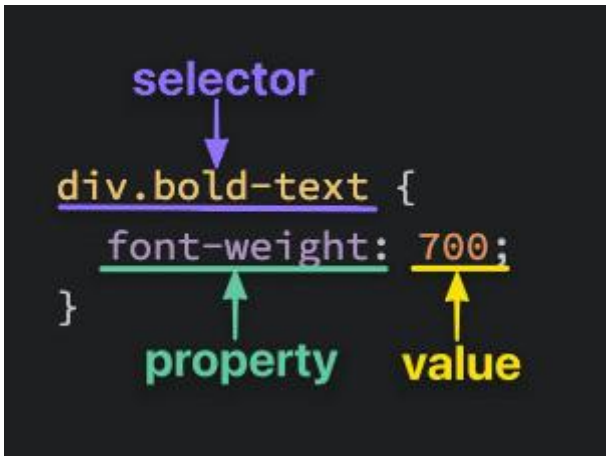
Fundamentos CSS

Introdução

Na lição anterior, você aprendeu a escrever o HTML que determina como uma página da Web é estruturada. O próximo passo é fazer com que essa estrutura pareça boa com algum estilo , que é exatamente para que serve o CSS. Nesta lição, vamos nos concentrar no que acreditamos serem alguns conceitos básicos de CSS, coisas que todos deveriam saber desde o início — estejam eles apenas começando ou simplesmente precisem de uma atualização.

Sintaxe Básica

No nível mais básico, CSS é composto de várias regras. Essas regras são compostas por um seletor (mais sobre isso daqui a pouco) e uma lista de declarações separadas por ponto e vírgula, com cada uma dessas declarações sendo composta por um par *propriedade: valor*.



Observação

A `<div>` é um dos elementos básicos do HTML. É simplesmente um recipiente (conteiner) vazio. Em geral, é melhor usar outras tags como `<h1>` ou `<p>` para conteúdo em seus projetos, mas à medida que aprendemos mais sobre CSS, você descobrirá que há muitos casos em que o que você precisa é apenas um contêiner para outros elementos. Muitos de nossos exercícios usam `<div>`s simples para simplificar. As lições posteriores serão muito mais detalhadas sobre quando é apropriado usar os vários elementos HTML.

Seletores

Os seletores simplesmente se referem aos elementos HTML aos quais as regras CSS se aplicam; eles são o que está realmente sendo "selecionado" para cada regra. As subseções a seguir não cobrem todos os seletores disponíveis, mas são de longe os mais comuns e os que você deve usar primeiro.

Seletor Universal

O seletor universal selecionará elementos de qualquer tipo, daí o nome "universal", e a sintaxe para ele é um simples asterisco. No exemplo abaixo, cada elemento teria o `color: purple;` estilo aplicado a ele.

```
* {
    color: purple;
  }
```

Seletores de tipo

Um seletor de tipo (ou seletor de elemento) selecionará todos os elementos do tipo de elemento fornecido, e a sintaxe é apenas o nome do elemento:

```
<!-- index.html -->

<div>Alô, Mundo!</div>

<div>Alô Mundo de novo!</div>

<p>Ola...</p>

<div>Okay, tchau.</div>
```

```
/* styles.css */

div {

    color: white;

  }
```

Aqui, todos os três elementos `<div>` seriam selecionados, enquanto o elemento `<p>` não seria.

Seletores de classe

Os seletores de classe selecionarão todos os elementos com a classe fornecida, que é apenas um atributo que você coloca em um elemento HTML. Veja como você adiciona uma classe a uma tag HTML e a seleciona em CSS:

```
<!-- index.html -->

<div class="alert-text">

  Por favor, concorde com nossos termos de serviço.

</div>
```

```
/* styles.css */

.alert-text {

    color: red;

  }
```

Observe a sintaxe para seletores de classe: um ponto imediatamente seguido pelo valor que diferencia maiúsculas de minúsculas do atributo de classe. As classes ***não precisam ser exclusivas***, portanto, você pode usar a mesma classe em quantos elementos desejar.

