

16 DE MARÇO DE 2023 / #HTML

# Manual de HTML – aprendizagem de HTML para iniciantes



Tradutor: Nielda Karla



Autor: Flavio Copes (em inglês)

# THE HTML HANDBOOK

Flavio Copes

Artigo original: [The HTML Handbook – Learn HTML for Beginners](#)

## Introdução

Boas-vindas! Eu escrevi esse manual para ajudar você a aprender HTML de maneira rápida e para se familiarizar com tópicos avançados de HTML.

HTML, forma abreviada de *Hypertext Markup Language* (em português, linguagem de marcação de hipertexto), é um dos blocos fundamentais da web.

O HTML nasceu oficialmente em 1993 e, desde então, evoluiu até o ponto atual, partindo de simples documentos de texto até as poderosas aplicações para a web.

Este manual é destinado a um público bem amplo.

Primeiro, o público iniciante. Eu explico HTML do zero de uma maneira sucinta, mas abrangente. Assim, você pode usar este manual para aprender HTML desde o básico.

Depois, temos o público profissional. O HTML é, com frequência, considerado algo secundário a ser aprendido e, muitas vezes, não recebe a devida atenção.

Ainda assim, diversos aspectos do HTML permanecem um mistério para muitas pessoas, inclusive para o autor deste artigo. Escrevi este manual para me ajudar a entender melhor esse tópico. Afinal, quando se decide ensinar algo, melhor se certificar de que se conhece o assunto de uma ponta a outra.

Mesmo que você não escreva HTML no seu dia a dia de trabalho, saber como o HTML funciona ajudará você a evitar dores de cabeça quando precisar entender do tema em algum momento, por exemplo, ao tentar ajustar algo em uma página da web.

Você pode encontrar o autor pelo [Twitter](#) ou no [site do autor](#).

Observação: você também pode fazer o download deste manual em inglês no formato PDF/ePub/Mobi e lê-lo off-line.

## Sumário

- [Prefácio](#)
- [HTML Básico](#)
- [Título do documento](#)
- [Corpo do documento](#)
- [Tags que interagem com texto](#)
- [Links](#)
- [Tags de contêiner e estrutura da página do HTML](#)
- [Formulários](#)
- [Tabelas](#)
- [Tags de multimídia: \*\*audio\*\* e \*\*video\*\*](#)
- [Iframes](#)
- [Imagens](#)
- [Acessibilidade](#)

## Prefácio

O HTML é a base da maravilha que chamamos de web.

Existe um poder incrível por baixo desse conjunto de regras simples e limitado, que nos permite – desenvolvedores, criadores, designers, escritores e pensadores – criar documentos, aplicações e experiências para pessoas em todo o planeta.

Meu primeiro livro sobre HTML saiu em 1997 e era chamado "HTML Unleashed", um livro grande e com muitas páginas.

Mais de vinte anos se passaram, mas o HTML ainda é a base da web, com mudanças mínimas desde então.

Claro, existem mais *tags* semânticas, o HTML de apresentação não é mais uma tendência, o CSS agora é o responsável pelo design das coisas, entre outras questões.

O sucesso do HTML é baseado em uma coisa: simplicidade.

Ele resistiu ao sequestro pelo dialeto XML via XHTML, quando, no fim, as pessoas notaram que era algo demasiadamente complexo.

Isso aconteceu por causa de outra característica que ele nos proporciona: **indulgência**. Existem *algumas* regras, certo, mas depois que você aprende o HTML, passa a ter muita liberdade.

Os navegadores aprenderam a ser resilientes e a sempre tentar fazer o melhor ao analisar e apresentar o HTML aos usuários.

Toda a plataforma da web fez uma coisa certa: nunca quebrou a compatibilidade com versões anteriores. Incrivelmente, podemos voltar aos documentos em HTML escritos em 1991 e eles se parecem muito com o que víamos na época.

Sabemos até qual foi a primeira página da web. Foi esta aqui:

<http://info.cern.ch/hypertext/WWW/TheProject.html>

Você pode ver a fonte da página, graças a outro grande recurso da web e do HTML: **podemos inspecionar o HTML de qualquer página da web**.

Não considere isso uma coisa óbvia. Não conheço nenhuma outra plataforma que nos dê essa capacidade.

As excepcionais Ferramentas de Desenvolvedor integradas a qualquer navegador nos permitem inspecionar e nos inspirar no HTML escrito por qualquer pessoa no mundo.

Se você é novo em HTML, este manual tem como objetivo ajudá-lo a começar. Se você é um desenvolvedor da web experiente, este manual ajudará a aprimorar seu conhecimento.

Aprendi muito escrevendo este manual, mesmo trabalhando com a web há mais de 20 anos, e tenho certeza de que você também encontrará algo novo.

Talvez você vá reaprender algo antigo de que você esqueceu.

De qualquer modo, o objetivo do manual é ser útil para você. Espero que ele seja bem-sucedido nisso.

# HTML BÁSICO

HTML é um padrão definido pelo **WHATWG**, um acrônimo para *Web Hypertext Application Technology Working Group*, uma organização formada por pessoas que trabalham nos navegadores da web mais populares. Isso significa que é basicamente controlado por Google, Mozilla, Apple e Microsoft.

No passado, o W3C (*World Wide Web Consortium*) era a organização encarregada de criar o padrão HTML.

O controle mudou informalmente do W3C para o WHATWG quando ficou claro que o impulso do W3C em direção ao XHTML não era uma boa ideia.

Se você nunca ouviu falar de XHTML, aqui está uma pequena história. No início dos anos 2000, todos acreditávamos que o futuro da web era o XML (sério). Assim, o HTML deixou de ser uma linguagem de autoria baseada em SGML e passou a ser uma linguagem de marcação XML.

Foi uma grande mudança. Tínhamos que conhecer e respeitar mais regras, regras mais rígidas.

No fim, os fornecedores de navegadores perceberam que esse não era o caminho certo para a web e recuaram, criando o que hoje é conhecido como HTML5.

O W3C realmente não concordou em abrir mão do controle do HTML. Durante anos, tivemos dois padrões concorrentes, cada um com o objetivo de ser o oficial. Ao fim, em 28 de maio de 2019, foi oficializado pelo W3C que a versão do HTML "verdadeira" era a publicada pelo WHATWG.

Eu mencionei HTML5. Deixe-me falar dessa pequena história. Eu sei, é meio confuso até agora, como acontece com muitas coisas na vida quando muitos atores estão envolvidos, mas também é fascinante.

Tínhamos a **versão HTML 1** em 1993. [Aqui está o RFC original](#) (em inglês).

O **HTML 2** foi lançado em 1995.

Recebemos o **HTML 3** em janeiro de 1997 e o **HTML 4** em dezembro de 1997.

Muita coisa aconteceu!

Mais de 20 anos se passaram. Houve toda essa onda de XHTML e, ao final, chegamos a essa "coisa" chamada HTML5, que não é mais *apenas* HTML.

HTML5 é um termo que agora define todo um conjunto de tecnologias, que inclui HTML, mas adiciona muitas APIs e padrões como WebGL, SVG e muito mais.

A chave para entender é a seguinte: agora, não existe (mais) uma versão de HTML. É um padrão de vida. Isso é o mesmo que ocorre com o CSS, que é chamado de "3", mas, na realidade, são vários módulos independentes desenvolvidos separadamente. Também é o mesmo que ocorre com o JavaScript, onde temos uma nova edição a cada ano, mas, hoje em dia, a única coisa que importa é quais recursos individuais são implementados pelo mecanismo.

