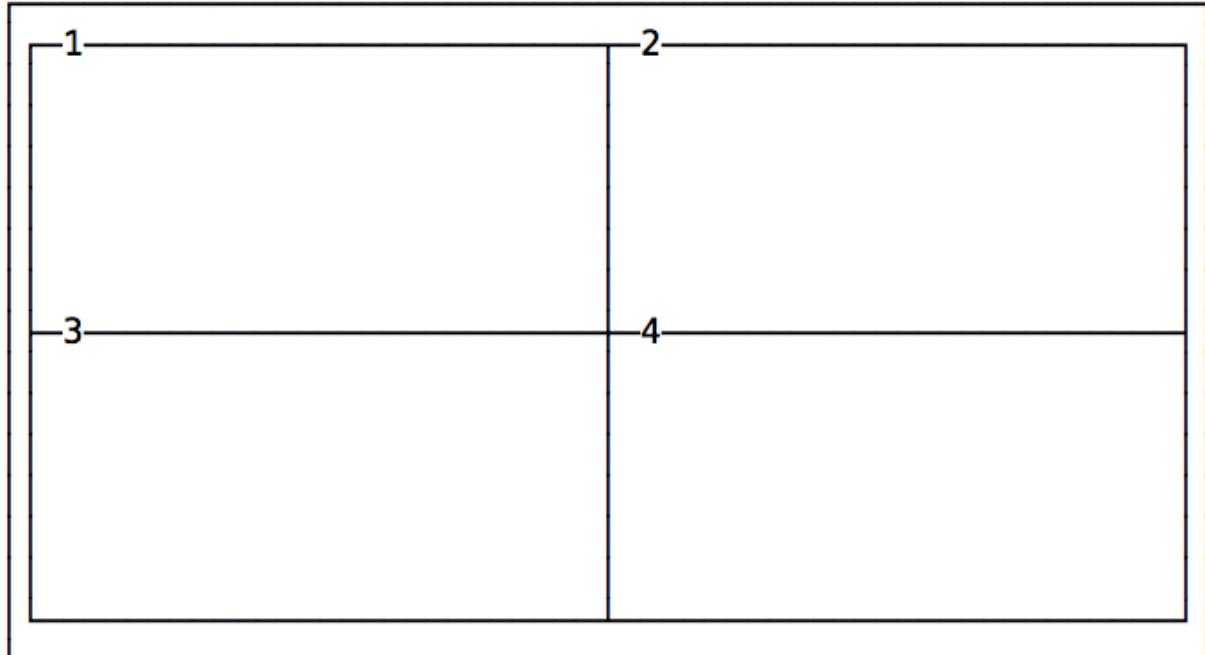
O trecho a seguir define uma grade com 4 colunas, cada uma com 200px de largura e 2 linhas com 300px de altura.

```
.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;  
}
```



Dimensões automáticas

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

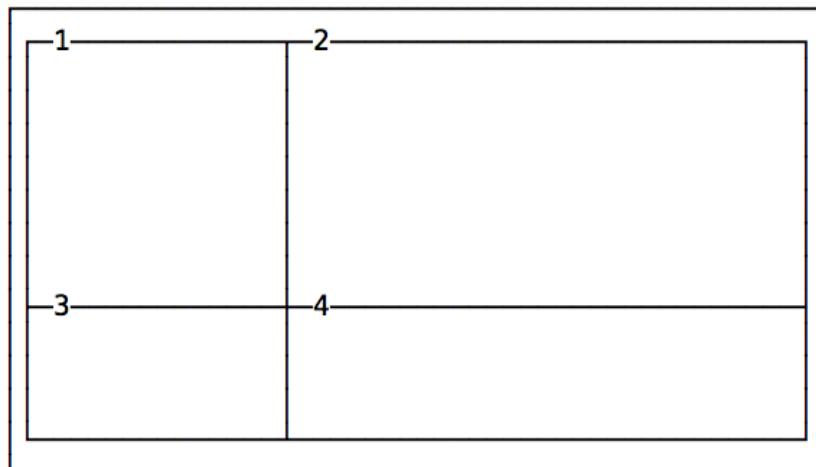
```
.container {  
  display: grid;  
  grid-template-rows: 100px auto 100px;  
}
```

Dimensões diferentes de colunas e linhas

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

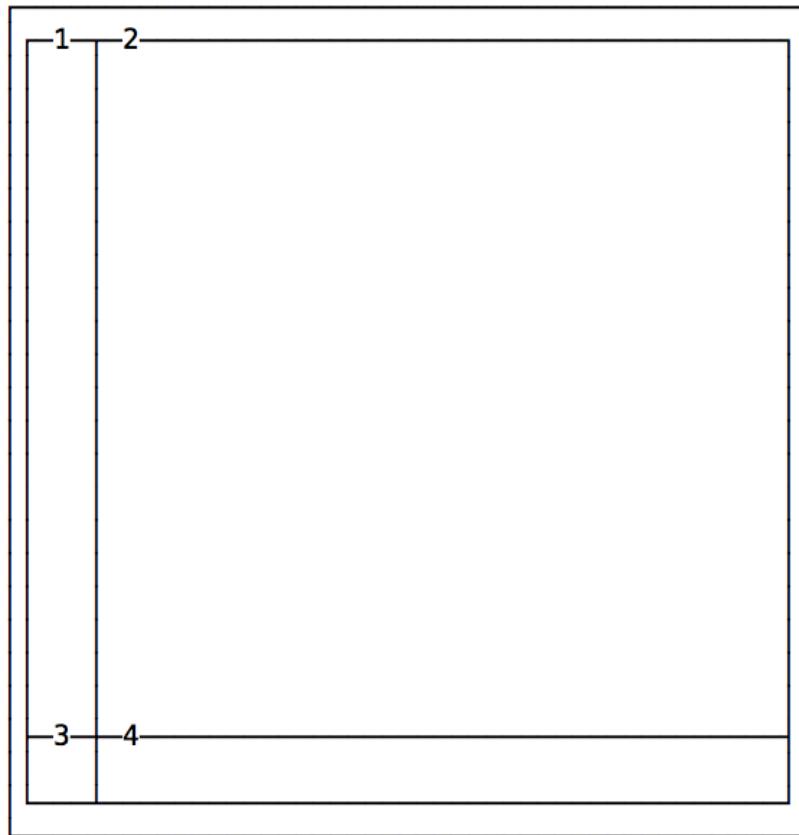
Você pode especificar qualquer valor para cada linha/coluna, para criar vários designs diferentes:

```
.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;  
}
```



Adicionando espaço entre as células

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

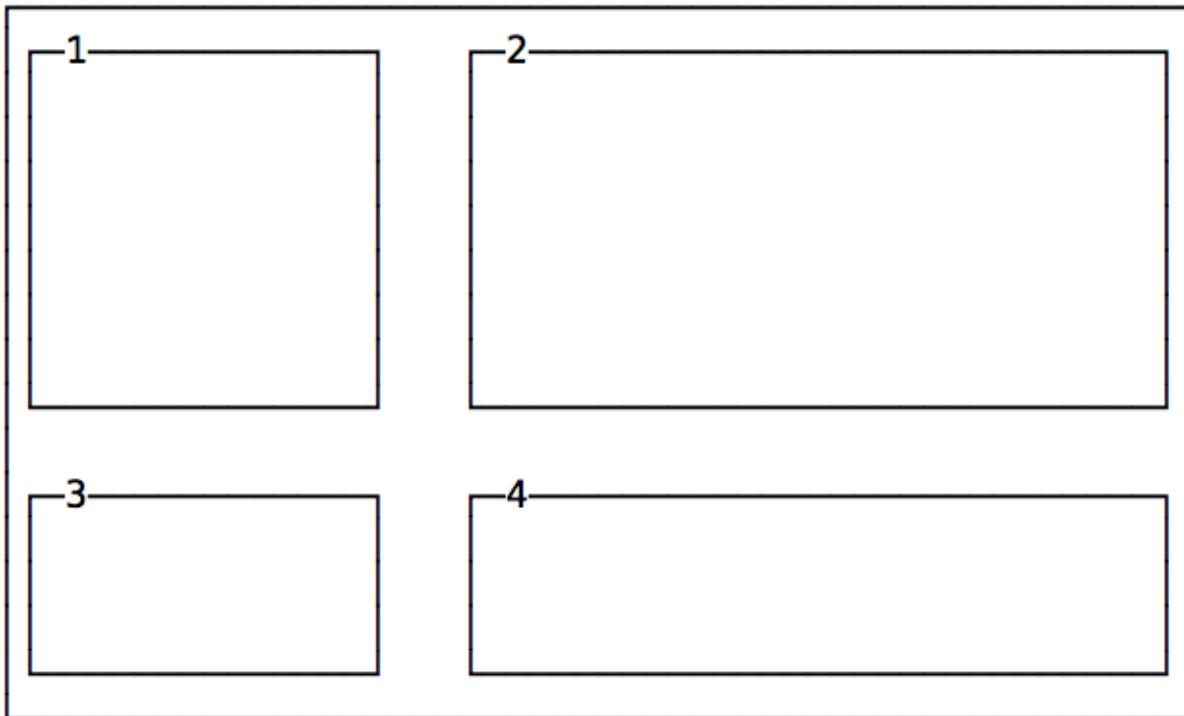
Você pode adicionar espaçamento usando estas propriedades:

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

Também é possível usar 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 abreviação:

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

Gerando itens em várias colunas e/ou linhas

Cada item da célula tem a opção de ocupar mais de uma caixa na linha, expandindo horizontalmente ou verticalmente para obter mais espaço, respeitando as proporções da grade definidas no contêiner.