Outra coisa que você pode fazer com o atributo *class* é adicionar várias classes a um único elemento como uma lista separada por espaços, como `class="alert-text severe-alert"`. Como o espaço em branco é usado para separar nomes de classe como este, você nunca deve usar espaços para nomes com várias palavras e deve usar um hífen.

Seletores de ID

Os seletores de ID são semelhantes aos seletores de classe. Eles selecionam um elemento com o ID fornecido, que é outro atributo que você coloca em um elemento HTML:

```
<!-- index.html -->

<div id="titulo">Minha Maravilhosa Página 90's</div>
```

```
/* styles.css */

#titulo {

    background-color: red;

  }
```

Em vez de um ponto, usamos uma hashtag imediatamente seguida pelo valor que diferencia maiúsculas de minúsculas do atributo ID. Uma armadilha comum é o uso excessivo do atributo ID quando não necessariamente precisam e quando as classes serão suficientes. Embora haja casos em que o uso de um ID faça sentido ou seja necessário, como aproveitar a especificidade ou ter links

redirecionados para uma seção na página atual, você deve usar IDs com moderação (se for o caso).

A principal diferença entre classes e IDs é que **um elemento só pode ter um ID**. Um ID não pode ser repetido em uma única página e o atributo ID não deve conter nenhum espaço em branco.

Seletor de Agrupamento

E se tivermos dois grupos de elementos que compartilham algumas de suas declarações de estilo?

```
.read {

    color: white;

    background-color: black;

    /* várias declarações exclusivas */

  }



  .unread {

    color: white;

    background-color: black;

    /* várias declarações exclusivas */

  }
```

Tanto o nosso `.read` quanto os seletores `.unread` compartilham as declarações `color: white;` e `background-color: black;`, mas, por outro lado, têm várias de suas próprias declarações exclusivas. Para reduzir a repetição, podemos agrupar esses dois seletores como uma lista separada por vírgulas:

```
.read,

.unread {

  color: white;

  background-color: black;

}



.read {

  /* várias declarações exclusivas */

}



.unread {

  /* várias declarações exclusivas */
```

```
}
```

Ambos os exemplos acima (com e sem agrupamento) terão o mesmo resultado, mas o segundo exemplo reduz a repetição de declarações e facilita a edição de ambas as classes `color` ou `background-color` de uma só vez.

Seletores de encadeamento

Outra maneira de usar seletores é encadeá-los como uma lista sem qualquer separação. Digamos que tivéssemos o seguinte HTML:

```
<div>

  <div class="subsection header">Últimas Postagens</div>

  <p class="subsection preview">É aqui que uma visualização de uma postagem pode ir.</p>

</div>
```

Temos dois elementos com a classe `subsection` que possuem algum tipo de estilo único, mas e se quisermos aplicar apenas uma regra separada ao elemento que também possui `header` como uma segunda classe? Bem, poderíamos encadear ambos os seletores de classe em nosso CSS assim:

```
.subsection.header {

    color: red;
```

```
  }
```

O que `.subsection.header` faz é selecionar qualquer elemento que tenha as classes `subsection` *e* `.header` Observe como não há espaço entre os seletores de classe `.subsection` e `.header` .Essa sintaxe funciona basicamente para encadear qualquer combinação de seletores, exceto para encadear mais de um seletor de tipo .

Isso também pode ser usado para encadear uma classe e um ID, conforme mostrado abaixo:

```
<div>

  <div class="subsection header">Últimas Postagens</div>

  <p class="subsection" id="preview">É aqui que uma
visualização de uma postagem pode ir.</p>

</div>
```

Você pode pegar os dois elementos acima e combiná-los com o seguinte:

```
.subsection.header {

    color: red;

  }
```

```
.subsection#preview {

  color: blue;

}
```

Em geral, você não pode encadear mais de um seletor de tipo, pois um elemento não pode ter dois tipos diferentes ao mesmo tempo. Por exemplo, encadear dois seletores de tipo like `div` e `p` nos daria o selector `divp`, o que não funcionaria já que o seletor tentaria encontrar um elemento `<divp>` literal, que não existe.