Sim, nós o chamamos de HTML5, mas o HTML4 é de 1997. Isso é muito tempo para qualquer coisa, especialmente para a web.

É aqui que o padrão "vive" agora: <https://html.spec.whatwg.org/multipage> (em inglês).

O HTML é a linguagem de marcação que usamos para estruturar o conteúdo que consumimos na web.

Ele é servido no navegador de diferentes maneiras.

- Ele pode ser gerado por uma aplicação do lado do servidor que o constrói dependendo da solicitação ou dos dados da sessão, por exemplo, uma aplicação de Rails, Laravel ou Django.
- Ele pode ser gerado por uma aplicação em JavaScript do lado do client que gera HTML em tempo real.
- No caso mais simples, ele pode ser armazenado em um arquivo e servido no navegador por um servidor da web.

Vamos nos aprofundar nesse último caso. Embora, na prática, seja provavelmente o modo menos popular de se gerar HTML, ainda é essencial conhecer seus blocos básicos de construção.

Por convenção, um arquivo HTML é salvo com uma extensão `.html` ou `.htm`.

Dentro desse arquivo, organizamos o conteúdo usando **tags**.

As *tags* envolvem o conteúdo e cada *tag* dá um significado especial ao texto que envolve.

Vamos mostrar alguns exemplos.

Este trecho de HTML cria um parágrafo usando a tag `p`:

```
<p>Um parágrafo de texto</p>
```

Este trecho de HTML cria uma lista de itens usando a tag `ul`, que significa lista não ordenada. As tags `li` significam *item de lista*:

```
<ul>
  <li>Primeiro item</li>
  <li>Segundo item</li>
  <li>Terceiro item</li>
</ul>
```

Quando uma página de HTML é servida pelo navegador, as tags são interpretadas e o navegador renderiza os elementos de acordo com as regras que definem sua aparência visual.

Algumas dessas regras são incorporadas, como quando uma lista é renderizada ou quando um link é sublinhado em azul.

Algumas outras regras são definidas por você com CSS.

O HTML não se destina à apresentação. Ele não está preocupado com a aparência das coisas. Em vez disso, seu foco é naquilo que as coisas *significam*.

Cabe ao navegador determinar como as coisas ficam, com as diretivas definidas por quem constrói a página, usando a linguagem CSS.

Agora, esses dois exemplos que fiz são trechos de HTML tirados de um contexto de página.

## Estrutura de página do HTML

Vamos criar um exemplo de uma página do HTML adequado.

As coisas começam com a declaração do **tipo de documento** (também conhecido como *doctype*), uma maneira de dizer ao navegador que esta é uma página em HTML e qual versão do HTML estamos usando.

O HTML moderno usa este doctype:

```
<!DOCTYPE html>
```

Então, temos o elemento `html`, que possui uma tag de abertura e outra de fechamento:

```
<!DOCTYPE html>
<html>
...
</html>
```

A maioria das tags vem em pares com uma tag de abertura e uma tag de fechamento. A tag de fechamento é escrita da mesma forma que a tag de abertura, mas com um `/`:

```
<exemplodetag>Conteúdo</exemplodetag>
```

Existem algumas tags de fechamento automático, o que significa que elas não precisam de uma tag de fechamento separada, pois *não contêm nada nelas*.

A tag inicial `html` é usada no início do documento, logo após a declaração do tipo de documento.

A tag final `html` é a última coisa presente em um documento HTML.

Dentro do elemento `html` temos 2 elementos, `head` e `body`:

```
<!DOCTYPE html>
<html>
```

```
<head>
...
</head>
<body>
...
</body>
</html>
```

Dentro de `head`, teremos tags que são essenciais para a criação de uma página web, como o título, os metadados, o CSS e o JavaScript internos ou externos. Em especial, coisas que não aparecem diretamente na página, mas apenas ajudam o navegador (ou bots como o bot de pesquisa do Google) a exibi-la corretamente.

Dentro do `body`, teremos o conteúdo da página em si – **as coisas visíveis**.

## Tags x elementos

Eu mencionei tags e elementos. Qual é a diferença?

Os elementos têm uma tag inicial e uma tag de fechamento. Neste exemplo, usamos as tags de abertura e de fechamento de `p` para criar um elemento `p`:

```
<p>Um parágrafo de texto</p>
```

Assim, um elemento constitui *todo o pacote*:

- tag de abertura
- conteúdo de texto (e, possivelmente, outros elementos)
- tag de fechamento

Se um elemento não tiver tag de fechamento, ele será escrito apenas com a tag de abertura e não poderá conter **nenhum conteúdo de texto**.

Dito isso, posso usar o termo tag ou elemento no manual significando a mesma coisa, exceto se mencionar explicitamente a tag de abertura ou a tag de fechamento.

## Atributos

A tag de abertura de um elemento pode ter trechos especiais de informações que podemos anexar. Esses trechos são chamados de **atributos**.

Atributos têm a sintaxe `chave="valor"`:

```
<p class="uma-classe">Um parágrafo de texto</p>
```

Você também pode usar aspas simples, mas usar aspas duplas em HTML é uma boa convenção.

Podemos ter muitos atributos:

```
<p class="uma-classe" id="um-id">Um parágrafo de texto</p>
```

Alguns atributos são booleanos, o que significa que você só precisa da chave:

```
<script defer src="file.js"></script>
```

Os atributos `class` e `id` são dois dos mais comuns que você encontrará sendo usados.

Eles têm um significado especial e são úteis tanto em CSS quanto em JavaScript.

A diferença entre os dois é que um `id` é único no contexto de uma página da web. Ele não pode ser duplicado.

As classes `class`, por outro lado, podem aparecer várias vezes em vários elementos.

Além disso, um `id` é apenas um valor. `class` pode conter vários valores, separados por um espaço:

```
<p class="uma-classe outra-classe">Um parágrafo de texto</p>
```

É comum usar o traço `-` para separar palavras em um valor de classe, mas é apenas uma convenção.

Esses são apenas dois dos atributos possíveis que você pode ter. Alguns atributos são usados apenas para uma tag. Eles são altamente especializados.

Outros atributos podem ser usados de maneira mais geral. Você acabou de ver `id` e `class`, mas temos outros também, como `style`, que pode ser usado para inserir regras CSS em linha (do inglês, *inline*) em um elemento.

## Diferenciação de maiúsculas e minúsculas

HTML **não** diferencia maiúsculas de minúsculas. As tags podem ser escritas em letras maiúsculas ou minúsculas. Inicialmente, AS MAIÚSCULAS eram a norma. Hoje, letras minúsculas são a norma. É apenas uma convenção.

Geralmente, escrevemos assim:

```
<p>Um parágrafo de texto</p>
```

Não escrevemos as *tags* assim:



```
<P>Um parágrafo de texto</P>
```

## Espaço em branco

Importante: em HTML, mesmo que você adicione vários espaços em branco em uma linha, ela é *recolhida* pelo mecanismo do CSS do navegador.

Por exemplo, a renderização deste parágrafo:

```
<p>Um parágrafo de texto</p>
```

é a mesma que esse:

```
<p>    Um parágrafo de texto</p>
```

e que esse também:

```
<p>Um parágrafo  
de  
    texto    </p>
```

Usando a propriedade do [white-space, do CSS](#), você pode alterar como as coisas se comportam. Você pode encontrar mais informações sobre como o CSS processa o espaço em branco no [CSS Spec](#) (texto em inglês).