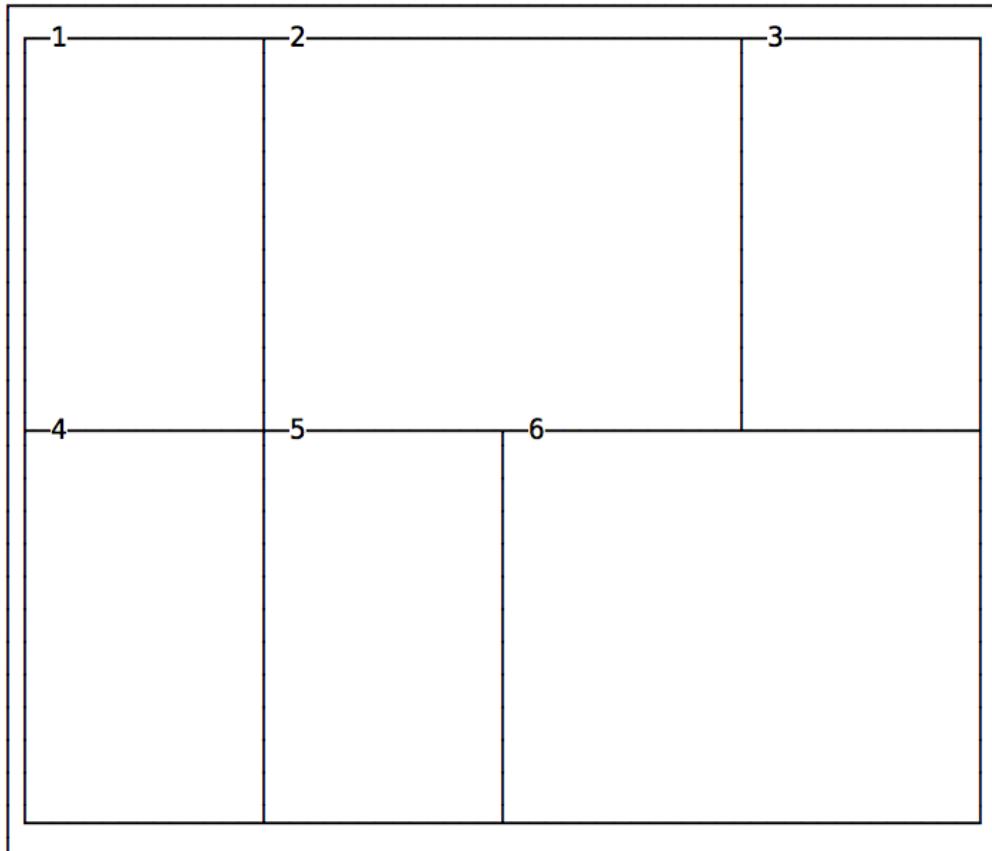
Estas são as propriedades que usaremos para isso:

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

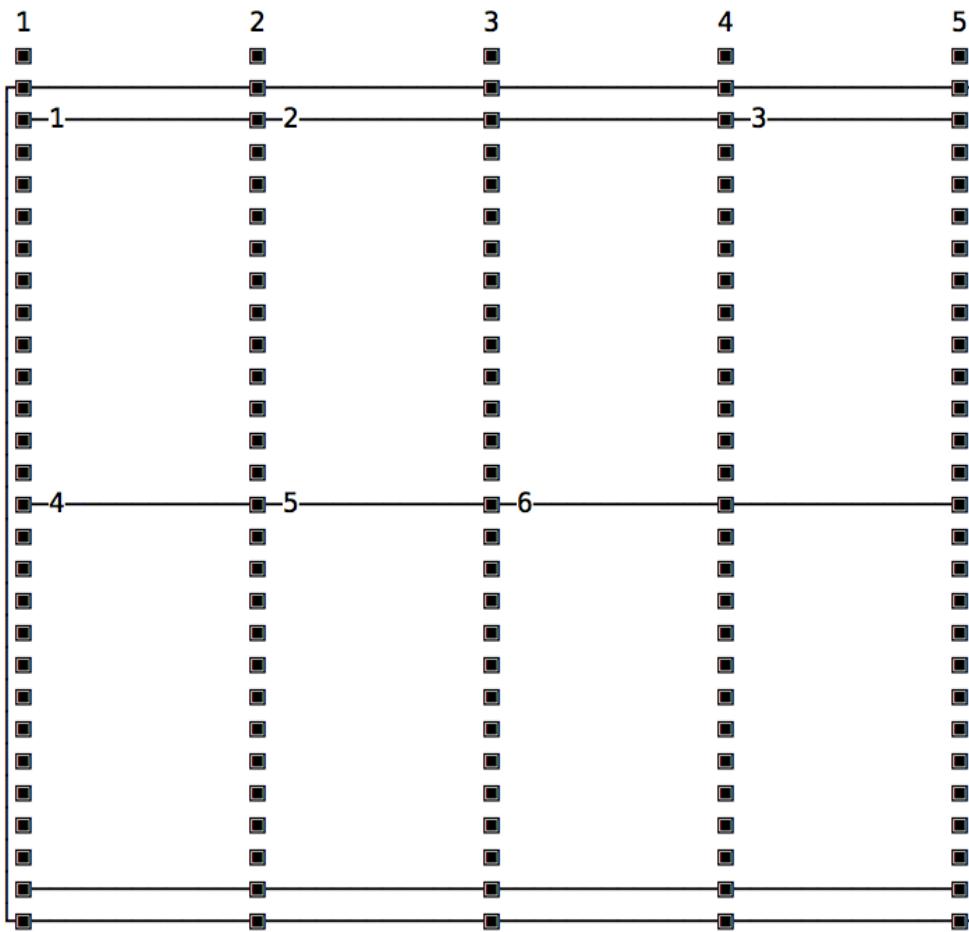
Exemplo:

```
.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;  
}
```



Os números correspondem à linha vertical que separa cada coluna, começando em 1:



O mesmo princípio se aplica ao `grid-row-start` e `grid-row-end`, exceto que, dessa vez, em vez de ocupar mais colunas, uma célula ocupa mais linhas.

Sintaxe abreviada

Essas propriedades têm uma sintaxe abreviada fornecida por:

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

O uso é simples, veja como replicar o layout acima:

```
.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 ela deve 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;  
}
```

Mais configurações do grid

Usando 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 uma.

```
.container {  
  grid-template-columns: 1fr 1fr 1fr;  
}
```

Usando porcentagens e rem

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

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

Usando `repeat()`

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

Se todas as colunas tiverem a mesma largura, você pode especificar o layout usando esta sintaxe:

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

Isso cria 4 colunas com a mesma largura.

Usando frações, temos:

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

Especificando uma largura mínima para uma linha

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

Aqui está um exemplo onde a barra lateral ocupa 1/4 da tela e nunca ocupa 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;  
}
```

É possível definir apenas um valor mínimo:

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

Posicionando elementos usando `grid-template-areas`

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

Usando `grid-template-areas`, 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-column`.

Aqui está 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 as `grid-template-areas` os definem, dependendo da propriedade `grid-area` associada a eles.

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 .";
}

```

Preencher uma página com uma grade

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

```

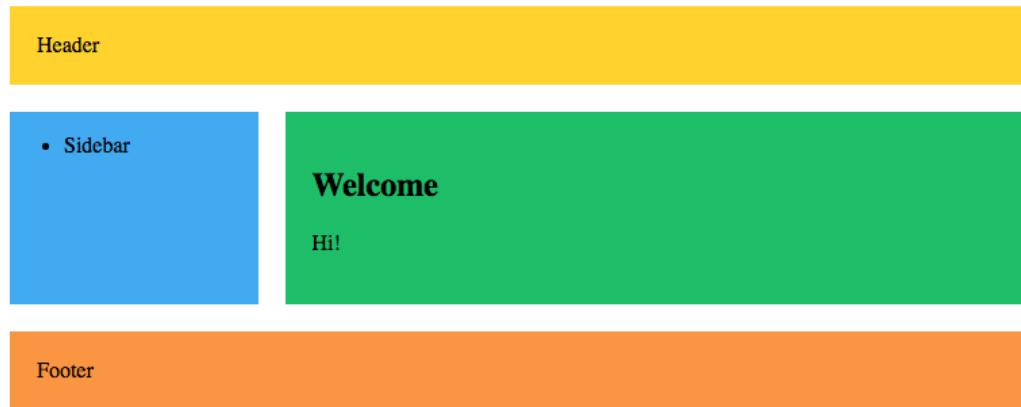
.container {
  display: grid;
  height: 100vh;
  grid-template-columns: 1fr 1fr 1fr 1fr;
}

```

```
grid-template-rows: 1fr 1fr;  
}
```

Exemplo: *header*, *sidebar*, *content* e *footer*

Aqui está um exemplo simples de uso do CSS Grid para criar um layout de site que fornece um cabeçalho no topo, 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>
```

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";
}
```

Adicionei algumas cores para deixá-lo mais bonito, mas, basicamente, atribuí a cada tag diferente o nome `grid-area`, que é usado na propriedade `grid-template-areas` em `.wrapper`.

Quando o layout é menor, podemos colocar a sidebar abaixo do conteúdo usando uma media query:

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

[Ver no CodePen](#)

Estes são os fundamentos do CSS Grid. Há muitas coisas que não incluí nesta introdução, mas queria torná-la muito simples, para que você possa começar a usar esse novo sistema de layout sem se sentir muito confuso.

