

# Aprenda Markdown



**Markdown** é uma ferramenta de conversão de *text-to-HTML*. Com ele é possível você marcar títulos, listas, tabelas, etc., de forma muito mais limpa, legível e precisa, do que se fosse fazer com HTML. Ainda não sabe tudo o que é possível fazer com Markdown? Então esse post é pra você! Vem comigo que eu te mostro :)

## Onde usar Markdown

Existem vários lugares que você pode usar **Markdown**: No **Github** mesmo, você pode usar no *README.md*, que é um arquivo que fica na raiz do seu projeto, e é renderizado pelo **Github** abaixo da lista de arquivos. Aquele texto que você lê quando acessa um repositório é um arquivo *README.md*, escrito em **Markdown**.

Ainda no **Github**, você pode usar **Markdown** no texto das issues, no texto de um *pull request* e na *wiki*.

Também a maior parte dos [geradores de estáticos](#) (Hexo, Jekyll, HarpJS, Docpad, etc.), permitem escrever em **Markdown**! É uma ótima oportunidade pra você que queria começar o seu blog, mas estava com preguiça de usar um CMS ou ficar fazendo HTML na mão xD

## Porque eu devo aprender e usar Markdown

Porque HTML é muito verboso! **Markdown** é mais legível, mais fácil de ler e interpretar.

## Devo parar de escrever HTML para sites e fazer tudo em Markdown?

Não! **Markdown** serve basicamente para escrever textos. Como toda ferramenta, ele tem algumas desvantagens com relação à escrever em HTML:

- Você não consegue colocar atributos nos elementos (class, id, title, etc.), além dos poucos que ele permite por padrão;
- Você não tem muito controle para fazer aninhamento de tags.

Por isso é importante frisar que o uso do **Markdown** deve ser especificamente para a escrita de textos, artigos de blog, etc. Não é para simplesmente usá-lo no lugar do HTML!

## Como eu começo a escrever em Markdown

Se você já tiver um *parser*, você só precisa criar um arquivo com uma dessas extensões: mark, markdown, md, mdml, mdown, text, mdtext, mdtxt, mdwn, mkd, mkdn.

Entre essas, a mais utilizada é a `_md_`. Crie um arquivo com uma dessas extensões e você já estará apto para começar a escrever em **Markdown**!

Agora, se você não tiver um *parser*, pode usar uma ferramenta online. Existem muitas ferramentas de *parse* de **Markdown** para **HTML**. Vou recomendar uma para que você possa acompanhar esse post, mas buscando por **Markdown Editor** no Google, você vai encontrar muitas outras!

Gosto bastante do [Stack Edit](#). Ele converte **Markdown** para **HTML** enquanto você digita. É bastante simples para ver o resultado :)

Vamos aprender então como escrever em **Markdown**!

## Títulos (<h1> a <h6>)

Para marcar um título, você vai usar # a quantidade de vezes que irá representar o nível do título. Exemplo:

```
1 # Título nível 1
2 ## Título nível 2
3 ### Título nível 3
4 #### Título nível 4
5 ##### Título nível 5
6 ##### Título nível 6
```

Parseia para:

```
1 <h1>Título nível 1</h1>
2 <h2>Título nível 2</h2>
3 <h3>Título nível 3</h3>
4 <h4>Título nível 4</h4>
5 <h5>Título nível 5</h5>
6 <h6>Título nível 6</h6>
```

Bastante simples, não? O h1 e o h2 ainda podem ser escritos da seguinte forma:

```
1 Título de nível 1
2 =====
3
4 Título de nível 2
5 -----
```

## Parágrafos e quebras de linha (<p> e <br />)

Para gerar parágrafos, basta você escrever o texto em uma linha:

```
1 Este é um parágrafo.
2
3 Este é outro parágrafo.
```

Isso gera:

```
1 <p>Este é um parágrafo.</p>
2 <p>Este é outro parágrafo.</p>
```

Note que eu pulei uma linha entre os parágrafos. Se eu não fizesse isso, o código gerado seria:

```
1 <p>Este é um parágrafo. Este é outro parágrafo.</p>
```