Combinador Descendente

Combinadores nos permitem combinar vários seletores de maneira diferente de agrupá-los ou encadeá-los, pois mostram uma relação entre os seletores. Existem quatro tipos de combinadores no total, mas por enquanto vamos mostrar apenas o combinador descendente , que é representado em CSS por *um único espaço* entre os seletores. Um combinador descendente só fará com que os elementos que correspondem ao último seletor sejam selecionados se eles também tiverem um ancestral (pai, avô, etc) que corresponda ao seletor anterior.

Então, algo como `.ancestor .child` selecionaria um elemento com a classe `child` se ele tivesse um ancestral com a classe `ancestor`. Outra maneira de pensar nisso é que `child` só será selecionado se estiver aninhado dentro de `ancestor`, não importa quão profundo seja. Dê uma olhada rápida no exemplo abaixo e veja se você pode dizer quais elementos seriam selecionados com base na regra CSS fornecida:

```html
<!-- index.html -->



<div class="ancestor"> <!-- A -->

  <div class="contents"> <!-- B -->

    <div class="contents"> <!-- C -->

    </div>

  </div>

</div>

<div class="contents"></div> <!-- D -->
```

```css
/* styles.css */



.ancestor .contents {

    /* some declarations */

  }
```

No exemplo acima, os dois primeiros elementos com a classe `contents` (B e C) seriam selecionados, mas o último elemento (D) não. Seu palpite estava correto?

Realmente não há limite para quantos combinadores você pode adicionar a uma regra, então `.one .two .three .four` seria totalmente válido. Isso apenas selecionaria um elemento que tem uma classe de `four` se ele tiver um ancestral com uma classe de `three`, e se esse ancestral tiver seu próprio ancestral com uma classe de `two`, e assim por diante. Você geralmente quer evitar tentar selecionar elementos que precisam desse nível de aninhamento, pois pode ficar muito confuso e longo, e pode causar problemas quando se trata de especificidade.

Propriedades para começar

Existem algumas propriedades CSS que você usará o tempo todo, ou pelo menos com mais frequência. Vamos apresentá-lo a várias dessas propriedades, embora esta não seja uma lista completa. Aprender as propriedades a seguir será suficiente para ajudá-lo a começar.

Cor e cor de fundo

A propriedade `color` define a cor do texto de um elemento, enquanto `background-color` define, bem, a cor de fundo de um elemento. Acho que terminamos aqui?

Quase. Ambas as propriedades podem aceitar um dos vários tipos de valores. Um comum é uma palavra-chave, como um nome de cor real como `red` ou a palavra-chave `transparent`. Eles também aceitam valores HEX, RGB e HSL, com os quais você pode estar familiarizado se já usou um programa de photoshop ou um site onde pode personalizar as cores do seu perfil.

```
p {

    /* hex example: */
```

```
    color: #1100ff;

    /* rgb example: */

    color: rgb(100, 0, 127);

    /* hsl example: */

    color: hsl(15, 82%, 56%);

}
```

Dê uma olhada rápida em CSS Legal Color Values para ver como você pode ajustar a opacidade dessas cores adicionando um valor alfa.

Noções básicas de tipografia e alinhamento de texto

- `font-family` pode ser um valor único ou uma lista de valores separados por vírgulas que determinam qual fonte um elemento usa. Cada fonte se enquadra em uma das duas categorias, um "nome de família de fonte" como `"Times New Roman"` (usamos aspas devido ao espaço em branco entre as palavras) ou um "nome de família genérico" como `sans-serif` (nomes de família genéricos nunca usam aspas).