Eu, geralmente, prefiro:

```
<p>Um parágrafo de texto</p>
```

ou

```
<p>  
    Um parágrafo de texto  
</p>
```

As tags aninhadas devem ser recuadas com 2 ou 4 caracteres, dependendo de sua preferência:

```
<body>
  <p>
    Um parágrafo de texto
  </p>
  <ul>
    <li>Um item de lista</li>
  </ul>
</body>
```

Observação: esse recurso em que o "espaço em branco não é relevante" significa que, se você quiser adicionar espaço adicional, pode acabar bem irritado. Sugiro que você use CSS para ganhar mais espaço quando necessário.

Observação: em casos especiais, você pode usar a entidade `&nbsp;` do HTML (um acrônimo que significa *espaço sem quebra*) – mais sobre entidades do HTML posteriormente. Acho que isso não deve ser abusado. O CSS é sempre preferido para alterar a apresentação visual.

## TÍTULO DO DOCUMENTO

A tag `head` contém tags especiais que definem as propriedades do documento.

É sempre escrita antes da tag `body` e logo após a tag `html` de abertura:

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

Nunca usamos atributos nessa tag. Também não escrevemos conteúdo nela.

Ela é apenas um recipiente para outras tags. Dentro dela, podemos ter uma grande variedade de tags, dependendo do que você precisa fazer:

- `title`
- `script`
- `noscript`
- `link`
- `style`
- `base`
- `meta`

### Tag `title`

A tag `title` determina o título da página. O título é exibido no navegador e é especialmente importante, pois é um dos principais fatores para a otimização de mecanismos de pesquisa (SEO).

## Tag `script`

Essa tag é usada para adicionar JavaScript à página.

Você pode incluí-lo em linha, usando uma tag de abertura, o código JavaScript e depois a tag de fechamento:

```
<script>
... código em JS aqui
</script>
```

Você também pode carregar um arquivo JavaScript externo usando o atributo `src`:

```
<script src="arquivo.js"></script>
```

O atributo `type`, por padrão, é definido como `text/javascript`. Portanto, é totalmente opcional.

Há algo muito importante a saber sobre essa tag.

Às vezes, ela é usada na parte inferior da página, logo antes da tag de fechamento de `</body>`. Por quê? Por motivos de desempenho.

Carregar scripts por padrão bloqueia a renderização da página até que o script seja analisado e carregado.

Ao colocar os scripts na parte inferior da página, o script é carregado e executado após a página inteira já ser analisada e carregada, proporcionando uma experiência melhor ao usuário em vez de mantê-los na tag `head`.

Minha opinião é que isso agora é uma má prática. Deixe o `script` em paz na tag `head`.

No JavaScript moderno, temos uma alternativa que é mais eficiente do que manter o script na parte inferior da página - o atributo `defer`. Este é um exemplo que carrega um arquivo `arquivo.js`, relativo ao URL atual:

```
<script defer src="arquivo.js"></script>
```

Esse é o cenário que aciona o caminho mais rápido para uma página de carregamento rápido e para JavaScript de carregamento rápido.

Observação: o atributo `async` é semelhante, mas, na minha opinião, é uma opção pior do que `defer`. Descrevo o porquê, com mais detalhes, na página <https://flaviocopes.com/javascript-async-defer/> (texto em inglês).

## Tag `noscript`

Essa tag é usada para detectar quando os scripts estão desabilitados no navegador.

Observação: os usuários podem optar por desabilitar os scripts de JavaScript nas configurações do navegador. O navegador também pode não os suportar por padrão.

Essa tag é usada de modos diferentes, dependendo do fato de ser colocada no cabeçalho do documento ou no corpo do documento.

Estamos falando sobre o cabeçalho do documento agora. Então, vamos primeiro apresentar esse uso.

Nesse caso, a tag `noscript` pode conter apenas outras tags:

- `link` tags
- `style` tags
- `meta` tags

Ela serve para alterar os recursos servidos pela página, ou as informações de `meta`, se os scripts estiverem desabilitados.

Neste exemplo, configurei um elemento com a classe `no-script-alert` para que seja exibido se os scripts estiverem desabilitados, pois era `display: none` por padrão:

```
<!DOCTYPE html>
<html>
  <head>
    ...
    <noscript>
      <style>
        .no-script-alert {
          display: block;
        }
      </style>
    </noscript>

    ...
  </head>
  ...
</html>
```

Vamos resolver o outro caso: se colocado no corpo, pode conter conteúdo, como parágrafos e outras tags, que são renderizadas na UI.

## Tag `link`

A tag de `link` é usada para definir relacionamentos entre um documento e outros recursos.

É usada principalmente para vincular um arquivo CSS externo a ser carregado.

Esse elemento não possui tag de fechamento.

Uso:

```
<!DOCTYPE html>
<html>
  <head>
    ...
    <link href="file.css" rel="stylesheet">
    ...
  </head>
  ...
</html>
```

O atributo `media` permite o carregamento de diferentes folhas de estilo, dependendo das capacidades do dispositivo:

```
<link href="file.css" media="screen" rel="stylesheet">
<link href="print.css" media="print" rel="stylesheet">
```

Também podemos vincular a outros recursos além de folhas de estilo.

Por exemplo, podemos associar a um feed de RSS usando

```
<link rel="alternate" type="application/rss+xml" href="/index.xml">
```

Ou podemos associar um favicon usando:

```
<link rel="apple-touch-icon" sizes="180x180" href="/assets/apple-touch-icon.png">
<link rel="icon" type="image/png" sizes="32x32" href="/assets/favicon-32x32.png">
<link rel="icon" type="image/png" sizes="16x16" href="/assets/favicon-16x16.png">
```

Essa tag também foi usada para conteúdo de múltiplas páginas, para indicar a página anterior e a próxima usando `rel="prev"` e `rel="next"`. Principalmente para o Google. A partir de 2019, o Google anunciou que não usa mais essa tag porque pode encontrar a estrutura correta da página sem ela.

## Tag `style`

Essa tag pode ser usada para adicionar estilos ao documento, em vez de carregar uma folha de estilo externa.

Uso:

```
<style>
... algum css {}
```

```
</style>
```

Assim como na tag `link`, você pode usar o atributo `media` para usar esse CSS apenas na mídia especificada:

```
<style media="print">
... algum css {}
</style>
```

## Tag `base`

Essa tag é usada para definir um URL base para todos os URLs relativos contidos na página.

```
<!DOCTYPE html>
<html>
  <head>
    ...
    <base href="https://flaviocopes.com/">
    ...
  </head>
  ...
</html>
```

## Tag `meta`

Tags `meta` executam uma variedade de tarefas e são muito, muito importantes, especialmente para SEO.

Elementos `meta` só tem a tag de abertura.

A mais básica é a tag `meta` de descrição, ou `description`:

```
<meta name="description" content="Uma página legal">
```

Ela pode ser usada pelo Google para gerar a descrição da página em suas páginas de resultados, se achar que descreve melhor a página do que o conteúdo da página (não me pergunte como).

A tag `meta charset` é usada para definir a codificação de caracteres da página. Essa codificação é a `utf-8`, na maioria dos casos:

```
<meta charset="utf-8">
```

A tag `meta robots` instrui os bots de mecanismos de busca a indexar uma página ou não:

```
<meta name="robots" content="noindex">
```

Ou se eles devem seguir os links ou não:

```
<meta name="robots" content="nofollow">
```

Você também pode definir `nofollow` em links individuais. É assim que você pode definir o `nofollow` globalmente.