FLEXBOX

O Flexbox, também chamado de módulo de caixas flexíveis, é um dos dois sistemas de layouts modernos, junto com o CSS Grid.

Comparado ao 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 juntas ao mesmo tempo.

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

A menos que você precise oferecer suporte a navegadores antigos como IE8 e IE9, o Flexbox é uma ferramenta que permite que você não precise usar

- Layouts de tabela
- Floats
- hacks com clearfix
- hacks com `display: table`

Vamos nos aprofundar no Flexbox e dominá-lo em muito pouco tempo.

Suporte dos navegadores

O Flexbox é suportado por 98,06% dos usuários. Todos os navegadores mais importantes já o implementam há anos. Portanto, navegadores mais antigos (incluindo o IE10+) dão suporte a ele:

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini *	Android *
6		53	59	7.1	44	8		1 3
7	12	54	60	8	45	8.4		1 4
8	13	55	61	9	46	9.2		1 4.1
9	14	56	62	9.1	47	9.3		1 4.3
10	15	57	63	10	48	10.2		4.4
11	16	58	64	10.1	49	10.3		4.4.4
17	59	65	11	50	11.2	all		62
18	60	66	11.1	51	11.3			
		67	TP	52				
		68						

Embora possamos ter de esperar alguns anos para que todos os usuários alcancem o CSS Grid, o Flexbox é uma tecnologia mais antiga e já pode ser usada agora.

Habilitando o Flexbox

Um layout flexbox é aplicado a um contêiner, definindo

```
display: flex;
```

Também é possível usar

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

O conteúdo dentro do contêiner será alinhado usando o Flexbox.

Propriedades do contêiner

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

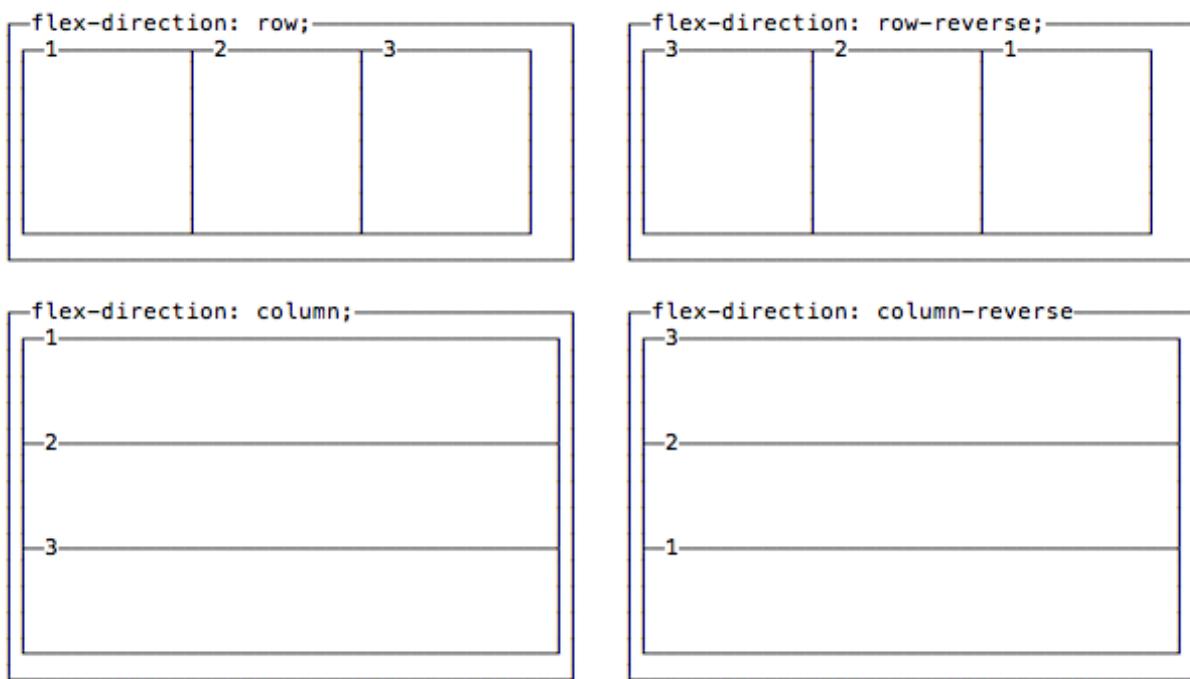
Elas são

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

Alinhar linhas ou colunas

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

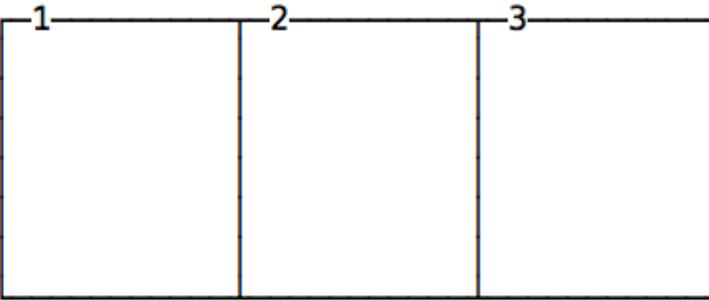
- `flex-direction: row` coloca os itens em uma **linha**, na direção do texto (da esquerda para a direita para países ocidentais)
- `flex-direction: row-reverse` coloca itens como `row`, mas na direção oposta
- `flex-direction: column` coloca itens em uma **coluna**, ordenando de cima para baixo
- `flex-direction: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



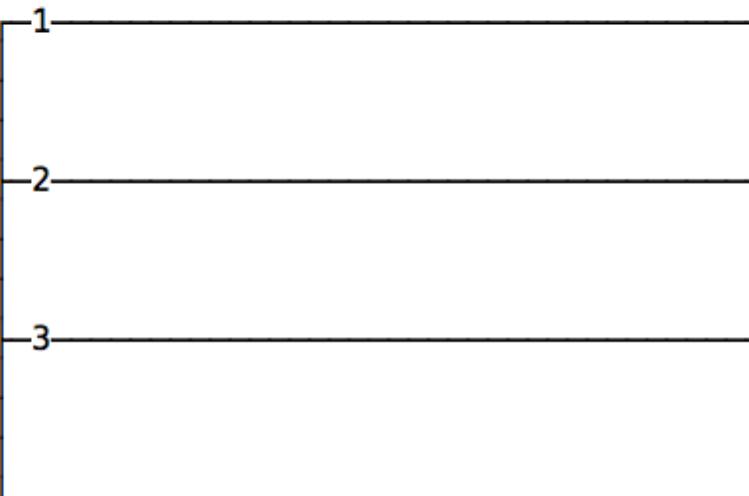
Alinhamento vertical e horizontal

Por padrão, os itens começam da esquerda se `flex-direction` é linha, e do topo se o `flex-direction` é coluna.

```
flex-direction: row;
```



```
flex-direction: column;
```



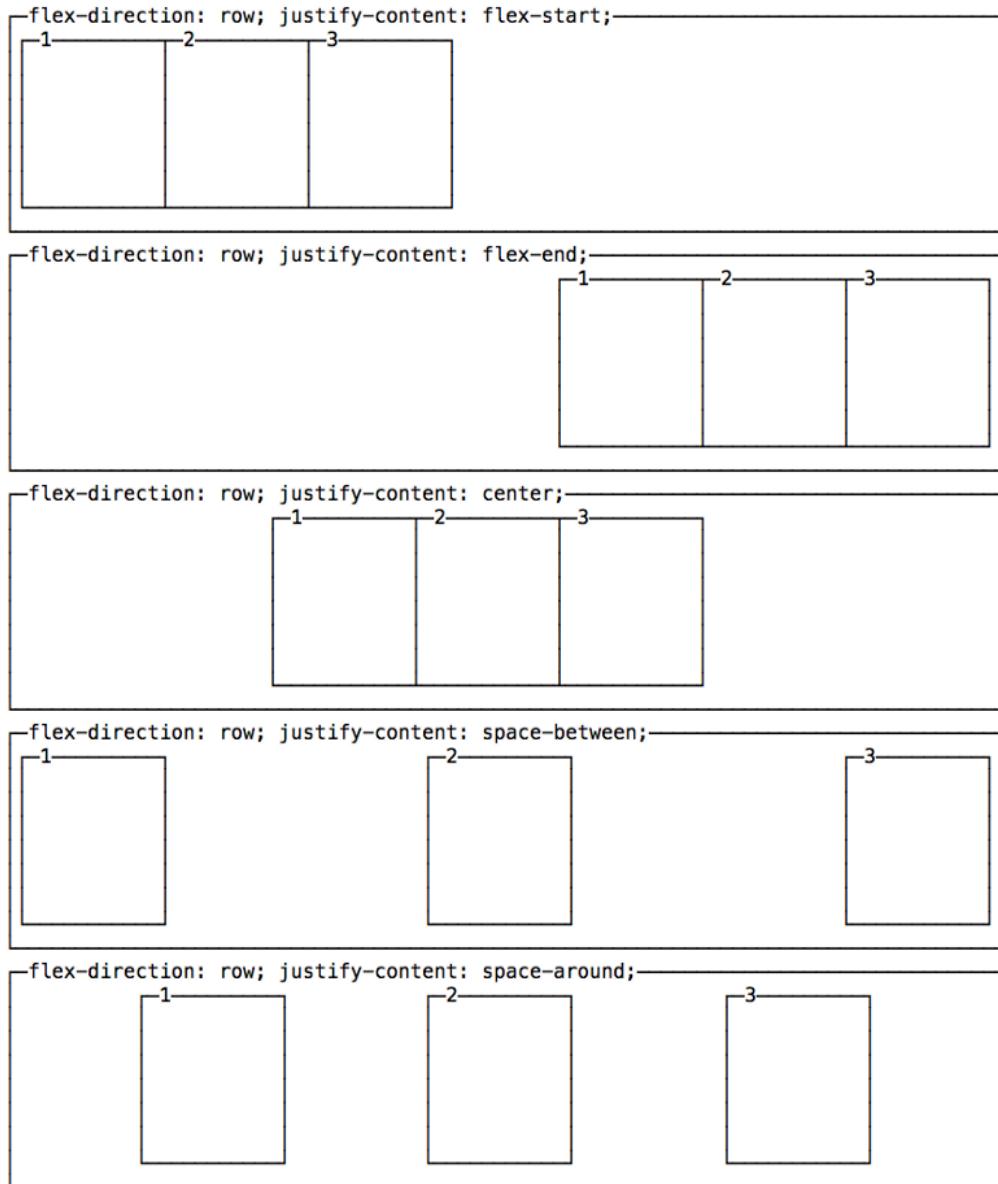
Você pode alterar esse comportamento usando o `justify-content` para alterar o alinhamento horizontal e o `align-items` para alterar o alinhamento vertical.