  Se um navegador não puder encontrar ou não suportar a primeira fonte em uma lista, ele usará a próxima, depois a próxima e assim por diante até encontrar uma fonte compatível e válida. É por isso que é uma prática recomendada incluir uma lista de valores para essa propriedade, começando com a fonte que você mais deseja usar e terminando com uma família de fontes genérica como alternativa, por exemplo `font-family: "Times New Roman", sans-serif;`

- `font-size` irá, como o nome da propriedade sugere, definir o tamanho da fonte. Ao atribuir um valor a esta propriedade, o valor não deve conter nenhum espaço em branco, por exemplo `font-size: 22px`, não há espaço entre “22” e “px”.
- `font-weight` afeta o negrito do texto, supondo que a fonte suporte o peso especificado. Esse valor pode ser uma palavra-chave, por exemplo `font-weight: bold`, ou um número entre 1 e 1000, por exemplo `font-weight: 700`(o equivalente a `bold`). Normalmente, os valores numéricos estarão em incrementos de 100 a 900, embora isso dependa da fonte.
- `text-align` irá alinhar o texto horizontalmente dentro de um elemento, e você pode usar as palavras-chave comuns que você pode encontrar em processadores de texto como o valor para esta propriedade, por exemplo `text-align: center`.

Altura e largura da imagem

As imagens não são os únicos elementos nos quais podemos ajustar a altura e a largura, mas queremos focar nelas especificamente neste caso.

Por padrão, os valores `height` e `width` de um elemento `<img>` serão os mesmos que a altura e a largura do arquivo de imagem real. Se você quisesse ajustar o tamanho da imagem sem fazer com que ela perdesse suas proporções, você usaria um valor de “auto” para a propriedade `height` e ajustaria o valor `width`:

```
img {

    height: auto;

    width: 500px;

  }
```

Por exemplo, se o tamanho original de uma imagem tiver 500 px de altura e 1.000 px de largura, usar o CSS acima resultaria em uma altura de 250 px.

É melhor incluir essas duas propriedades para elementos `<img>`, mesmo se você não planeja ajustar os valores dos originais do arquivo de imagem. Quando esses valores não são incluídos, se uma imagem demorar mais para carregar do que o restante do conteúdo da página, a imagem não ocupará nenhum espaço na página a princípio, mas causará uma mudança drástica no conteúdo da outra página. uma vez que ele carrega. Afirmar explicitamente a `height` e `width` impede que isso aconteça, pois o espaço será "reservado" na página e aparecerá apenas como um espaço em branco até que a imagem seja carregada.

A cascata (cascade) de CSS

Às vezes, podemos ter regras que entram em conflito umas com as outras e acabamos com alguns resultados inesperados. "Mas eu queria que *esses* parágrafos fossem azuis, por que eles são vermelhos como esses outros parágrafos?!" Por mais frustrante que isso possa ser, é importante entender que o CSS não *faz* as coisas apenas contra nossos desejos. **CSS só faz o que mandamos fazer**. Uma exceção a isso são os estilos padrão fornecidos por um navegador. Esses estilos padrão variam de navegador para navegador, e é por isso que alguns elementos criam uma grande "lacuna" entre eles e outros elementos, ou porque os botões têm a aparência que têm, apesar de não escrevermos nenhuma regra CSS para estilizá-los dessa maneira.

Então, se você acabar com algum comportamento inesperado como esse, é por causa desses estilos padrão ou por não entender como uma propriedade funciona ou não entender essa coisinha chamada cascata.

A cascata é o que determina quais regras realmente são aplicadas ao nosso HTML. Existem diferentes fatores que a cascata usa para determinar isso, três dos quais vamos falar para ajudá-lo a evitar (como muitos) aqueles momentos frustrantes "Eu odeio CSS".

Especificidade

Uma declaração CSS mais específica terá precedência sobre as menos específicas. Os estilos inline, que veremos mais na seção Adicionando CSS ao HTML no final da lição, têm a maior especificidade em comparação com os seletores, enquanto cada tipo de seletor tem seu próprio nível de especificidade que contribui para a especificidade de uma declaração. Outros seletores contribuem para a especificidade, mas estamos focando apenas nos mencionados nesta lição:

1. Seletores de ID (seletor mais específico)
2. Seletores de classe
3. Seletores de tipo

A especificidade só será levada em consideração quando um elemento tiver várias declarações conflitantes direcionadas a ele, como um **desempate**. Um seletor de ID sempre vencerá qualquer número de seletores de classe, um seletor de classe sempre vencerá qualquer número de seletores de tipo e um seletor de tipo sempre vencerá qualquer número de qualquer coisa menos específica do que ele. *Quando nenhuma declaração possui um seletor com maior especificidade, uma quantidade maior de um único seletor vencerá uma quantidade menor desse mesmo seletor.*

