✍️

Guia de Código do @mdo

Padrões para desenvolver código HTML e CSS flexíveis, duráveis e sustentáveis.

Índice

HTML

- Sintaxe
- Doctype do HTML5
- Encoding de caractéres
- Inserindo CSS e JavaScript
- Praticidade em vez de pureza
- Ordem dos Atributos
- Atributos booleanos
- Reduzindo o markup
- Código gerado por JavaScript

CSS

- Sintaxe do CSS
- Ordem das declarações
- Onde colocar as Media Queries
- Propriedades com prefixo
- Declarações únicas
- Propriedades resumidas
- Aninhamento (nesting) no LESS e SASS
- Comentários
- Nomes de classes
- Seletores
- Organização

Regra de ouro

Reforce isso, concordando com estas guidelines todas as vezes. Pequenos ou grandes, mostre o que está

incorreto. Para adições ou contribuições para este Guia de Código, por favor abra uma Issue no Github.

> Toda linha de código deve parecer que foi escrita por uma única pessoa, não importa a quantidade de contribuidores.

HTML

Sintaxe do HTML

- Use soft-tabs com dois espaços.
- Elementos aninhados devem estar identados uma vez (2 espaços).
- Sempre use aspas duplas, nunca aspas simples.
- Nunca inclua uma barra invertida nos elementos viúvos (exemplo: `<img>` ou `<hr>`).

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <img src="images/company-logo.png" alt="Company">
    <h1 class="hello-world">Hello, world!</h1>
  </body>
</html>
```

Doctype do HTML5

Reforce os padrões com este simples doctype no início de todas as páginas HTML.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
</html>
```

Encoding de caractéres

Assegure-se rapidamente a renderização do seu conteúdo declarando o

encoding de caractéres explicitamente.

```
<head>
  <meta charset="UTF-8">
</head>
```

Inserindo CSS e JavaScript

De acordo com as especificações do HTML5, não há necessidade de especificar um atributo `type` quando incluímos arquivos de CSS e JavaScript como `text/css` e `text/javascript`.

Links para especificação do HTML5

- Usando link
- Usando style
- Usando scripts

```
<!-- External CSS -->
<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->
<style>
  /* ... */
</style>

<!-- JavaScript -->
<script src="code-guide.js"></script>
```

Praticidade em vez de pureza

Esforce-se para manter o padrão e a semântica do HTML, mas não às custas da praticidade. Use o mínimo de código com a menor complexidade possível.

Ordem dos atributos

Atributos HTML devem usar esta ordem em particular para facilitar a leitura do código.

- `class`
- `id`

- `data-*`
- `for`, `type`, ou `href`
- `src`, `for`, `type`, `href`
- `title`, `alt`
- `aria-*`, `role`

```
<a class="..." id="..." data-modal="toggle" href="#">
  Example link
</a>

<input class="form-control" type="text">

<img src="..." alt="...">
```

Atributos booleanos

Um atributo boleano é o único que não precisa ter valores declarados. O XHTML precisa que você declare um valor, mas no HTML5 isso não é necessário.

Para uma leitura adicional, consulte a seção do WhatWG sobre atributos booleanos:

> A presença de um atributo boleano representa um valor verdadeiro e a ausência do atributo representa um valor falso.

Se você *precisa* incluir valores nos atributos, **você não precisa** seguir esta regra do WhatWG:

> Se o atributo está presente, seu valor deve ser uma string vazia ou [...] o nome canônico do atributo, sem espaços em branco.

Resumindo, não adicione nenhum valor.

```
<input type="text" disabled>

<input type="checkbox" value="1" checked>

<select>
  <option value="1" selected>1</option>
</select>
```

Reduzindo o markup

Sempre que possível, evite elementos pais supérfluos quando escrever HTML. Muitas vezes é necessário uma interação ou refatoração, mas isso produz pouco HTML. Veja o seguinte exemplo:

```
<!-- Not so great -->
<span class="avatar">
  <img src="...">
</span>

<!-- Better -->
<img class="avatar" src="...">
```

Markup gerado por Javascript

Escrever código em um arquivos Javascript faz com que o conteúdo seja difícil de ser encontrado, difícil de ser editado e o torna menos performático. Evite fazer isso o máximo possível.

CSS

Sintaxe do CSS

- Use soft-tabs com dois espaços.
- Quando agrupar seletores, deixe seletores individuais em uma linha.
- Inclua um espaço antes de abrir um colchete.
- Coloque o fechamento do colchete em uma nova linha.
- Inclua um espaço depois dos dois pontos `:` em cada declaração de propriedade.
- Cada declaração deve estar em sua própria linha para encontrarmos melhor os erros.
- Feche todas as declarações com ponto-vírgula `;`.