Você pode combiná-los:

```
<meta name="robots" content="noindex, nofollow">
```

O comportamento padrão é `index, follow`.

Você pode usar outras propriedades, incluindo `nosnippet`, `noarchive`, `noimageindex` e mais.

Você também pode simplesmente informar algo ao Google em vez de visar *todos* os mecanismos de pesquisa:

```
<meta name="googlebot" content="noindex, nofollow">
```

Outros mecanismos de pesquisa também podem ter sua própria tag `meta`.

Falando nisso, podemos dizer ao Google para desativar alguns recursos. Isso impede a funcionalidade de tradução nos resultados do mecanismo de pesquisa:

```
<meta name="google" content="notranslate">
```

A tag `meta viewport` é usada para informar ao navegador para definir a largura da página com base na largura do dispositivo.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

[Leia mais sobre essa tag aqui](#) (texto em inglês).

Outra tag `meta` bastante popular é a `http-equiv="refresh"`. Esta linha diz ao navegador para esperar 3 segundos e redirecionar para outra página:

```
<meta http-equiv="refresh" content="3;url=http://flaviocopes.com/outra-pagina">
```

Usar 0 em vez de 3 redirecionará o mais rápido possível.

Esta não é uma referência completa. Existem outras tags `meta` menos usadas.

Após essa introdução ao título do documento, podemos começar a mergulhar no corpo do documento.

## CORPO DO DOCUMENTO

Após o fechamento da tag `head`, só podemos ter uma coisa em um documento HTML: o elemento `body`.

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

Assim como as tags `head` e `html`, só podemos ter uma tag `body` em uma página.

Dentro da tag `body` temos todas as tags que definem o conteúdo da página.

Tecnicamente, as tags de abertura e de fechamento de `body` são opcionais. Considero, no entanto, uma boa prática adicioná-las, apenas para maior clareza.

Nos próximos capítulos, definiremos a variedade de tags que você pode usar dentro do corpo da página.

Antes, porém, devemos introduzir uma diferença entre elementos de bloco e elementos em linha (do inglês, *inline*).

## Elementos de bloco x elementos em linha

Os elementos visuais, aqueles definidos no corpo da página, podem ser geralmente classificados em 2 categorias:

- elementos de bloco (`p`, `div`, elementos de título, listas, itens de lista, ...)
- elementos em linha (`a`, `span`, `img`, ...)

Qual é a diferença?

Elementos de bloco, quando posicionados na página, não permitem outros elementos próximos a eles, para a esquerda ou para a direita.



Os elementos em linha podem ficar ao lado de outros elementos em linha.

A diferença também está nas propriedades visuais que podemos editar usando CSS. Podemos alterar a largura/altura, margem, preenchimento e borda dos elementos em bloco. Não podemos fazer isso para elementos em linha.

Observe que, usando CSS, podemos alterar o padrão para cada elemento, definindo uma tag `p` para que seja em linha, por exemplo, ou um `span` para ser um elemento em bloco.

Outra diferença é que os elementos em linha podem estar contidos em elementos em bloco. O contrário, no entanto, não é verdade.

Alguns elementos em bloco podem conter outros elementos em bloco, mas isso depende. A tag `p`, por exemplo, não permite tal opção.

## TAGS QUE INTERAGEM COM TEXTO

### Tag `p`

Essa tag define um parágrafo de texto.

```
<p>Um texto</p>
```

É um elemento de bloco.

Dentro dele, podemos adicionar qualquer elemento em linha que desejarmos, como `span` ou `a`.

Não podemos adicionar elementos em bloco.

Não podemos aninhar um elemento `p` em outro.

Por padrão, os navegadores estilizam um parágrafo com uma margem na parte superior e na parte inferior. Essa margem é de `16px` no Chrome, mas o valor exato pode variar entre os navegadores.

Isso faz com que dois parágrafos consecutivos fiquem espaçados, replicando o que pensamos de um "parágrafo" no texto impresso.

### Tag `span`

Essa é uma tag em linha que pode ser usada para criar uma seção em um parágrafo. Essa seção pode ser segmentada usando CSS:

```
<p>Uma parte do texto <span>e aqui vai outra parte</span></p>
```

# Tag `br`

Essa tag representa uma quebra de linha. É um elemento em linha e não precisa de uma tag de fechamento.

Usamos para criar uma linha dentro de uma tag `p` sem criar um parágrafo.

Em comparação com a criação de um parágrafo, não adiciona espaçamento a mais.

```
<p>Um texto<br>Uma nova linha</p>
```

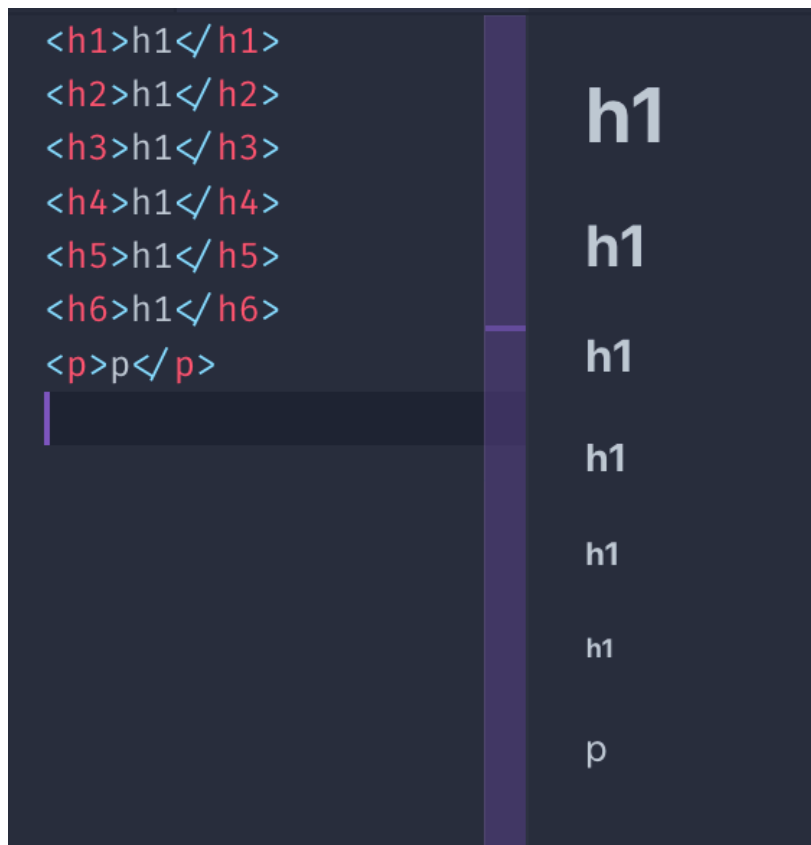
## Tags de título

O HTML nos fornece 6 tags de título. Do mais importante ao menos importante, temos `h1`, `h2`, `h3`, `h4`, `h5` e `h6`.

Normalmente, uma página terá um elemento `h1`, que é o título da página. Então, você pode ter um ou mais elementos `h2` dependendo do conteúdo da página.

Os títulos, especialmente a organização dos títulos, também são essenciais para o SEO. Os mecanismos de pesquisa os usam de várias maneiras.

O navegador, por padrão, tornará a tag `h1` maior e diminuirá o tamanho dos elementos à medida que o número próximo a `h` aumentar:



Todos os títulos são elementos em bloco. Eles não podem conter outros elementos, apenas texto.