Vamos dar uma olhada em alguns exemplos rápidos para visualizar como a especificidade funciona. Considere o seguinte código HTML e CSS:

```
<!-- index.html -->



<div class="main">

  <div class="list subsection"></div>
```

```
</div>
```

```
/* rule 1 */

.subsection {

    color: blue;

  }



  /* rule 2 */

  .main .list {

    color: red;

  }
```

No exemplo acima, ambas as regras estão usando apenas seletores de classe, mas a regra 2 é mais específica porque está usando mais seletores de classe, então a `color: red;` declaração teria precedência.

Agora vamos mudar um pouco as coisas:

```
<!-- index.html -->
```

```
<div class="main">
  <div class="list" id="subsection"></div>
</div>
```

```
/* rule 1 */
#subsection {
    color: blue;
  }

  /* rule 2 */
  .main .list {
    color: red;
  }
```

No exemplo acima, apesar da regra 2 ter mais seletores de classe do que seletores de ID, a regra 1 é mais específica porque ID supera classe. Neste caso, a `color: blue;` declaração teria precedência.

```
/* rule 1 */

#subsection .list {

    background-color: yellow;

    color: blue;

  }



/* rule 2 */

#subsection .main .list {

    color: red;

  }
```

Neste exemplo final, ambas as regras estão usando seletores de ID e classe, portanto, nenhuma regra está usando um seletor mais específico que a outra. A cascata então verifica as quantidades de cada tipo de seletor. Ambas as regras têm apenas um seletor de ID, mas a regra 2 tem mais seletores de classe, então a regra 2 tem uma especificidade maior!

Embora a declaração `color: red` tenha precedência, a declaração `background-color: yellow` ainda será aplicada, pois não há declaração conflitante para ela.

Nota: Ao comparar seletores, você pode encontrar símbolos especiais para o seletor universal (`*`), bem como combinadores (`+`, `~`, `>`, e um espaço vazio). Esses símbolos não adicionam nenhuma especificidade em si.

```
/* rule 1 */

.class.second-class {

    font-size: 12px;

  }



/* rule 2 */

.class .second-class {

    font-size: 24px;

  }
```

Aqui, tanto a regra 1 quanto a regra 2 têm a mesma especificidade. A regra 1 usa um seletor de encadeamento (sem espaço) e a regra 2 usa um combinador descendente (o espaço vazio). Mas ambas as regras têm duas classes e o próprio símbolo combinador não adiciona especificidade.

```
/* rule 1 */

.class.second-class {

    font-size: 12px;

  }



/* rule 2 */

.class > .second-class {

    font-size: 24px;

  }
```

Este exemplo mostra a mesma coisa. Embora a regra 2 esteja usando um combinador filho (>), isso não altera o valor de especificidade. Ambas as regras ainda têm duas classes para que tenham os mesmos valores de especificidade.

```
/* rule 1 */

* {

    color: black;

  }
```

```
/* rule 2 */

h1 {

    color: orange;

  }
```

Neste exemplo, a regra 2 teria maior especificidade e o valor `orange` teria precedência para este elemento. A regra 2 usa um seletor de tipo, que tem o menor valor de especificidade. Mas a regra 1 usa o seletor universal (`*`) que não tem valor de especificidade.

Herança

Herança refere-se a certas propriedades CSS que, quando aplicadas a um elemento, são herdadas pelos descendentes desse elemento, mesmo que não escrevamos explicitamente uma regra para esses descendentes. Propriedades baseadas em tipografia (`color`, `font-size`, `font-family`, etc.) geralmente são herdadas, enquanto a maioria das outras propriedades não são.

A exceção a isso é quando direcionar diretamente um elemento, pois isso sempre supera a herança:

```
<!-- index.html -->



<div id="parent">
```

```
  <div class="child"></div>

</div>
```

```
/* styles.css */



#parent {

    color: red;

  }



.child {

    color: blue;

  }
```

Apesar do elemento `parent` ter uma especificidade maior com um ID, o elemento `child` teria o estilo `color: blue` aplicado, pois essa declaração o direciona diretamente, enquanto do pai `color: red` é apenas herdado.