Como mudar o alinhamento horizontal

`justify-content` tem 5 valores possíveis:

- `flex-start`: alinhar ao lado esquerdo do contêiner.
- `flex-end`: alinhar ao lado direito do contêiner.
- `center`: alinhar ao centro do contêiner.
- `space-between`: exibir com espaçamento igual entre eles.

- `space-around` : exibir com espaçamento igual ao redor deles.

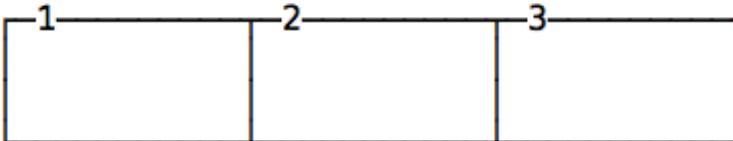


Alterar o alinhamento vertical

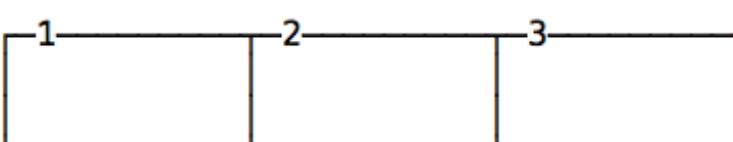
`align-items` tem 5 valores possíveis:

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

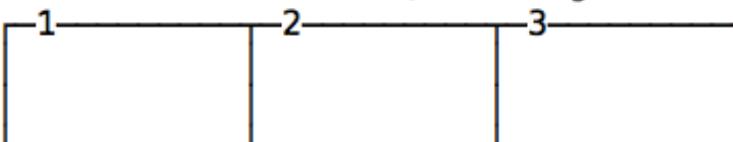
```
flex-direction: row;— align-items: flex-start;
```



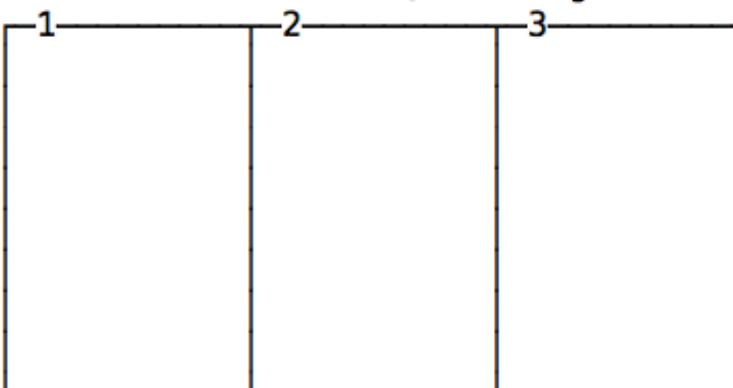
```
flex-direction: row;— align-items: flex-end;
```



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



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



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



Uma nota sobre `baseline`:

Neste exemplo, `baseline` é semelhante ao `flex-start`. Isso ocorre devido ao fato de minhas caixas serem muito simples. Dê uma olhada [neste Codepen](#) para ter um exemplo mais útil. Eu o extraí de uma Pen originalmente criada por [Martin Michálek](#). Como você pode ver, as dimensões do item estão alinhadas.

Wrap

Por padrão, os itens em um contêiner Flexbox são mantidos em uma única linha, encolhendo-os para caber no contêiner.

Para forçar os itens a se espalharem por várias linhas, use `flex-wrap: wrap`. Isso distribuirá os itens de acordo com a ordem definida na `flex-direction`. Use `flex-wrap: wrap-reverse` para inverter essa ordem.

Uma propriedade abreviada chamada `flex-flow` permite que você especifique `flex-direction` e `flex-wrap` em uma única linha, adicionando primeiro o valor `flex-direction`, seguido pelo valor de `flex-wrap`, por exemplo: `flex-flow: row wrap`.

Propriedades que se aplicam a cada item

Até este ponto, vimos as propriedades que você pode aplicar ao contêiner.

Itens individuais podem ter certa independência e flexibilidade. Você pode alterar sua aparência usando estas propriedades:

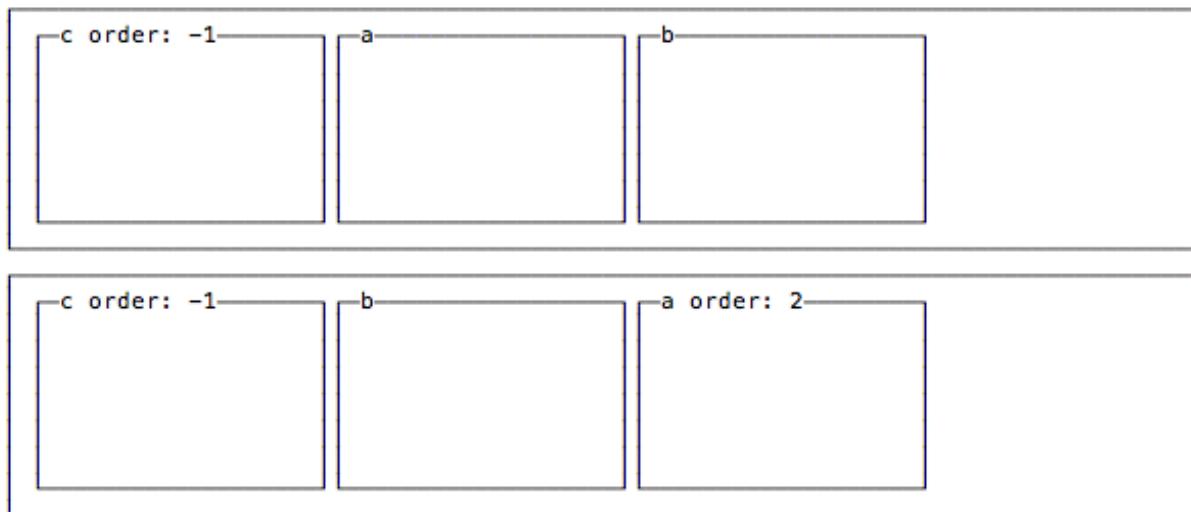
- `order`
- `align-self`
- `flex-grow`
- `flex-shrink`
- `flex-basis`
- `flex`

Vamos vê-los em detalhes.

Mover itens para antes/depois de outro usando `order`

Os itens são ordenados com base na ordem a que foram atribuídos. Por padrão, cada item tem ordem `0` e a aparência no HTML determina a ordem final.

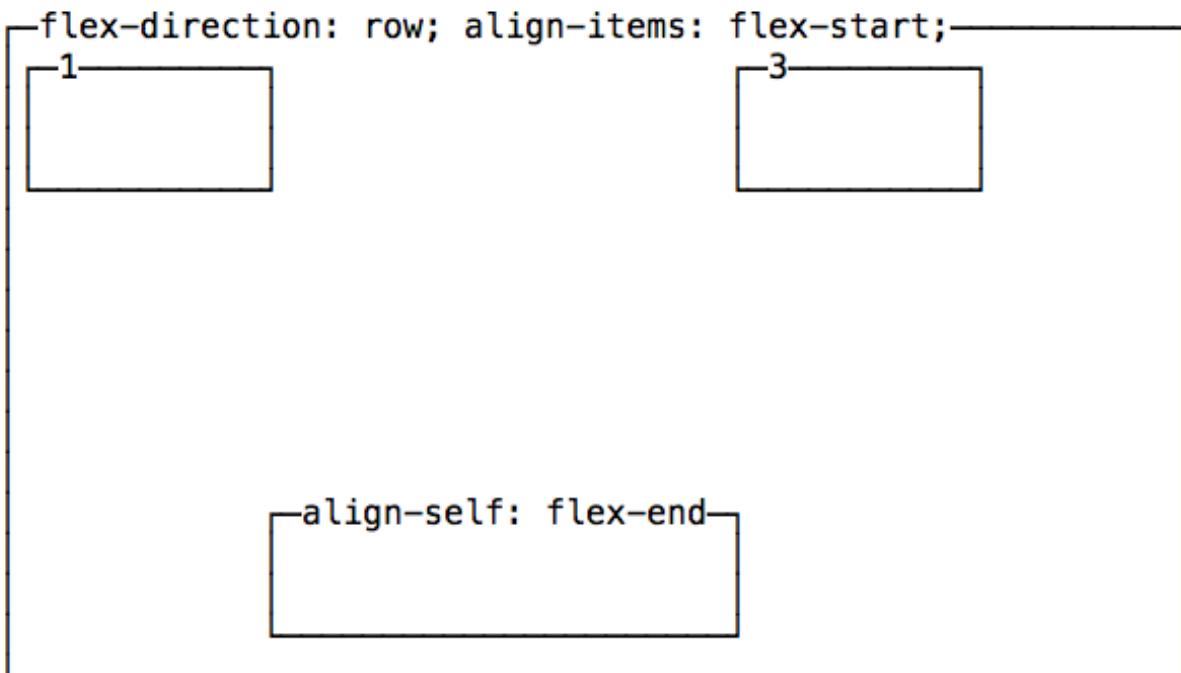
Você pode substituir essa propriedade usando `order` em cada item separado. Esta é uma propriedade que você define no item, não no contêiner. Você pode fazer um item aparecer antes de todos os outros definindo um valor negativo.