*Mas ele não deveria usar um ` para quebrar linha?*

Isso é muito particular de cada parser. Alguns quebram linha quando você dá enter. Mas a documentação do **Markdown** diz que, para quebras de linha, você precisa deixar dois espaços no final da linha:

```
1 Primeira linha do parágrafo.··
2 Segunda linha do parágrafo.
```

Coloquei o .. no final da primeira linha somente para facilitar a visualização. Você deve substituir esse símbolo por dois espaços em branco. Isso deve gerar:

```
1 <p>
2  Primeira linha do parágrafo.<br />
3  Segunda linha do parágrafo.
4 </p>
```

Por isso, se você estiver usando o [.editorconfig](#) no seu projeto, deixe a opção trim\_trailing\_whitespace como false para arquivos **Markdown**. Assim, os espaços adicionais não serão removidos :)

## Ênfase (<strong> e <em>)

Para enfatizar uma palavras (<em>), usamos um \* ou \_:

```
1 Javascript é _cool_!
```

ou:

```
1 Javascript é *cool*!
```

Que irá gerar:

```
1 <p>
2   Javascript é <em>cool</em>!
3 </p>
```

O mais utilizado para ênfase (<em>) é o *underline*.

Para dar forte ênfase em palavras (<strong>), você usa dois \*\* ou \_\_:

```
1 **Da2k** é a pronúncia para **Daciuk**: DA-TW0-K!
```

ou

```
1 __Da2k__ é a pronúncia para __Daciuk__: DA-TW0-K!
```

Que irá gerar:

```
1 <p>
2   <strong>Da2k</strong> é a pronúncia para <strong>Daciuk:</strong> DA-TW0-K!
3 </p>
```

O mais utilizado para forte ênfase (<strong>) são dois asteriscos.

## Links (<a>)

Para gerar links, você usa [](). Dentro dos colchetes você coloca o texto do link, e dentro dos parênteses, você coloca a URL:

```
1 [Blog do Da2k](https://blog.da2k.com.br)
```

Que irá gerar:

```
1 <a href="https://blog.da2k.com.br">Blog do Da2k</a>
```

Passando um texto após a URL, separando o link do texto por um espaço em branco, esse texto será usado como title:

```
1 [Blog do Da2k](https://blog.da2k.com.br "Clique e acesse agora!")
```

Vai gerar:

```
1 <a href="https://blog.da2k.com.br" title="Clique e acesse agora!">Blog do Da2k</a>
```

## Links automáticos

Se o texto do seu link é o próprio link, você pode envolvê-lo entre < e >, que o link será gerado automaticamente:

```
1 <https://www.google.com.br>
```

Irá gerar:

```
1 <a href="https://www.google.com.br">https://www.google.com.br</a>
```

E isso funciona também para e-mails:

```
1 <meu@email.com>
```

Vai gerar:

```
1 <a href="mailto:meu@email.com">meu@email.com</a>
```

Da hora, não? ;)

## Referências

Expliquei sobre referências [nesse post](#).

## Blocos de citação (<blockquote>)

Para criar blocos de citação, você usa o sinal de >:

```
1 > Esse é um bloco de citação.
2 > Ele pode ter várias linhas por parágrafo.
3 >
4 > Inclusive, dando um espaço, você tem um novo parágrafo.
```

Que gera o seguinte:

```
1 <blockquote>
2   <p>
3     Esse é um bloco de citação.
4     Ele pode ter várias linhas.
5   </p>
6 <p>Inclusive, uma quebra de linha.</p>
7 </blockquote>
```

## Listas (<ul> e <ol>)

Para listas não ordenadas (<ul>), você pode usar \*, + ou -. Veja:

```
1 * Item 1
2 * Item 2
3 * Item 3
4
5 + Item 1
6 + Item 2
7 + Item 3
8
9 - Item 1
10 - Item 2
11 - Item 3
```

Os três formatos acima geram a mesma marcação:

```
1 <ul>
2   <li>Item 1</li>
3   <li>Item 2</li>
4   <li>Item 3</li>
5 </ul>
```

E para listas ordenadas, você usa o número, seguido de ponto:

```
1 1. Item 1
2 2. Item 2
3 3. Item 3
```

Que irá gerar:

```
1 <ol>
2   <li>Item 1</li>
3   <li>Item 2</li>
4   <li>Item 3</li>
5 </ol>
```