Ordem da regra

O fator final, o fim da linha, o desempate do desempate. Digamos que, depois que todos os outros fatores forem levados em consideração, ainda existam várias regras conflitantes visando um elemento. Como a cascata determina qual regra aplicar?

Muito simples, na verdade. *A última* regra definida é a vencedora.

```
/* styles.css */



.alert {

    color: red;

  }



.warning {

    color: yellow;

  }
```

Para um elemento que tem as classes `alert` e `warning`, a cascata passaria por todos os outros fatores, incluindo herança (nenhuma aqui) e especificidade (nenhuma regra é mais específica que a outra). Como a regra `.warning` foi a última definida, e nenhum outro fator foi capaz de determinar qual regra aplicar, é aquela que é aplicada ao elemento.

Adicionando CSS ao HTML

Ok, nós vimos bastante conteúdo até agora. A única coisa que resta por enquanto é ver *como* adicionar todo esse CSS ao nosso HTML. Existem três métodos para fazê-lo.

CSS externo

O CSS externo é o método mais comum que você encontrará e envolve a criação de um arquivo separado para o CSS e vinculá-lo dentro das tags de abertura e fechamento de um HTML `<head>` com um elemento `<link>` de fechamento automático:

```
<!-- index.html -->



<head>

  <link rel="stylesheet" href="styles.css">

</head>
```

```
/* styles.css */



div {

    color: white;
```

```
    background-color: black;

  }



p {

    color: red;

  }
```

Primeiro, adicionamos um elemento `<link>` de fechamento automático dentro das tags de abertura e fechamento `<head>` do arquivo HTML. O `href` atributo é a localização do arquivo CSS, seja uma URL absoluta ou, o que você utilizará, uma URL relativa à localização do arquivo HTML. Em nosso exemplo acima, estamos assumindo que ambos os arquivos estão localizados no mesmo diretório. O atributo `rel` é obrigatório e especifica o relacionamento entre o arquivo HTML e o arquivo vinculado.

Em seguida, dentro do arquivo recém-criado `styles.css`, temos o seletor (o `div` e `p`), seguido por um par de chaves de abertura e fechamento, que criam um “bloco de declaração”. Finalmente, colocamos quaisquer declarações dentro do bloco de declaração. `color: white;` é uma declaração, sendo `color` a propriedade e `white` sendo o valor, e `background-color: black;`é outra declaração.

Uma observação sobre nomes de arquivos: `styles.css` é exatamente o que usamos como nome de arquivo aqui. Você pode nomear o arquivo como quiser, desde que o tipo de arquivo seja `.css`, embora "style" ou "styles" sejam os mais usados.

Alguns dos prós deste método são:

1. Ele mantém nosso HTML e CSS separados, o que resulta em um arquivo HTML menor e deixando as coisas mais limpas.
2. Só precisamos editar o CSS em *um só* lugar, o que é especialmente útil para sites com muitas páginas que compartilham estilos semelhantes.

CSS interno

CSS interno (ou CSS incorporado) envolve adicionar o CSS dentro do próprio arquivo HTML em vez de criar um arquivo completamente separado. Com o método interno, você coloca todas as regras dentro de um par de tags `<style>` de abertura e fechamento, que são então colocadas dentro das tags `<head>` de abertura e fechamento do seu arquivo HTML. Como os estilos estão sendo colocados diretamente dentro das tags `<head>`, não precisamos mais de um elemento `<link>` que o método externo requer.

Além dessas diferenças, a sintaxe é exatamente a mesma do método externo (seletor, chaves, declarações):

```
<head>

  <style>

    div {

      color: white;

      background-color: black;

    }
```

```
    p {
      color: red;
    }
  </style>
</head>
<body>...</body>
```

Esse método pode ser útil para adicionar estilos exclusivos a uma *única página* de um site, mas não mantém as coisas separadas como o método externo e, dependendo de quantas regras e declarações existem, pode fazer com que o arquivo HTML fique muito grande .