Alinhamento vertical usando `align-self`

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

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



Aumentar ou diminuir um item, se necessário

flex-grow

O padrão 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".

flex-shrink

O padrão para qualquer item é `1`.

Se todos os itens forem definidos como 1 e um for definido como 3, o elemento maior encolherá 3 vezes os outros. Quando menos espaço estiver disponível, ocupará 3 vezes menos espaço.

flex-basis

Se definido como `auto`, ele dimensiona um item de acordo com sua largura ou altura e adiciona espaço extra com base na propriedade `flex-grow`.

Se definido como `0`, não adiciona 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 é uma linha ou um item de coluna)

flex

Esta propriedade combina as 3 propriedades acima:

- `flex-grow`

- `flex-shrink`
- `flex-basis`

e fornece uma sintaxe abreviada: `flex: 0 1 auto`

TABELAS

No passado, as tabelas eram muito usadas em CSS, pois eram uma das únicas maneiras de se criar um layout de página sofisticado.

Hoje, com o Grid e o Flexbox, podemos mover as tabelas de volta para o trabalho que deveriam fazer: estilizar tabelas.

Vamos começar pelo 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 tem muita graça. O navegador fornece alguns estilos padrão e só:

Name	Age
Flavio	36
Roger	7
Syd	6

Podemos usar CSS para estilizar todos os elementos da tabela, é claro.

Começemos pela borda. Uma borda bonita pode obter bons resultados.

Podemos aplicá-la no elemento `table` ou nos elementos internos, como `th` e `td`:

```
table, th, td {  
    border: 1px solid #333;  
}
```

Se emparelhamos com alguma margem, obteremos um bom resultado:

Name	Age
Flavio	36
Roger	7
Syd	6

Uma coisa comum com as tabelas é a capacidade de adicionar uma cor a uma linha e uma cor diferente a outra linha. Isso é possível usando o seletor `:nth-child(odd)` ou `:nth-child(even)`:

```
tbody tr:nth-child(odd) {  
    background-color: #af47ff;  
}
```

Isso resulta em:

Name	Age
Flavio	36
Roger	7
Syd	6

Se você adicionar `border-collapse: collapse;` ao elemento `table`, todas as bordas são mescladas em uma:

Name	Age
Flavio	36
Roger	7
Syd	6

CENTRALIZANDO

Centralizar coisas em CSS é uma tarefa muito diferente se você precisar centralizar horizontalmente ou verticalmente.

Neste artigo, explico os cenários mais comuns e como resolvê-los. Se uma nova solução for fornecida pelo [Flexbox](#), eu ignoro as técnicas antigas porque precisamos seguir em frente. O Flexbox é suportado por navegadores há anos, incluindo pelo IE10.

Centralizar horizontalmente

Texto

O texto é muito simples de centralizar horizontalmente usando a propriedade `text-align` definida para `center`:

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

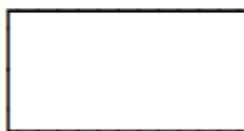
Blocos

A maneira moderna de centralizar qualquer coisa que não seja texto é usar o Flexbox:

```
#minha-section {  
  display: flex;  
  justify-content: center;  
}
```

qualquer elemento dentro de `#minha-section` será centralizado horizontalmente.

`justify-content: center;`



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%;  
}
```

`margin: 0 auto;` é uma abreviação para:

```
section {  
    margin-top: 0;  
    margin-bottom: 0;  
    margin-left: auto;  
    margin-right: auto;  
}
```

Lembre-se de configurar o item para `display: block` se for um elemento *inline*.

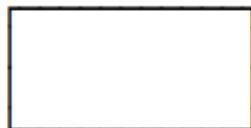
Centralizar verticalmente

Tradicionalmente, esta sempre foi uma tarefa difícil. O Flexbox agora nos oferece uma ótima maneira muito simples de se fazer isso:

```
#minha-section {  
    display: flex;  
    align-items: center;  
}
```

Qualquer elemento dentro de `#minha-section` será centrado verticalmente.

`align-items: center;`

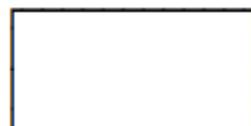


Centralize na vertical e na horizontal

Técnicas de Flexbox para centralizar vertical e horizontalmente podem ser combinadas para centralizar completamente um elemento na página.

```
#minha-section {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
}
```

align-items: center; justify-content: center;



A mesma coisa pode ser feita usando [CSS Grid](#):

```
body {  
    display: grid;  
    place-items: center;  
    height: 100vh;  
}
```

LISTAS

As listas são uma parte muito importante de muitas páginas da web.

O CSS pode estilizá-las 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> (em inglês) com exemplos de sua aparência. Alguns dos mais populares são `disc`, `circle`, `square` e `none`.

`list-style-image` é usado para usar um marcador personalizado quando um marcador predefinido não é apropriado:

```
li {  
    list-style-image: url(imagem-da-lista.png);  
}
```

`list-style-position` permite adicionar o marcador `outside` (padrão) ou `inside` do conteúdo da lista, no fluxo da página e não fora dela

```
li {  
    list-style-position: inside;  
}
```

A propriedade abreviada de `list-style` nos permite especificar todas essas propriedades na mesma linha:

```
li {  
    list-style: url(imagem-da-lista.png) inside;  
}
```

MEDIA QUERIES E DESIGN RESPONSIVO

Nesta seção, vamos primeiro apresentar os tipos de mídia e os descritores de recursos de mídia. Em seguida, explicaremos as consultas de mídia.

Tipos de mídia

Usados em consultas de mídia (em inglês, *media queries*) e declarações `@import`, os tipos de mídia nos permitem determinar em qual mídia um arquivo CSS, ou um trecho de CSS, é carregado.

Temos os seguintes tipos de mídia

- `all` significa toda as mídias
- `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 está obsoleta, pois provou ser uma maneira ineficaz de determinar as necessidades do dispositivo.

Podemos usá-los em declarações `@import` como esta:

```
@import url(meu- arquivo.css) screen;  
@import url(meu- arquivo- impressao.css) print;
```

Podemos carregar um arquivo CSS em vários tipos de mídia, separando cada um com uma vírgula:

```
@import url(meu- arquivo.css) screen, print;
```

A mesma coisa funciona para a tag `link` em HTML:

```
<link rel="stylesheet" type="text/css" href="meu- arquivo.css" media="screen" />  
<link rel="stylesheet" type="text/css" href="outro- arquivo.css" media="screen, print" />
```

Não estamos limitados a apenas usar tipos de mídia no atributo `media` e na declaração `@import`. Há outros.

Descritores de recursos de mídia

Primeiro, vamos apresentar os descritores de recursos de mídia. São palavras-chave adicionais que podemos adicionar ao atributo `link` da mídia ou à declaração `@import`, para expressar mais condicionais sobre o carregamento do CSS.

Aqui está a lista:

- `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 `width`, `height`, `device-width` e `device-height`.

Por exemplo:

```
@import url(meu-arquivo.css) screen and (max-width: 800px);
```

Observe que envolvemos cada bloco usando descritores de recursos de mídia entre parênteses.

Alguns aceitam um valor fixo. `orientation`, usado para detectar a orientação do dispositivo, aceita `portrait` ou `landscape`.

Exemplo:

```
<link rel="stylesheet" type="text/css" href="meu-arquivo.css" media="screen and (orientation: portrait)" />
```

`scan`, usado para determinar o tipo de tela, aceita `progressive` (para monitores modernos) ou `interlace` (para dispositivos CRT mais antigos).

Alguns outros requerem um número inteiro.

Também temos `color`, que inspeciona o número de bits por componente de cor usado pelo dispositivo. É de um nível muito baixo, mas você só precisa saber que está lá para seu uso (como `grid`, `color-index`, `monochrome`).

`aspect-ratio` e `device-aspect-ratio` aceitam um valor de proporção que representa a proporção entre largura e altura da janela de visualização, que é expressa como uma fração.

Exemplo:

```
@import url(meu-arquivo.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(meu-arquivo.css) screen and (min-resolution: 100dpi);
```

Operadores lógicos

Podemos combinar regras usando `and`:

```
<link rel="stylesheet" type="text/css" href="meu-arquivo.css" media="screen and (max-width: 800px)" />
```

Podemos realizar uma operação lógica do tipo "`or`" usando vírgulas, que combina várias *media queries*:

```
@import url(meu-arquivo.css) screen, print;
```

Podemos usar `not` para negar uma *media query*:

```
@import url(meu-arquivo.css) not screen;
```

Importante: `not` só pode ser usado para negar uma consulta de mídia inteira. Portanto, deve ser colocado no início dela (ou após uma vírgula).

Media queries

Todas as regras acima que vimos aplicadas a `@import` ou à tag `link` do HTML também podem ser aplicadas dentro do CSS.

Você precisa envolvê-los em uma estrutura `@media () {}`.

Exemplo:

```
@media screen and (max-width: 800px) {  
    /* seu CSS vai aqui */  
}
```

e esta é a base para o design responsivo.

As *media queries* podem ser bastante complexas. Este exemplo aplica o CSS somente se for um dispositivo de tela, se a largura estiver entre 600 e 800 pixels e se a orientação for paisagem:

```
@media screen and (max-width: 800px) and (min-width: 600px) and (orientation: landscape) {  
    /* seu CSS vai aqui */  
}
```

FEATURE QUERIES

Feature Queries são uma habilidade recente e relativamente desconhecida do CSS, mas com bom suporte.

Podemos usá-la para verificar se um recurso é suportado pelo navegador usando a palavra-chave `@supports`.

Acho que isso é especialmente útil, no momento em que escrevo, para verificar se um navegador suporta CSS Grid, por exemplo, o que pode ser feito usando:

```
@supports (display: grid) {  
    /* aplique este CSS */  
}
```

Verificamos se o navegador suporta o valor `grid` para a propriedade `display`.

Podemos usar `@supports` para qualquer propriedade do CSS, para verificar qualquer valor.

Também podemos usar os operadores lógicos `and`, `or` e `not` para construir *feature queries* complexas:

```
@supports (display: grid) and (display: flex) {  
    /* aplique este CSS */  
}
```

FILTROS

Filtros nos permitem realizar operações nos elementos.

Coisas que você normalmente faz com o Photoshop ou outro software de edição de fotos, como alterar a opacidade ou o brilho e muito mais, sobre os elementos.

Você usa a propriedade `filter`. Aqui está um exemplo dela aplicada em uma imagem, mas esta propriedade pode ser usada em *qualquer* elemento:

```
img {  
    filter: <algum-filtro>;  
}
```

Você pode usar vários valores aqui:

- `blur()`
- `brightness()`
- `contrast()`

- `drop-shadow()`
- `grayscale()`
- `hue-rotate()`
- `invert()`
- `opacity()`
- `sepia()`
- `saturate()`
- `url()`

Observe os parênteses após cada opção, pois todas requerem um parâmetro.

Por exemplo:

```
img {
  filter: opacity(0.5);
}
```

significa que a imagem será 50% transparente, porque `opacity()` assume um valor de 0 a 1, ou uma porcentagem.

Você também pode aplicar vários filtros de uma só vez:

```
img {
  filter: opacity(0.5) blur(2px);
}
```

Vamos agora falar sobre cada filtro em detalhes.

`blur()`

Desfoca o conteúdo de um elemento. Você passa um valor, expresso em `px`, `em` ou `rem`, que será usado para determinar o raio do desfoco.

Exemplo:

```
img {
  filter: blur(4px);
}
```

`opacity()`

`opacity()` assume um valor de 0 a 1, ou uma porcentagem, e determina a transparência da imagem com base nele.

0, ou 0%, significa totalmente transparente. 1, ou 100%, ou superior, significa totalmente visível.

Exemplo:

```
img {  
  filter: opacity(0.5);  
}
```

CSS também tem uma propriedade `opacity`. No entanto, `filter` pode ser acelerado por hardware, dependendo da implementação. Portanto, esse deve ser o método preferencial.

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 da imagem. Se a imagem não tiver um canal alfa, a sombra será aplicada em toda a caixa da 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.
- `blur-radius`, opcional, define o raio de desfoque para a sombra. O padrão é 0, sem desfoque.
- `spread-radius`, opcional, define o raio de propagação. Expresso em `px`, `rem` ou `em`
- `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);  
}
```

grayscale()

Faz com que o elemento tenha uma cor cinza.

Você passa um valor de 0 a 1, ou de 0% a 100%, onde 1 e 100% significa totalmente cinza e 0 ou 0% significa que a imagem não é tocada e as cores originais permanecem.

Exemplo:

```
img {  
  filter: grayscale(50%);  
}
```

sepia()

Faz com que o elemento tenha uma cor sépia.

Você passa um valor de 0 a 1, ou de 0% a 100%, onde 1 e 100% significam totalmente sépia, e 0 ou 0% significa que a imagem não é tocada e as cores originais permanecem.

Exemplo:

```
img {  
  filter: sepia(50%);  
}
```

invert()

Inverte 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 de amarelo é azul, o oposto de 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%);  
}
```

hue-rotate()

A roda de cores HSL é representada em graus. Usando `hue-rotate()` você pode girar a cor usando uma rotação positiva ou negativa.

a função aceita um valor `deg`.

Exemplo:

```
img {  
  filter: hue-rotate(90deg);  
}
```

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 clara até atingir um elemento branco total.

Exemplo:

```
img {  
  filter: brightness(50%);  
}
```

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% fornecem mais contraste.

Exemplo:

```
img {  
  filter: contrast(150%);  
}
```

saturate()

Altera a saturação de um elemento.

0 ou 0% dá um elemento total em tons de cinza (com menos saturação). 1 ou 100% dá uma imagem inalterada.

Valores superiores a 1 ou 100% dão mais saturação.

Exemplo:

```
img {  
  filter: saturate();  
}
```

url()

Este filtro permite aplicar um filtro definido em um arquivo SVG. Você aponta para o local do arquivo SVG.

Exemplo:

```
img {  
  filter: url(filtro.svg);  
}
```

Os filtros SVG estão fora do escopo deste artigo, mas você pode ler mais sobre isso neste artigo da Smashing Magazine: <https://www.smashingmagazine.com/2015/05/why-the-svg-filter-is-awesome/>

TRANSFORMAÇÕES

Transformações permitem mover, girar, dimensionar e inclinar elementos no espaço 2D ou 3D. Elas são um recurso CSS muito legal, especialmente quando combinadas com animações.

Transformações em 2D

A propriedade `transform` aceita essas funções:

- `translate()` para mover elementos ao redor
- `rotate()` para girar elementos
- `scale()` escalar elementos em tamanho
- `skew()` 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 de eixo:

- `translateX()` para mover elementos no eixo X
- `translateY()` para mover elementos no eixo Y
- `scaleX()` dimensionar elementos em tamanho no eixo X
- `scaleY()` dimensionar elementos em tamanho no eixo Y
- `skewX()` torcer ou inclinar um elemento no eixo X
- `skewY()` torcer ou inclinar um elemento no eixo Y

Aqui está um exemplo de uma transformação que muda a largura do elemento `.box` multiplicando-a por 2 (duplicando-a) e a altura por 0,5 (reduzindo-a à metade):

```
.box {  
  transform: scale(2, 0.5);  
}
```

```
}
```

`transform-origin` permite definir a origem (as coordenadas `(0, 0)`) para a transformação, permitindo alterar o centro de rotação.

Combinando várias transformações

Você pode combinar várias transformações separando cada função com um espaço.

Por exemplo:

```
transform: rotateY(20deg) scaleX(3) translateY(100px);
```

Transformações em 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 nossos visuais.

Usando a propriedade `perspective`, você pode especificar a que distância o objeto 3D está do visualizador.

Exemplo:

```
.3Delement {  
  perspective: 100px;  
}
```

`perspective-origin` determina a aparência da posição do visualizador, como estamos olhando nos eixos X e Y.

Agora, podemos usar funções adicionais que controlam o eixo Z e que se somam às outras transformações dos eixos X e Y:

- `translateZ()`
- `rotateZ()`
- `scaleZ()`

As abreviações correspondentes, `translate3d()`, `rotate3d()` e `scale3d()` podem ser usadas como atalhos para usar as funções `translateX()`, `translateY()` e `translateZ()` e assim por diante.

As transformações 3D são um pouco avançadas demais para este manual, mas são um ótimo tópico para explorar por conta própria.

TRANSIÇÕES

As transições do CSS são a maneira mais simples de se 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 do CSS são definidas por estas propriedades:

Property	Description
<code>transition-property</code>	the CSS property that should transition
<code>transition-duration</code>	the duration of the transition
<code>transition-timing-function</code>	the timing function used by the animation (common values: linear, ease). Default: ease
<code>transition-delay</code>	optional number of seconds to wait before starting the animation

A propriedade `transition` é uma abreviação útil:

```
.container {  
  transition: property  
            duration  
            timing-function  
            delay;  
}
```

Exemplo de transição do CSS

Este código implementa uma transição do CSS:

```
.one,  
.three {  
  background: rgba(142, 92, 205, .75);  
  transition: background 1s ease-in;  
}  
  