## Tag `strong`

Essa tag é usada para marcar o texto dentro dela como *forte*. Isso é muito importante. Não é uma dica visual, mas uma dica semântica. Dependendo do meio utilizado, sua interpretação vai variar.

Os navegadores, por padrão, deixam o texto dessa tag em **negrito**.

## Tag `em`

Essa tag é usada para marcar o texto dentro dela como *ênfatisado*. Do mesmo modo que em `strong`, não é uma dica visual, mas uma dica semântica.

Os navegadores, por padrão, deixam o texto em **itálico**.

## Citações

A tag HTML `blockquote` é útil para inserir citações no texto.

Os navegadores, por padrão, aplicam uma margem ao elemento `blockquote`. O Chrome aplica uma margem à esquerda e à direita de 40 px e uma margem superior e inferior de 10 px.

A tag HTML `q` é usada para aspas em linha.

## Linha horizontal

Essa tag não é realmente baseada em texto. A tag `hr` é frequentemente usada dentro de uma página. "hr", aqui, significa `horizontal rule` e adiciona uma linha horizontal na página.

É útil para separar seções na página.

## Bloco de código

A tag `code` é especialmente útil para mostrar código, pois os navegadores fornecem uma fonte monoespaçada.

Essa é normalmente a única coisa que os navegadores fazem. Este é o CSS aplicado pelo Chrome:

```
code {  
  font-family: monospace;  
}
```

Essa tag é normalmente envolvida em uma tag `pre` porque o elemento `code` ignora espaços em branco e quebras de linha, assim como a tag `p`.

O Chrome dá a `pre` este estilo padrão:

```
pre {  
  display: block;  
  font-family: monospace;  
  white-space: pre;  
  margin: 1em 0px;  
}
```

Este estilo evita a remoção do espaço em branco e o torna um elemento em bloco.

## Listas

Temos 3 tipos de listas:

- listas não ordenadas
- listas ordenadas
- listas de definição

As listas não ordenadas são criadas usando a tag `ul`. Cada item da lista é criado com a tag `li`:

```
<ul>  
  <li>Primeiro</li>  
  <li>Segundo</li>  
</ul>
```

As listas ordenadas são semelhantes, mas são criadas com a tag `ol`:

```
<ol>  
  <li>Primeiro</li>  
  <li>Segundo</li>  
</ol>
```

A diferença entre os dois é que as listas ordenadas têm um número antes de cada item:

```

<ul>
  <li>First</li>
  <li>Second</li>
</ul>

<ol>
  <li>First</li>
  <li>Second</li>
</ol>

```

- First
  - Second
1. First
  2. Second

As listas de definição são um pouco diferentes. Você tem um termo e sua definição:

```

<dl>
  <dt>Flavio</dt>
  <dd>O nome</dd>
  <dt>Copes</dt>
  <dd>O sobrenome</dd>
</dl>

```

É assim que os navegadores normalmente os renderizam:

```

<dl>
  <dt>Flavio</dt>
  <dd>The name</dd>
  <dt>Copes</dt>
  <dd>The surname</dd>
</dl>

```

Flavio  
The name  
Copes  
The surname

Devo dizer que você raramente verá listas de definição por aí. Com certeza, não tanto quanto `ul` e `ol`. Às vezes, contudo, elas podem ser úteis.

## Outras tags de texto

Aqui está uma série de tags com fins de apresentação:

- tag `mark`
- tag `ins`
- tag `del`
- tag `sup`
- tag `sub`
- tag `small`
- tag `i`
- tag `b`

Este é um exemplo da renderização visual deles que é aplicado por padrão pelos navegadores:



Você pode se perguntar, como `b` é diferente de `strong`? Como `i` é diferente de `em`?

A diferença está no significado semântico. Enquanto `b` e `i` são uma dica direta ao navegador para deixar um trecho de texto em negrito ou em itálico, `strong` e `em` dão ao texto um significado especial, cabendo ao navegador dar o estilo. Esses estilos são exatamente os mesmos que os de `b` e `i`, por padrão. Você pode, no entanto, alterar isso usando o CSS.

Existem várias outras tags menos usadas relacionadas ao texto. Mencionei aqui apenas as que vejo sendo mais usadas.

## LINKS

Os links são definidos usando a tag `a`. O destino do link é definido por meio de seu atributo `href`.

Exemplo:

```
<a href="https://flaviocopes.com">Clique aqui</a>
```

Entre a tag de abertura e a tag de fechamento temos o texto do link.

O exemplo acima é um URL absoluto. Os links também funcionam com URLs relativos:

```
<a href="/test">Clique aqui</a>
```

Nesse caso, ao clicar no link, o usuário é movido para o URL `/test` na origem atual.

Tenha cuidado com o caractere `/`. Se omitido, em vez de começar da origem, o navegador apenas adicionará a string `test` ao URL atual.

Exemplo, estou na página <https://flaviocopes.com/axios/> e tenho esses links:

- `/test` quando clicado me leva para `https://flaviocopes.com/test`
- `test` quando clicado me leva para `https://flaviocopes.com/axios/test`

As tags de link podem incluir outras coisas dentro delas, não apenas texto. Podem incluir, por exemplo, imagens:

```
<a href="https://flaviocopes.com">  
    
</a>
```

Também podem ter quaisquer outros elementos, exceto outras tags `<a>`.

Caso queira abrir o link em uma nova aba, você pode usar o atributo `target`:

```
<a href="https://flaviocopes.com" target="_blank">Abra em uma nova aba</a>
```

# TAGS DE CONTÊINER E ESTRUTURA DA PÁGINA DO HTML

## Tags de contêiner

O HTML fornece um conjunto de tags de contêiner. Essas tags podem conter um conjunto não especificado de outras tags.

Temos:

- `article`
- `section`

- `div`

Pode ser confuso entender a diferença entre elas.

Vamos ver quando usar cada uma.

## `article`

A tag `article` identifica *algo* que pode ser independente de outras coisas em uma página.

Por exemplo, uma lista de publicações em blog na página inicial.

Ou uma lista de links.

```
<div>
  <article>
    <h2>Uma publicação em blog</h2>
    <a ...>Leia mais</a>
  </article>
  <article>
    <h2>Outra publicação em blog</h2>
    <a ...>Leia mais</a>
  </article>
</div>
```

Não estamos limitados a listas: um artigo (em inglês, *article*) pode ser o elemento principal de uma página.

```
<article>
  <h2>Uma publicação em blog</h2>
  <p>Este é o conteúdo...</p>
</article>
```

Dentro de uma tag `article` devemos ter um título ( `h1` - `h6` ) e

## `section`

Representa uma seção de um documento. Cada seção tem uma tag de título ( `h1` - `h6` ) e, em seguida, o corpo da seção.

Exemplo:

```
<section>
  <h2>Uma seção de página</h2>
  <p>...</p>
  <img ...>
</section>
```

É útil dividir um artigo longo em diferentes `sections`.



Não deve ser usado como um elemento de contêiner genérico. `div` é feito para isso.

## div

`div` é um elemento de contêiner genérico:

```
<div>
  ...
</div>
```

Geralmente, adicionamos um atributo `class` ou `id` a esse elemento, para permitir que ele seja estilizado usando CSS.

Usamos `div` em qualquer lugar onde precisamos de um contêiner, mas as tags existentes não são adequadas.