CSS embutido (inline)

O CSS embutido possibilita adicionar estilos diretamente aos elementos HTML, embora esse método não seja o recomendado:

```
<body>
  <div style="color: white; background-color:
black;">...</div>
</body>
```

A primeira coisa a notar é que não usamos nenhum seletor aqui, já que os estilos estão sendo adicionados diretamente à própria tag `<div>` de abertura. Em seguida, temos o atributo `style=`, com seu valor dentro do par de aspas sendo as declarações.

Se você precisar adicionar um estilo *único* para um *único* elemento, esse método pode funcionar bem. Geralmente, porém, essa não é exatamente uma maneira recomendada de adicionar CSS ao HTML por alguns motivos:

- Pode ficar muito confuso rapidamente quando você começa a adicionar *muitas* declarações a um único elemento, fazendo com que seu arquivo HTML fique desnecessariamente inchado.
- Se você quiser que muitos elementos tenham o mesmo estilo, você teria que copiar + colar o mesmo estilo para cada elemento individual, causando muitas repetições desnecessárias e mais inchaço.
- Qualquer CSS embutido substituirá os outros dois métodos, o que pode causar resultados inesperados. (Embora não nos aprofundemos aqui, isso pode realmente ser aproveitado).

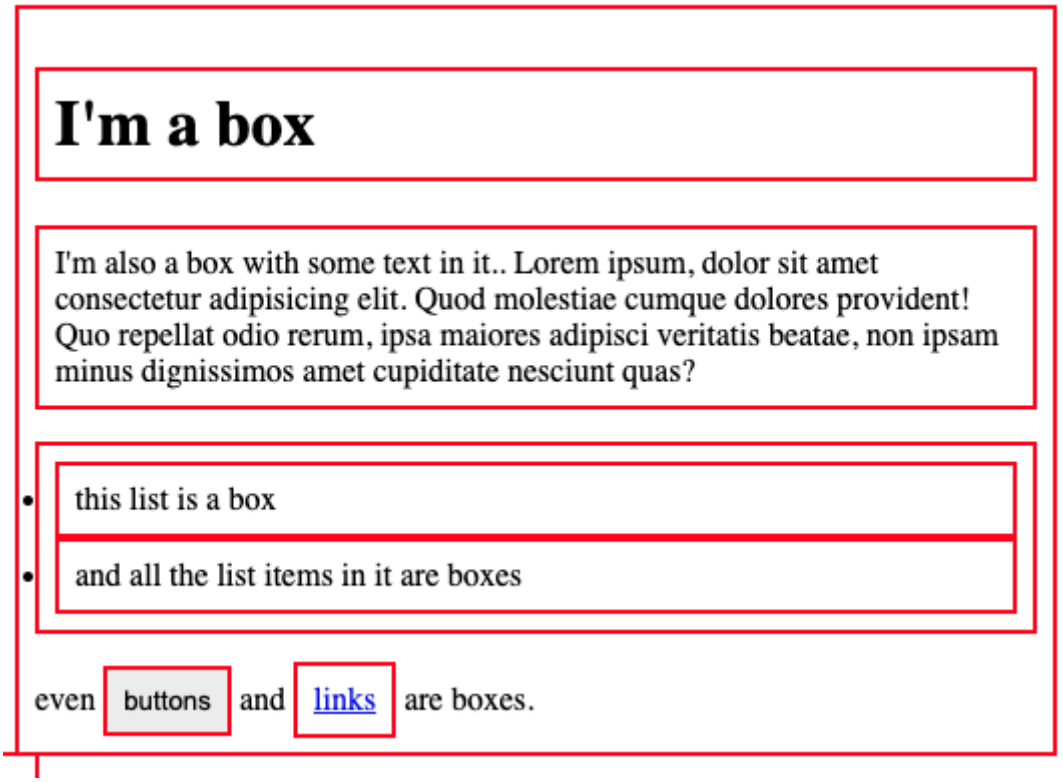
Modelo de caixa (margin, padding, border)

Tudo o que é exibido pelo CSS é uma caixa.