.two,  
.four {  
  background: rgba(236, 252, 100, .75);  
}  
  
.circle:hover {  
  background: rgba(142, 92, 205, .25); /* lighter */  
}
```

Veja o exemplo no Glitch <https://flavio-css-transitions-example.glitch.me>

Ao passar o mouse sobre os elementos `.one` e `.three`, os círculos roxos, há uma animação de transição que facilita a mudança de fundo, enquanto nos círculos amarelos não, pois não possuem a propriedade `transition` definida.

Valores da função de tempo 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`

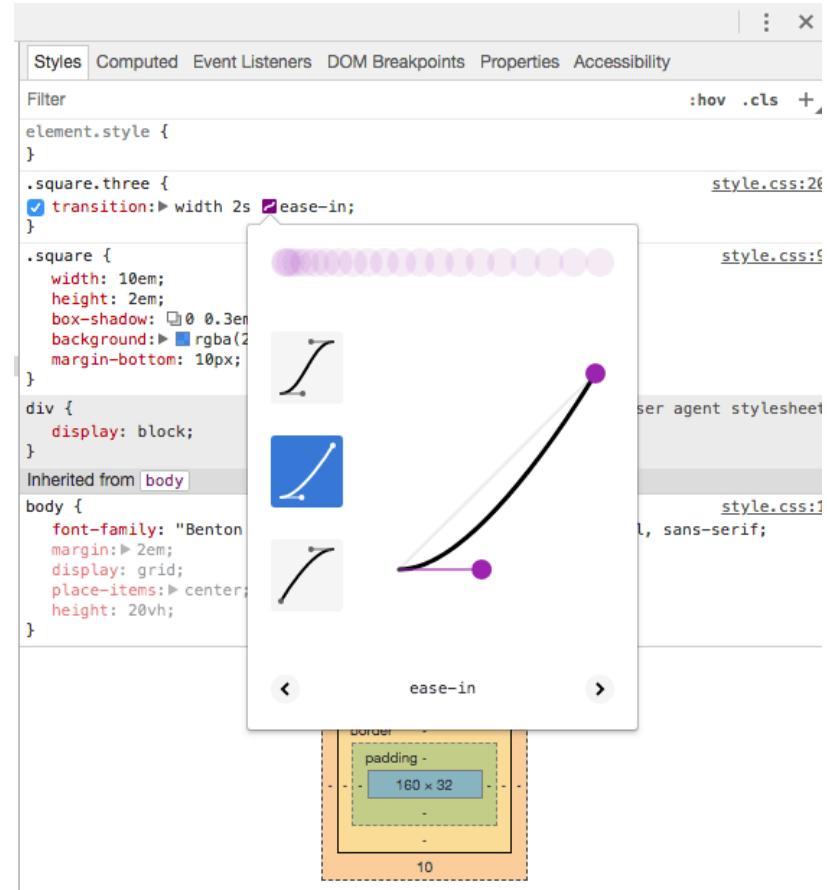
[Este Glitch](#) mostra como eles funcionam na prática.

Você pode criar uma função de temporização completamente personalizada usando [curvas cúbicas de Bezier](#). Isso é bastante avançado, mas, basicamente, qualquer uma das funções acima é construída usando curvas de Bezier. Temos nomes úteis, pois são comuns.

Transições do CSS nas ferramentas do desenvolvedor do navegador

As [ferramentas do desenvolvedor do navegador](#) oferecem uma ótima maneira de visualizar as transições.

Esta é a visão no Chrome:



Esta é a visão no Firefox:

Rules Computed Layout Animations Fonts

Filter Styles + .cls

```

element { inline }
.square.three { style.css:20
  transition: width 2s ease-in;
}
.square {
  width: 10em;
  height: 2em;
  box-shadow: 0 0.3em 0.3em 0.3em;
  background: rgba(243, 243, 243);
  margin-bottom: 10px;
}
Inherited from body
body {
  font-family: "Benton Sans Compressed", helvetica, arial, sans-serif;
  margin: 2em;
  display: grid;
  place-items: center;
  height: 20vh;
}

```

ease-in

ease-out

ease-in-out

Ease-in Ease-out Ease-in-out

Linear Ease-in Sine

Quadratic Cubic Quartic

Quintic Exponential Circular

Backward

A partir desses painéis, você pode editar a transição ao vivo e experimentar na página diretamente sem recarregar seu código.

Quais propriedades você pode animar usando transições do CSS

Muitas! Elas são as mesmas que você pode animar usando as animações do CSS.

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`

ANIMAÇÕES

As animações do CSS são uma ótima maneira de criar animações visuais, não limitadas a um único movimento, como as transições do 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 **keyframes**. Exemplo de uma animação que gira um item:

```
@keyframes spin {  
    0% {  
        transform: rotateZ(0);  
    }  
    100% {  
        transform: rotateZ(360deg);  
    }  
}
```

Dentro da definição `@keyframes` você pode ter quantos *waypoints* intermediários quiser.

Neste caso, instruímos o CSS a fazer com que a propriedade `transform` gire o eixo Z de 0 a 360 graus, completando um loop.

Você pode usar qualquer transformação do CSS aqui.

Observe como isso não dita nada sobre o intervalo temporal que a animação deve ter. Isso é definido quando você o usar via `animation`.

Um exemplo de animações do CSS

Quero desenhar quatro círculos, todos com um ponto de partida em comum, todos distantes 90 graus 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, .5);
}

.one,
.three {
  background: rgba(142, 92, 205, .75);
}

.two,
.four {
  background: rgba(236, 252, 100, .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 keyframes 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 em <https://flavio-css-animations-four-steps.glitch.me>

As propriedades de animação do CSS

As animações do CSS oferecem muitos parâmetros diferentes que você pode ajustar:

Property	Description
animation-name	the name of the animation, it references an animation created using <code>@keyframes</code>
animation-duration	how long the animation should last, in seconds
animation-timing-function	the timing function used by the animation (common values: <code>linear</code> , <code>ease</code>). Default: <code>ease</code>
animation-delay	optional number of seconds to wait before starting the animation
animation-iteration-count	how many times the animation should be performed. Expects a number, or <code>infinite</code> . Default: 1
animation-direction	the direction of the animation. Can be <code>normal</code> , <code>reverse</code> , <code>alternate</code> or <code>alternate-reverse</code> . In the last 2, it alternates going forward and then backwards
animation-fill-mode	defines how to style the element when the animation ends, after it finishes its iteration count number. <code>none</code> or <code>backwards</code> go back to the first keyframe styles. <code>forwards</code> and <code>both</code> use the style that's set in the last keyframe
animation-play-state	if set to <code>paused</code> , it pauses the animation. Default is <code>running</code>

A propriedade `animation` é um atalho para todas essas propriedades, nesta ordem:

```
.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;  
}
```

Eventos do JavaScript para as animações do CSS

Usando JavaScript, você pode "escutar" os seguintes eventos:

- `animationstart`
- `animationend`
- `animationiteration`

Cuidado com o `animationstart`, pois se a animação começar no carregamento da página, seu código JavaScript é sempre executado após o processamento do CSS. Então, a animação já foi iniciada e você não pode interceptar o evento.

```
const container = document.querySelector('.container')  
  
container.addEventListener('animationstart', (e) => {  
    //fazer algo  
}, false)  
  
container.addEventListener('animationend', (e) => {  
    //fazer algo  
}, false)  
  
container.addEventListener('animationiteration', (e) => {  
    //fazer algo  
}, false)
```

Quais propriedades você pode animar usando as animações do CSS

Muitas! Elas são as mesmas que você pode animar usando as transições do CSS.

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`

NORMALIZANDO O CSS

A folha de estilo padrão do navegador é o conjunto de regras que os navegadores devem aplicar para fornecerem um estilo mínimo aos elementos.

Na maioria das vezes, esses estilos são muito úteis.

Como cada navegador tem seu próprio conjunto, é comum encontrar uma base em comum.

Em vez de remover todos os padrões, como faz uma das abordagens de redefinição de CSS, o processo de normalização remove as inconsistências dos navegadores, 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.

LIDANDO COM ERROS

O CSS é resiliente. Quando encontra um erro, ele não age como o JavaScript, que simplesmente para de funcionar completamente, encerrando todas as execuções de script depois que o erro é encontrado.

O CSS se esforça muito para fazer o que você quer.

Se uma linha tiver um erro, ele pula e passa 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** serão aplicadas, mas a terceira regra será aplicada com sucesso na página. Basicamente, ele varre tudo até encontrar um ponto e vírgula, mas quando o encontra, a regra agora é `font-size: 20px color: black;`, que é inválido. Então, ele a pula.

Às vezes é complicado perceber que há um erro em algum lugar e onde está esse erro, porque o navegador não nos informa.

É por isso que ferramentas como o [CSS Lint](#) existem.

PREFIXOS DOS FABRICANTES

Os prefixos dos fabricantes de navegadores são uma maneira que os navegadores usam para fornecer aos desenvolvedores de CSS acesso a recursos mais recentes que ainda não são considerados estáveis.

Antes de continuar, lembre-se de que essa abordagem está perdendo popularidade. As pessoas agora preferem usar sinalizadores experimentais, que devem ser ativados explicitamente no navegador do usuário.

Por quê? Porque os desenvolvedores, em vez de considerar os prefixos dos navegadores como uma forma de visualizar os recursos, às vezes os enviam para a produção – algo considerado prejudicial pelo CSS Working Group.

Em grande parte, isso se dá pelo fato de que, ao adicionar uma *flag* e os desenvolvedores começarem a utilizá-la em produção, os navegadores ficam em uma situação complicada se eles percebem que algo

deve mudar. Com as *flags*, não é possível enviar um recurso a menos que você convença todos os visitantes a ativarem aquela flag no navegador deles (é brincadeira – nem pense em fazer isso).

Dito isso, vamos ver quais são os prefixos dos fabricantes.

Eu me lembro especificamente deles por trabalhar com transições do CSS no passado. Em vez de apenas usar a propriedade `transition`, você tinha que fazer isso:

```
.minhaClasse {  
    -webkit-transition: all 1s linear;  
    -moz-transition: all 1s linear;  
    -ms-transition: all 1s linear;  
    -o-transition: all 1s linear;  
    transition: all 1s linear;  
}
```

Agora é só usar

```
.minhaClasse {  
    transition: all 1s linear;  
}
```

A propriedade, agora, é bem suportada por todos os navegadores modernos.

Os prefixos usados são:

- `-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 também, `-o-` e `-ms-` provavelmente sairão de moda em breve. Porém, como dissemos, os prefixos de fornecedores como um todo também estão saindo de moda.

Escrever com prefixos é difícil, principalmente por causa da incerteza. Você realmente precisa de um prefixo para uma propriedade? Vários recursos on-line também estão desatualizados, o que torna ainda mais difícil fazer o que é 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 de 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 `create-react-app` e Vue CLI, duas maneiras comuns de começar a construir uma aplicação, vêm com `autoprefixer` pronto para uso, para que você nem precise se preocupar com isso.

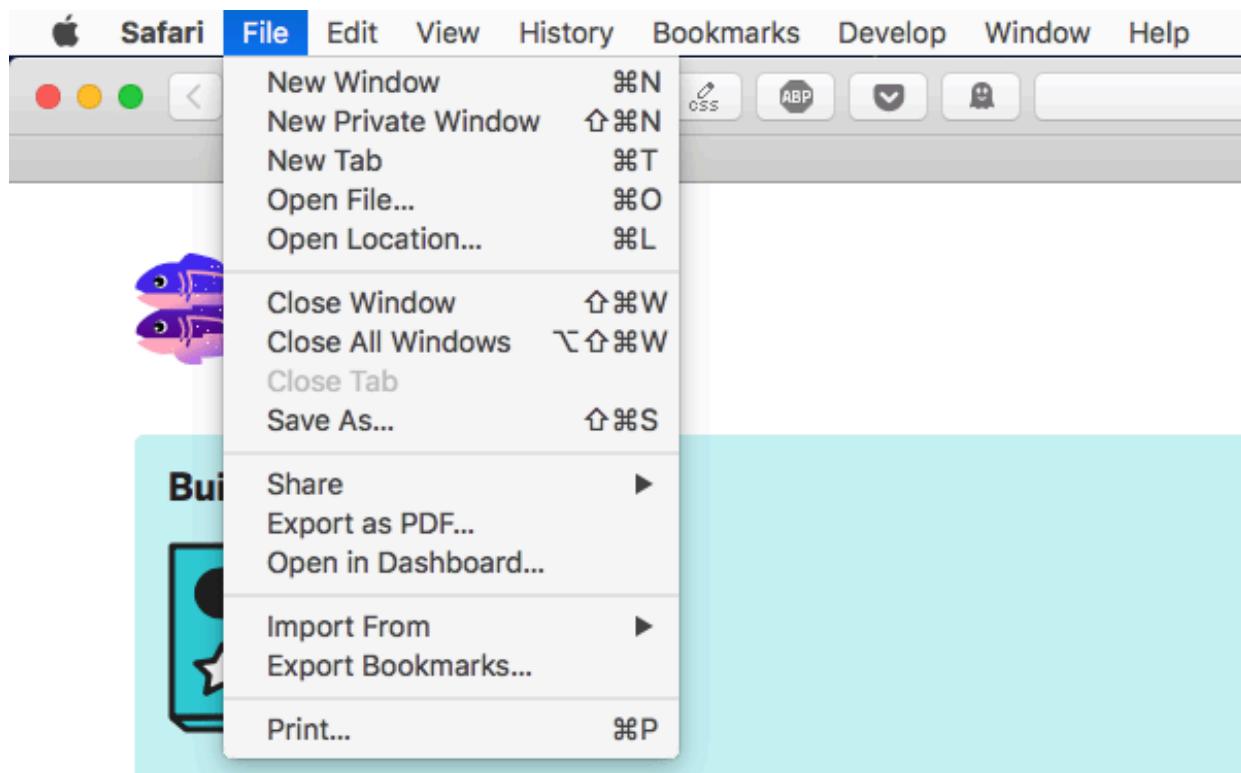
CSS PARA IMPRESSÃO

Embora olhemos cada vez mais para nossas telas, a impressão ainda existe.

Mesmo com postagens de blog. Lembro-me de uma vez em 2009 em que conheci uma pessoa que me disse que fazia seu assistente pessoal imprimir todos os posts de blog que eu publicava (sim, fiquei um pouco sem entender). Foi definitivamente inesperado.

Meu caso de uso principal para procurar 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 padrão do Chrome em "Salvar" ao tentar imprimir um documento e uma impressora não está disponível, e o Safari, que tem um botão dedicado na barra de menus:



Print CSS

Uma coisa comum que você pode querer fazer ao imprimir é ocultar algumas partes do documento, talvez o rodapé, algo no cabeçalho, a barra lateral.

Talvez você queira usar uma fonte diferente para imprimir, o que é um motivo válido.

Se você tiver um CSS grande para impressão, é melhor usar um arquivo separado para ele. Os navegadores só vão baixá-lo ao imprimir:

```
<link rel="stylesheet"
      src="print.css"
      type="text/css"
      media="print" />
```

CSS @media print

Uma alternativa à abordagem anterior são as *media queries*. Qualquer coisa que você adicionar dentro deste bloco:

```
@media print {  
    /* ... */  
}
```

será aplicada apenas a documentos impressos.

Links

O HTML é ótimo por causa dos links. Chama-se de hipertexto 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 `<a>`, usando:

```
@media print {  
    a[href*='//']:after {  
        content: " (" attr(href) ") ";  
        color: $primary;  
    }  
}
```

Eu direciono `a[href*='//']` para fazer isso apenas para links externos. Posso ter links internos para fins de navegação e indexação interna, o que seria inútil na maioria dos meus casos de uso. Se você também deseja que os links internos sejam impressos, basta fazer:

```
@media print {  
    a:after {  
        content: " (" attr(href) ") ";  
        color: $primary;  
    }  
}
```

Margem das páginas

Você pode adicionar margens a cada página. `cm` ou `in` são boas unidades para impressão em papel.

```
@page {  
    margin-top: 2cm;  
    margin-bottom: 2cm;  
    margin-left: 2cm;  
    margin-right: 2cm;  
}
```

`@page` também pode ser usado para segmentar apenas a primeira página, usando `@page :first`, ou apenas as páginas da esquerda e da direita, usando `@page :left` e `@page: right`.

Quebra de página

Você pode querer adicionar uma quebra de página após alguns elementos ou antes deles. Usar `page-break-after` e `page-break-before`:

```
.book-date {  
    page-break-after: always;  
}  
  
.post-content {  
    page-break-before: always;  
}
```

Essas propriedades aceitam uma ampla variedade de valores (texto em inglês).

Evite quebrar imagens ao meio

Isso aconteceu comigo com o Firefox: as imagens por padrão são cortadas no meio e continuam na próxima página. Também pode acontecer com o texto.

Use

```
p {  
    page-break-inside: avoid;  
}
```

e envolva suas imagens em uma tag `p`. Direcionar `img` diretamente não funcionou em meus testes.

Isso também se aplica a outros conteúdos, não apenas imagens. Se você perceber que algo foi cortado quando você não deseja, use esta propriedade.

Debug da apresentação de impressão

As ferramentas do desenvolvedor do Chrome oferecem maneiras de emular o layout de impressão:

The screenshot shows the Chrome DevTools interface. The top navigation bar includes Application, Security, Audits, and a tab count (1). On the right, there are icons for Dock side, Hide console drawer (Esc), Search all files (⌘ F), Open file (⌘ P), More tools (highlighted in blue), Shortcuts (F1), and Help.

The main area displays the Styles tab of the Elements panel. It shows the CSS rules for the selected element, including:

```

element.style {
}
.post-content:last-child {
    margin-bottom: 0;
}
div {
    display: block;
}
Inherited from body
body {
    margin: 0;
    background: #fefefe;
    color: #424242;
    font-family: -apple-system,
        "Oxygen", "Ubuntu",
        "Emojи", "Segoe UI Emo
    font-size: 20px;
}
Inherited from html
html {
    line-height: 1.15;
    -ms-text-size-adjust: 100%;
    -webkit-text-size-adjust: 100%;
}

```

A context menu is open over the 'body' style rule, with 'Rendering' highlighted in blue. Other options in the menu include Animations, Changes, Coverage, JavaScript Profiler, Layers, Network conditions, Performance monitor, Quick source, Remote devices, Request blocking, Search, Sensors, and What's New.

Quando o painel abrir, altere a emulação de renderização para `print`:

The screenshot shows the Chrome DevTools interface with the Rendering tab selected. At the top, there are tabs for Console, Rendering (highlighted in blue), What's New, and Coverage.

The main area contains several checkboxes and a dropdown menu:

- 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". A dropdown menu shows three options: "No emulation" (checked), "print" (highlighted in blue), and "screen".

CONCLUSÕES

Espero que este artigo tenha ajudado você a se familiarizar com o CSS e a obter uma visão geral dos principais recursos que você pode usar para estilizar suas páginas e aplicações. Escrevi o artigo para ajudá-lo a se familiarizar rapidamente com o CSS e com o uso dessa ferramenta incrível que permite criar designs impressionantes na web. Espero ter alcançado esse objetivo.

Clique aqui para obter uma [versão em inglês em PDF / ePub / Mobi deste post](#) para ler off-line