## Tags relacionadas à página

### nav

Essa tag é usada para criar a marcação que define a navegação da página. Para isso, normalmente, adicionamos uma lista `ul` ou `ol`:

```
<nav>
  <ol>
    <li><a href="/">Home</a></li>
    <li><a href="/blog">Blog</a></li>
  </ol>
</nav>
```

### aside

A tag `aside` é usada para adicionar um conteúdo relacionado ao conteúdo principal, por exemplo, uma caixa onde adicionamos uma citação ou uma barra lateral.

Exemplo:

```
<div>
  <p>Um texto...</p>
  <aside>
    <p>Uma citação...</p>
  </aside>
  <p>Outro texto...</p>
</div>
```

Usar `aside` é um sinal de que as coisas que ele contém não fazem parte do fluxo regular da seção em que está.

## header

A tag `header` representa uma parte da página que é a introdução. Ela pode, por exemplo, conter uma ou mais tags de título ( `h1` - `h6` ), a linha orientadora do artigo, uma imagem, entre outros.

```
<article>
  <header>
    <h1>Título do artigo</h1>
  </header>
  ...
</div>
```

## main

A tag `main` representa a parte principal de uma página:

```
<body>
  ....
  <main>
    <p>....</p>
  </main>
</body>
```

## footer

A tag `footer` é usada para determinar o rodapé de um artigo ou o rodapé da página:

```
<article>
  ....
  <footer>
    <p>Notas de rodapé...</p>
  </footer>
</div>
```

# FORMULÁRIOS

Os formulários são a maneira pela qual você pode interagir com uma página ou com uma aplicação criada com tecnologias da web.

Você tem um conjunto de controles e, ao enviar o formulário, seja com um clique em um botão de "enviar" ou de maneira programada, o navegador enviará os dados para o servidor.

Por padrão, esse envio de dados faz com que a página seja recarregada logo após a ação de envio, mas, usando JavaScript, você pode alterar esse comportamento (não explicaremos isso aqui).

Um formulário é criado usando a tag `form`:

```
<form>
...
</form>
```

Por padrão, os formulários são enviados usando o método de HTTP GET. Isso tem suas desvantagens e, em geral, você vai querer usar o método POST.

Você pode definir o formulário para usar POST quando enviado usando o atributo `method`:

```
<form method="POST">
...
</form>
```

O formulário é enviado, usando GET ou POST, para o mesmo URL em que reside.

Portanto, se o formulário estiver na página `https://flaviocopes.com/contacts`, pressionar o botão de "enviar" fará uma solicitação para esse mesmo URL.

Isso pode resultar em não acontecer nada.

Você precisa de algo do lado do servidor para lidar com a solicitação e, normalmente, "ouve" esses eventos de envio de formulário em um URL dedicado.

Você pode especificar o URL por meio do parâmetro `action`:

```
<form action="/novo-contato" method="POST">
...
</form>
```

Isso fará com que o navegador envie os dados do formulário usando POST para o URL `/novo-contato` na mesma origem.

Se a origem (protocolo + domínio + porta) for `https://flaviocopes.com` (porta 80 é o padrão), isso significa que os dados do formulário serão enviados para `https://flaviocopes.com/novo-contato`.

Eu falei sobre dados. Quais dados?

Os dados são fornecidos pelos usuários através do conjunto de controles que estão disponíveis na plataforma da web:

- caixas de entrada (em inglês, *input* – texto de linha única)
- áreas de texto (em inglês, *textarea* – texto de várias linhas)
- menus de seleção (em inglês, *select* – escolha uma opção em um menu suspenso)
- botões de opção (em inglês, *radio buttons* – escolha uma opção de uma lista sempre visível)
- caixas de seleção (em inglês, *checkboxes* – escolha zero, uma ou mais opções)

- uploads de arquivos
- e mais!

Vamos apresentar cada um deles na visão geral dos campos de formulário a seguir.

## Tag `input`

O campo `input` é um dos elementos de formulário mais usados. Também é um elemento muito versátil e pode mudar completamente o comportamento com base no atributo `type`.

O comportamento padrão é ser um controle de entrada de texto de linha única:

```
<input>
```

Isso é equivalente a usar:

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

Assim como todos os outros campos a seguir, você precisa dar um `name` (nome) ao campo para que seu conteúdo seja enviado ao servidor quando o formulário for enviado:

```
<input type="text" name="usuario">
```

O atributo `placeholder` é usado para que algum texto apareça, em cinza claro, quando o campo estiver vazio. Útil para adicionar uma dica ao usuário sobre o que digitar:

```
<input type="text" name="usuario" placeholder="Seu nome de usuário">
```

## Email

O uso de `type="email"` validará um e-mail do lado do client (no navegador) quanto à correção (correção semântica, não garantindo que o endereço de e-mail exista) antes de enviar.

```
<input type="email" name="e-mail" placeholder="Seu endereço de e-mail">
```

## Password

O uso de `type="password"` fará com que cada caractere inserido apareça como um asterisco (\*) ou ponto, útil para campos que hospedam uma senha.

```
<input type="password" name="senha" placeholder="Sua senha">
```

## Numbers

Você pode fazer com que um elemento de entrada aceite apenas números:

```
<input type="number" name="idade" placeholder="Sua idade">
```

Você pode especificar um valor mínimo e máximo aceito:

```
<input type="number" name="idade" placeholder="Sua idade" min="18" max="110">
```

O atributo `step` ajuda a identificar as etapas entre os diferentes valores. Por exemplo, ele aceita um valor entre 10 e 50, em etapas de 5:

```
<input type="number" name="um-numero" min="10" max="50" step="5">
```

## Campo oculto

Os campos podem ser ocultados do usuário. Eles ainda serão enviados ao servidor após o envio do formulário:

```
<input type="hidden" name="algun-campo-oculto" value="algun-valor">
```

Isso é comumente usado para armazenar valores como um token CSRF, usado para segurança e identificação de usuários, ou até mesmo para detectar robôs que enviam spam, usando técnicas especiais.

Também pode ser usado apenas para identificar um formulário e sua ação.

## Definindo um valor padrão

Todos esses campos aceitam um valor predefinido. Caso o usuário não o altere, este será o valor enviado ao servidor:

```
<input type="number" name="idade" value="18">
```

Se você definir um placeholder, esse valor aparecerá se o usuário limpar o valor do campo de entrada:

```
<input type="number" name="idade" placeholder="Sua idade" value="18">
```

## Enviando formulários

O campo `type="submit"` é um botão que, uma vez pressionado pelo usuário, envia o formulário:

```
<input type="submit">
```

O atributo `value` define o texto no botão, que, em sua ausência, mostra o texto "Submit" (em português, enviar):

```
<input type="submit" value="Clique aqui">
```

## Validação de formulários

Os navegadores fornecem a funcionalidade de validação do lado do client para formulários.

Você pode definir os campos conforme necessário, garantindo que eles sejam preenchidos, e impor um formato específico para a entrada de cada campo.

Vejamos as duas opções.

### Definir campo como obrigatório

O atributo `required` ajuda na validação. Se o campo não estiver definido, a validação do lado do client falha e o navegador não envia o formulário:

```
<input type="text" name="usuario" required>
```

### Aplicar um formato específico

Descrevi o campo `type="email"` acima. Ele valida automaticamente o endereço de e-mail de acordo com um formato definido na especificação.

No campo `type="number"`, mencionei o atributo `min` e `max` para limitar os valores inseridos em um intervalo.

Podemos fazer mais.

Podemos impor um formato específico em qualquer campo.

O atributo `pattern` permite definir uma expressão regular para validar o valor.