Alguns parsers renderizam automaticamente os próximos números, após o 1. Você só precisa usar \* para os itens do 2 em diante:

```
1 1. Item 1
2 * Item 2
3 * Item 3
```

Mas não são todos que renderizam dessa forma, então é bom ficar ligado ;)

## Imagens (<img />)

Geração de imagens é bem parecido com a geração de links: você só precisa adicionar uma ! no início. E o texto que você coloca entre os colchetes, é usado como alt na imagem:

```
1 ![Banana](http://cdn.osxdaily.com/wp-content/uploads/2013/07/dancing-banana.gif)
```

Esse código vai gerar:

```
1 
```

O title também funciona como no link:

```
1 ![Banana](http://cdn.osxdaily.com/wp-content/uploads/2013/07/dancing-banana.gif "Olha a banana dançando!")
```

Que gera:

```
1 
```

## Tabelas (<table>)

Já falei sobre tabelas em um [post anterior](#). Nesse post eu falo também sobre as task lists, mas elas são específicas do **Github**, não funcionam com qualquer parser ;)

## Código inline e bloco (<code> e <pre>)

Você ainda pode adicionar trechos de código via **Markdown**. Para adicionar código a nível *inline*, você usa `:

```
1 0 `<blockquote>` é uma tag HTML.
```

Isso irá gerar:

```
1 <p>
2 0 <code>&lt;blockquote&gt;</code> é uma tag HTML!
3 </p>
```

E para gerar blocos de código, você simplesmente indenta o código 4 espaços (ou 1 tab) à frente do parágrafo:

```
1 Essa é a função sayHello():
2   function sayHello() {
3     return 'hi!';
4   }
```

Que irá gerar:

```
1 <p>
2   Essa é a função sayHello():
3   <pre><code>function sayHello() {
4     return 'hi!';
5   }</code></pre>
6 </p>
```

Isso é como está na documentação. Mas a maior parte dos parses que eu conheço não funcionam dessa forma. Eles geram blocos de código usando três crases no início da primeira e última linha, para marcar o início e o fim do bloco:

```
...
function sayHello() {
  return 'hi!';
}
...
```

PS.: Tive que colocar como imagem, pois o meu parser não consegue escapar as 3 crases ⇐⇐

O **Github** inclusive [recomenda que se use as 3 crases](#), pois é mais fácil de visualizar e dar manutenção no código.

No **Github**, você ainda consegue definir qual a linguagem que está sendo utilizada, para que seja feito *code highlight* no seu código. Só passe a linguagem após as 3 crases, dessa forma:

```
```js
function sayHello() {
  return 'hi!';
}
```
```

Que ó seu código será mostrado bonitinho assim:

```
function sayHello() {
  return 'hi!';
}
```

:D

## Backslash scapes

Para escapar caracteres que são parseados pelo **Markdown**, você pode usar a barra invertida \ (*backslash*), seguida do caractere, para imprimí-lo literalmente. O escape funciona para os caracteres listados abaixo:

|       |   |
|-------|---|
| 1 \   | backslash (barra invertida)                   |
| 2 `   | backtick (crase)                              |
| 3 *   | asterisk (asterisco)                          |
| 4 _   | underscore                                    |
| 5 { } | curly braces (chaves)                         |
| 6 [ ] | square brackets (colchetes)                   |
| 7 ( ) | parentheses (parênteses)                      |
| 8 #   | hash mark (sustenido / hash / jogo da velha)  |
| 9 +   | plus sign (sinal de "mais" ou somar)          |
| 10 -  | minus sign (hyphen) (sinal de menos ou hífen) |
| 11 .  | dot (ponto)                                   |
| 12 !  | exclamation mark (ponto de exclamação)        |

Além de tudo isso, é importante saber também, que é possível usar HTML junto com **Markdown**! Isso mesmo! Se você precisar adicionar uma classe em uma imagem para alinhar, ou colocar uma cor específica em alguma palavra, você pode usar tags HTML normalmente :D

Para saber mais sobre **Markdown**, recomendo a leitura da documentação oficial:

<http://daringfireball.net/projects/markdown/>

E alguns links de como o **Github** usa **Markdown**:

<https://help.github.com/articles/markdown-basics/>

<https://help.github.com/articles/github-flavored-markdown/>

<https://guides.github.com/features/mastering-markdown/>

<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>