O primeiro conceito importante que você precisa entender para ter sucesso em CSS é o modelo de caixa. Não é complicado, mas pulá-lo agora causará muita frustração no futuro.

Cada coisa em uma página da web é uma caixa retangular. Essas caixas podem ter outras caixas e podem ficar lado a lado. Você pode ter uma ideia aproximada de como isso funciona colando uma borda em cada item da página assim:

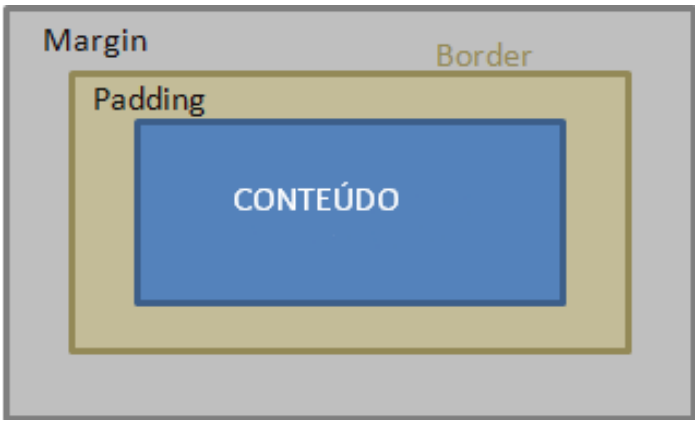
```
/* styles.css */


* {


  border: 2px solid red;


}
```



É uma caixa, embalada em torno de modelo de caixa de elementos HTML CSS na natureza, incluindo: margens, bordas, preenchimento e conteúdo real.

O modelo de caixa permite o espaço entre o outro elemento e colocado em torno de elemento de borda do elemento.

A figura abaixo ilustra o modelo de caixa (Box Model):



- `padding` Aumenta o espaço entre a borda de uma caixa e o conteúdo dentro dela.

CSS tem propriedades para especificar o preenchimento de cada lado de um elemento:

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

Se a propriedade padding tiver quatro valores:

- **padding: 25px 50px 75px 100px;**
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px

Se a propriedade padding tiver três valores:

- **padding: 25px 50px 75px;**
  - top padding is 25px
  - right and left paddings are 50px
  - bottom padding is 75px

Se a propriedade padding tiver dois valores:

- **padding: 25px 50px;**
  - top and bottom paddings are 25px
  - right and left paddings are 50px

- `margin` Aumenta o espaço entre uma caixa e quaisquer outros que se sentam ao lado dela.

CSS tem propriedades para especificar a margem para cada lado de um elemento:

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

**Abreviação semelhante à propriedade de `padding`.

- `border` adiciona espaço (mesmo que seja apenas um pixel ou dois) entre a margem e o preenchimento.

A propriedade `border` é uma propriedade abreviada para as seguintes propriedades de borda individuais:

- `border-width`
- `border-style` (obrigatório)
- `border-color`

Forma abreviada:

```
p {
    border: 5px solid red;
  }
```

Exemplos - Box Model CSS

Vamos utilizar a marcação abaixo. Nós temos uma *div* dentro da outra e um parágrafo só para ilustrar o conteúdo.

```
<div class="externo">
    <div class="interno">
        <p>Eu sou o conteúdo!</p>
    </div>
</div>
```

Passo 1

Vamos aplicar uma borda preta na div externa.

```
.externo {
    border: 1px solid black;
}
```

Passo 2

E agora uma borda vermelha para a div interna.

```
.interno {
    border: 1px solid red;
}
```

Passo 3

Então vamos dar um espaçamento ? Podemos dar um `padding` de `10px` em cada lado.

```
.interno {
    border: 1px solid red;
    padding: 10px 10px;
}
```

Passo 4

Vamos aplicar `10px` nos 4 cantos da margem da div interior.

```
.interno {
```

```
    border: 1px solid red;
    padding: 10px 10px;
    margin: 10px 10px;
}
```

Passo 5 (último)

Para terminar vamos aplicar uma cor no background.

```
.interno {
    border: 1px solid red;
    padding: 10px 10px;
    margin: 10px 10px;
    background-color: orange;
}
```