- Valores separados por vírgula devem incluir um espaço logo depois da vírgula (ex., `box-shadow`).
- Não inclua espaços depois das vírgulas *como em* cores <`rgb()`, `rgba()`, `hsl()`, `hsla()` ou `rect()`. Isto ajuda a diferenciar múltiplos valores (vírgulas, sem espaço) de cores de propriedades múltiplas (vírgulas, com espaço). Também não coloque valores iniciados com zero (ex., `.5` em vez de `0.5`).
- Deixe todos os valores hexadecimais em caixa baixa, exemplo: `#fff`. Letras com caixa baixa são mais fáceis de serem lidas e entendidas quando queremos scanear o documento.
- Valores separados por vírgula devem incluir um espaço logo depois da vírgula.
- Não inclua espaços depois das vírgulas *como em* cores `rgb()` ou `rgba()` e não inicie valores com zero.
- Deixe todos os valores hexadecimais em caixa baixa, exemplo: `#fff`.
- Use cores em hexadecimal abrevidas quando puder, exemplo: `#fff` em vez de `#ffffff`.
- Coloque aspas nos valores em seletores com atributos, exemplo: `input[type="text"]`.
- Evite especificar unidades quando os valores forem 0, exemplo: `margin: 0;` em vez de `margin: 0px;`.

Dúvidas sobre os termos usados aqui? Veja a seção de sintaxe dos CSS na Wikipedia.

```
/* Bad CSS */
.selector, .selector-secondary, .selector[type=text] {
  padding:15px;
  margin:0px 0px 15px;
  background-color:rgba(0, 0, 0, 0.5);
  box-shadow:0 1px 2px #CCC,inset 0 1px 0 #FFFFFF
}

/* Good CSS */
.selector,
.selector-secondary,
.selector[type="text"] {
  padding: 15px;
  margin: 0 0 15px;
  background-color: rgba(0,0,0,.5);
  box-shadow: 0 1px 2px #ccc, inset 0 1px 0 #fff;
}
```

Ordem das declarações

Declarações relacionadas devem ser agrupadas segundo a seguinte ordem:

1. Posicionamento
2. Box model
3. Tipografia
4. Visual

Posicionamento vem primeiro por que isto pode remover um elemento do fluxo normal do documento e substituir estilos relacionados. O box model vem depois pois ele dita as dimensões e lugar do componente.

Tudo o mais que toma lugar *dentro* do componente ou não impacta as duas seções anteriores, vem por último.

Para uma lista completa de propridades e suas ordens, por favor veja Recess,

Declarações relacionadas devem ser agrupadas, colocando posicionamento e as propriedades de box-model perto do topo,

seguido das propriedades de tipografia e depois propriedades visuais.

Para uma lista completa das propriedades e suas ordens, por favor verifique a página do Recess.

```
.declaration-order {
  /* Positioning */
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 100;

  /* Box-model */
  display: block;
  float: right;
  width: 100px;
  height: 100px;

  /* Typography */
  font: normal 13px "Helvetica Neue", sans-serif;
  line-height: 1.5;
  color: #333;
  text-align: center;

  /* Visual */
  background-color: #f5f5f5;
  border: 1px solid #e5e5e5;
  border-radius: 3px;

  /* Misc */
  opacity: 1;
}
```

Onde colocar as Media Queries

Coloque as Media Queries o mais perto possível de suas regras originais. Não as coloque separadas em outros arquivos ou no final do documento. Se você fizer isso, outros poderão não encontrá-las no futuro. Veja um exemplo típico:

```
.element { ... }
.element-avatar { ... }
.element-selected { ... }

@media (min-width: 480px) {
  .element { ...}
  .element-avatar { ... }
  .element-selected { ... }
}
```

Propriedades com prefixo

Quando usar prefixos de browsers, idente cada propriedade alinhada verticalmente para facilitar a edição multi-linha.