Recomendo a leitura do meu Guia de Expressões Regulares em [flaviocopes.com/javascript-regular-expressions/](https://flaviocopes.com/javascript-regular-expressions/) (texto em inglês).

Exemplo de padrão: `pattern="https://.*"`

```
<input type="text" name="usuario" pattern="[a-zA-Z]{8}">
```

## Outros campos

### Upload de arquivos

Você pode carregar arquivos do seu computador local e enviá-los para o servidor usando um elemento de entrada `type="file"`:

```
<input type="file" name="documentos-secretos">
```

É possível anexar vários arquivos:

```
<input type="file" name="documentos-secretos" multiple>
```

Você pode especificar um ou mais tipos de arquivo permitidos usando o atributo `accept`. Este aceita imagens:

```
<input type="file" name="documentos-secretos" accept="image/*">
```

Você pode usar um tipo MIME específico, como `application/json`, ou definir uma extensão de arquivo, como `.pdf`. Também é possível definir várias extensões de arquivo, assim:

```
<input type="file" name="documentos-secretos" accept=".jpg, .jpeg, .png">
```

## Botões

Os campos de entrada `type="button"` podem ser usados para adicionar botões adicionais ao formulário, que não são botões de envio:

```
<input type="button" value="Clique aqui">
```

Eles são usados para fazer algo de maneira programada, usando JavaScript.

Existe um campo especial renderizado como um botão, cuja ação especial é limpar todo o formulário e trazer os campos de volta ao estado inicial:

```
<input type="reset">
```

## Botões de opção

Os botões de opção são usados para criar um conjunto de opções. Quando uma das opções é selecionada, todas as outras são desabilitadas.

O nome em inglês, *radio buttons*, vem de rádios de carros antigos que tinham esse tipo de interface.

Você define um conjunto de entradas `type="radio"`, todas com o mesmo atributo `name` e diferentes atributos `value`:

```
<input type="radio" name="cor" value="amarelo">  
<input type="radio" name="cor" value="vermelho">  
<input type="radio" name="cor" value="azul">
```

Depois que o formulário for enviado, a propriedade de dados `cor` terá um único valor.

Há sempre um elemento selecionado. O primeiro item é o marcado por padrão.

Você pode definir o valor pré-selecionado usando o atributo `checked`. Você pode usá-lo apenas uma vez por grupo de botões de opção.

## Caixas de seleção

Essa entrada é semelhante aos botões de opção, mas permitem que vários valores sejam escolhidos, ou nenhum.

Você define um conjunto de entradas `type="checkbox"`, todas com o mesmo atributo `name` e diferentes atributos `value`:

```
<input type="checkbox" name="cor" value="amarelo">  
<input type="checkbox" name="cor" value="vermelho">  
<input type="checkbox" name="cor" value="azul">
```

Todas essas caixas de seleção estarão desmarcadas por padrão. Use o atributo `checked` para habilitá-las no carregamento da página.



Como esse campo de entrada permite vários valores, no envio do formulário, os valores serão enviados ao servidor como um array.

## Data e hora

Temos alguns tipos de entrada para aceitar valores de data.

O campo de entrada `type="date"` permite que o usuário insira uma data e mostra um seletor de data, se necessário:

```
<input type="date" name="aniversario">
```

O campo de entrada `type="time"` permite que o usuário insira um horário e mostra um seletor de horário, se necessário:

```
<input type="time" name="hora-de-buscar">
```

O campo de entrada `type="month"` permite que o usuário insira um mês e um ano:

```
<input type="month" name="selecionar-mes-do-lancamento">
```

O campo de entrada `type="week"` permite que o usuário insira uma semana e um ano:

```
<input type="week" name="selecionar-semana">
```

Todos esses campos permitem limitar o intervalo entre cada valor. Eu recomendo conferir a [MDN](#) para os pequenos detalhes sobre seu uso.

O campo `type="datetime-local"` permite que você escolha uma data e uma hora.

```
<input type="datetime-local" name="data-e-hora">
```

Aqui está uma página para testá-los: <https://codepen.io/flaviocopes/pen/ZdWQPm>

## Seletor de cores

Você pode permitir que os usuários escolham uma cor usando o elemento `type="color"`:

```
<input type="color" name="cor-do-carro">
```

Você define um valor padrão usando o atributo `value`:

```
<input type="color" name="cor-do-carro" value="#000000">
```

O navegador se encarregará de mostrar um seletor de cores para o usuário.

## Range

Este elemento de entrada mostra um elemento deslizante (em inglês, um *slider*). As pessoas podem usá-lo para passar de um valor inicial para um valor final:

```
<input type="range" name="idade" min="0" max="100" value="30">
```

Você pode fornecer passos, como opção:

```
<input type="range" name="idade" min="0" max="100" value="30" step="10">
```

## Telefone

O campo de entrada `type="tel"` é usado para inserir um número de telefone:

```
<input type="tel" name="numero-de-telefone">
```

O principal apelo para o uso de `tel` em vez de `text` são os celulares, pois o dispositivo pode optar por mostrar um teclado numérico.

Especifique um atributo `pattern` para validação adicional:

```
<input type="tel" pattern="[0-9]{3}-[0-9]{8}" name="numero-de-telefone">
```

## URL

O campo `type="url"` é usado para inserir um URL.

```
<input type="url" name="site-da-web">
```

Você pode validá-lo usando o atributo `pattern`:

```
<input type="url" name="site-da-web" pattern="https://.*">
```

## A tag `textarea`

O elemento `textarea` permite que os usuários insiram texto de várias linhas. Ao contrário do que acontece com `input`, essa tag requer uma tag de fechamento:

```
<textarea></textarea>
```

Você pode definir as dimensões usando CSS, ou também usando os atributos `rows` (linhas) e `cols` (colunas):

```
<textarea rows="20" cols="10"></textarea>
```

Assim como as outras tags de formulário, o atributo `name` determina o nome nos dados enviados ao servidor:

```
<textarea name="article"></textarea>
```

## A tag `select`

Essa tag é usada para criar um menu suspenso.

O usuário pode escolher uma das opções disponíveis.

Cada opção é criada usando a tag `option`. Você deve adicionar um nome à seleção e um valor para cada opção:

```
<select name="cor">
  <option value="vermelho">Vermelho</option>
  <option value="amarelo">Amarelo</option>
</select>
```

Você pode definir uma opção desativada:

```
<select name="cor">
  <option value="vermelho" disabled>Vermelho</option>
  <option value="amarelo">Amarelo</option>
</select>
```

Você pode definir uma opção vazia:

```
<select name="cor">
  <option value="">Nenhum</option>
  <option value="vermelho">Vermelho</option>
  <option value="amarelo">Amarelo</option>
</select>
```

As opções podem ser agrupadas usando a tag `optgroup`. Cada grupo de opções tem um atributo `label`:

```
<select name="cor">
  <optgroup label="Primaria">
    <option value="vermelho">Vermelho</option>
    <option value="amarelo">Amarelo</option>
    <option value="azul">Azul</option>
  </optgroup>
  <optgroup label="Outras">
    <option value="verde">Verde</option>
    <option value="rosa">Rosa</option>
  </optgroup>
</select>
```

# TABELAS

Nos primórdios da web, as tabelas eram uma parte muito importante da construção dos layouts.

Mais tarde, elas foram substituídas pelo CSS e seus recursos de layout. Hoje, temos ferramentas poderosas como CSS Flexbox e CSS Grid para construir layouts. As tabelas agora são usadas apenas para, adivinhe, construir tabelas!