No Textmate, use **Text → Edit Each Line in Selection** (^⌘A). No Sublime Text 2, use **Selection → Add Previous Line** (^⇧↑) e **Selection → Add Next Line** (^⇧↓).

```
/* Prefixed properties */
.selector {
  -webkit-box-shadow: 0 1px 2px rgba(0,0,0,.15);
          box-shadow: 0 1px 2px rgba(0,0,0,.15);
}
```

Declarações únicas

Em lugares onde são definidas apenas uma linha de propriedade, considere remover as quebras de linha para melhorar a leitura e edição. Considere este exemplo:

```
/* Single declarations on one line */
.span1 { width: 60px; }
.span2 { width: 140px; }
.span3 { width: 220px; }

/* Multiple declarations, one per line */
.sprite {
  display: inline-block;
  width: 16px;
  height: 15px;
  background-image: url(../img/sprite.png);
}
.icon           { background-position: 0 0; }
.icon-home      { background-position: 0 -20px; }
.icon-account   { background-position: 0 -40px; }
```

Propriedades resumidas

Esforce-se muito para usar declarações de propriedades resumidas

onde você define todos os valores dessa propriedade. As propriedades resumidas mais usadas incluem:

- `padding`
- `margin`
- `font`
- `background`
- `border`
- `border-radius`

Frequentemente não precisamos definir todos os valores que uma propriedade representa. Por exemplo, os títulos do HTML tem margens no top e bottom definidas por padrão, então, quando necessário, apenas substitua estes dois valores. O uso excessivo de propriedades resumidas (ou shorthand properties) pode fazer com que o código fique um pouco bagunçado quando substituímos propriedades não utilizadas.

O Mozilla Developer Network tem um ótimo artigo em shorthand properties para quem não está familiarizado essa forma de escrever.

```
/* Bad example */
.element {
  margin: 0 0 10px;
  background: red;
  background: url("image.jpg");
  border-radius: 3px 3px 0 0;
}

/* Good example */
.element {
  margin-bottom: 10px;
  background-color: red;
  background-image: url("image.jpg");
  border-top-left-radius: 3px;
  border-top-right-radius: 3px;
}
```

Aninhamento (nesting) no LESS e SASS

Evite aninhar seletores (fazer "nesting"). Só porque você pode fazer isso, não significa que você deve fazê-lo sempre. Considere aninhar se você tem um escopo de estilos para um pai e se há múltiplos elementos para serem aninhados.

```
// Without nesting
.table > thead > tr > th { … }
.table > thead > tr > td { … }

// With nesting
.table > thead > tr {
  > th { … }
  > td { … }
}
```

Legível para humanos

Comentários

Código é escrito e mantido por pessoas. Certifique-se de que o código é descritivo, bem comentado e amigável para os outros. Grandes pedaços de comentários devem ter contexto e não devem apenas reiterar um nome de classe ou componente.

Certifique-se de escrever em sentenças completas ou grandes comentários e frases sucintas para notas gerais.

```
/* Bad example */
/* Modal header */
.modal-header {
  ...
}

/* Good example */
/* Wrapping element for .modal-title and .modal-close */
.modal-header {
```

```
  ...
}
```

Nomes de classes

- Mantenha as classes em caixa baixa e use traços (não use underscores ou camelCase).
- Evite usar arbitrariamente notações abreviadas.
- Mantenha-as pequenas e sucintas ao máximo possível.
- Use nomes com significado; use nomes ligados à estrutura em vez do visual.
- Prefixe nomes de classes baseadas nos componentes pais mais próximos.

```
/* Bad example */
.t { ... }
.red { ... }
.header { ... }

/* Good example */
.tweet { ... }
.important { ... }
.tweet-header { ... }
```

Seletores

- Use classes em vez de nomes de tags.
- Mantenha os seletores pequenos e limite o número de elementos em no máximo três.
- Use nomes de classes do elemento pais **somente** quando necessário (ex: quando não usar classes prefixadas).

Leitura adicional:

- Scope CSS classes with prefixes
- Stop the cascade

```
/* Bad example */
span { ... }
.page-container #stream .stream-item .tweet .tweet-header .username { ... }
.avatar { ... }
```

```
/* Good example */
.avatar { ... }
.tweet-header .username { ... }
.tweet .avatar { ... }
```

Organização

- Organize seções do código por componentes.
- Desenvolva uma hierarquia de comentários consistente.
- Se usar múltiplos arquivos CSS, quebre-os em componentes.

Preferências de editor

Defina seu editor com as seguintes configurações para evitar inconsistências comuns no código e diffs sujos:

- Use soft-tabs com 2 espaços
- Apague os espaços em brancos ao salvar
- Defina o encoding como UTF-8
- Coloque uma nova linha no final dos arquivos

Considere documentar e aplicar estas configurações para o seu projeto com o `.editorconfig`. Para um exemplo, veja o arquivo utilizado no Bootstrap. Aprenda mais em EditorConfig

<3

Muito inspirado por Idiomatic CSS e o GitHub Styleguide. Feito com todo amor do mundo por @mdo.

Versão em português
mantida por @diegoeis e
contribuidores via Github.

- Star 88
- Fork 1,116

- Seguir @diegoeis 8.635 seguidores
- Tweetar