## Tag `table`

Você define uma tabela usando a tag `table`:

```
<table>

</table>
```

Dentro da tabela vamos definir os dados. Raciocinamos em termos de linhas, o que significa que adicionamos linhas em uma tabela (não colunas). Vamos definir colunas dentro de uma linha.

## Linhas (rows)

Uma linha é adicionada usando a tag `tr`. Essa é a única coisa que podemos adicionar em um elemento `table`:

```
<table>
  <tr></tr>
  <tr></tr>
  <tr></tr>
</table>
```

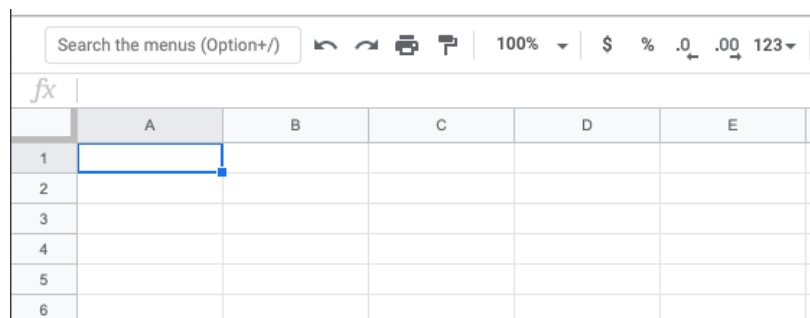
Essa é uma tabela com 3 linhas.

A primeira linha *pode* assumir a função de cabeçalho.

## Cabeçalho de colunas

O cabeçalho da tabela contém o nome de uma coluna, normalmente em negrito.

Pense em um documento do Excel/Planilhas do Google. O cabeçalho `A-B-C-D...` que fica acima.



	A	B	C	D	E
1					
2					
3					
4					
5					
6					

Definimos o cabeçalho usando a tag `th`:

```
<table>
  <tr>
    <th>Coluna 1</th>
    <th>Coluna 2</th>
    <th>Coluna 3</th>
  </tr>
  <tr></tr>
  <tr></tr>
</table>
```

## O conteúdo da tabela

O conteúdo da tabela é definido usando tags `td`, dentro dos outros elementos `tr`:

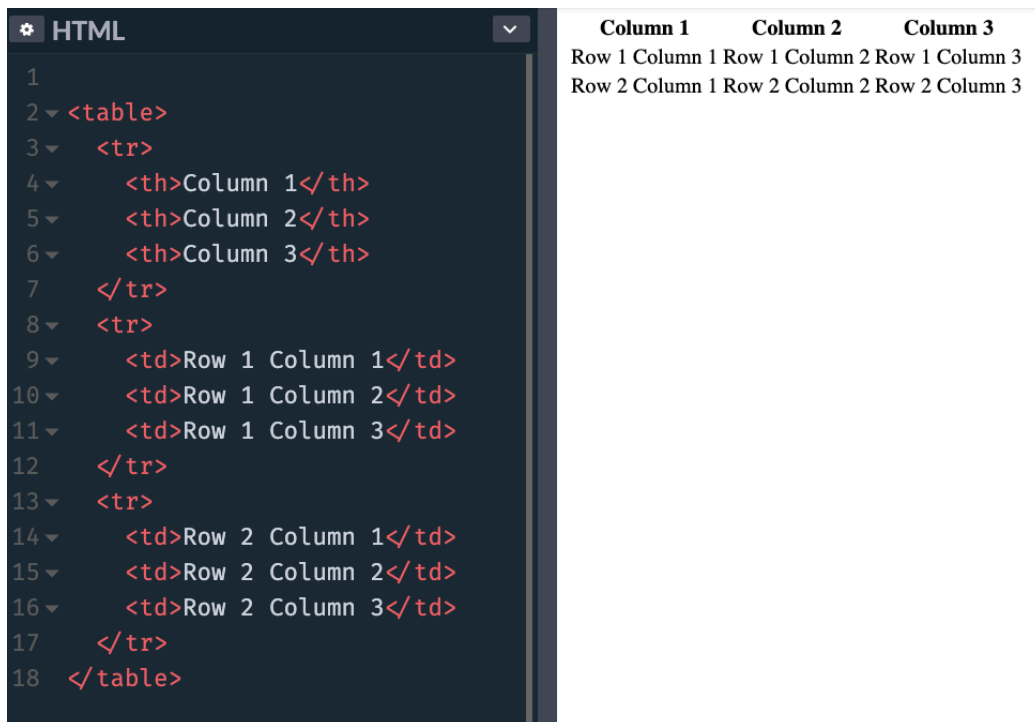
```
<table>
  <tr>
    <th>Coluna 1</th>
    <th>Coluna 2</th>
```

```

<th>Coluna 3</th>
</tr>
<tr>
<td>Linha 1 Coluna 1</td>
<td>Linha 1 Coluna 2</td>
<td>Linha 1 Coluna 3</td>
</tr>
<tr>
<td>Linha 2 Coluna 1</td>
<td>Linha 2 Coluna 2</td>
<td>Linha 2 Coluna 3</td>
</tr>
</table>

```

É assim que os navegadores renderizam tabelas se você não adicionar nenhum estilo CSS:



The image shows a code editor on the left with HTML code for a table. The code defines a table with 3 columns and 2 rows. The first row contains three header cells with the text 'Column 1', 'Column 2', and 'Column 3'. The second row contains three data cells with the text 'Row 1 Column 1', 'Row 1 Column 2', and 'Row 1 Column 3'. The third row contains three data cells with the text 'Row 2 Column 1', 'Row 2 Column 2', and 'Row 2 Column 3'. To the right of the code editor, the rendered table is displayed. It has three columns and two rows. The first row has three header cells with the text 'Column 1', 'Column 2', and 'Column 3'. The second row has three data cells with the text 'Row 1 Column 1', 'Row 1 Column 2', and 'Row 1 Column 3'. The third row has three data cells with the text 'Row 2 Column 1', 'Row 2 Column 2', and 'Row 2 Column 3'.

Column 1	Column 2	Column 3
Row 1 Column 1	Row 1 Column 2	Row 1 Column 3
Row 2 Column 1	Row 2 Column 2	Row 2 Column 3

Adicionamos, então, este CSS:

```

th, td {
padding: 10px;
border: 1px solid #333;
}

```

Isso faz com que a tabela pareça mais com uma tabela propriamente dita:

Column 1	Column 2	Column 3
Row 1 Column 1	Row 1 Column 2	Row 1 Column 3
Row 2 Column 1	Row 2 Column 2	Row 2 Column 3

## Expandir colunas e linhas

Uma linha pode abranger 2 ou mais colunas, usando o atributo `colspan`:

```
<table>
  <tr>
    <th>Coluna 1</th>
    <th>Coluna 2</th>
    <th>Coluna 3</th>
  </tr>
  <tr>
    <td colspan="2">Linha 1 Colunas 1 e 2</td>
    <td>Linha 1 Coluna 3</td>
  </tr>
  <tr>
    <td colspan="3">Linha 2 Colunas 1 a 3</td>
  </tr>
</table>
```

Column 1	Column 2	Column 3
Row 1 Columns 1-2		Row 1 Column 3
Row 2 Columns 1-3		

Também é possível abranger 2 ou mais linhas, usando o atributo `rowspan`:

```
<table>
  <tr>
    <th>Coluna 1</th>
    <th>Coluna 2</th>
    <th>Coluna 3</th>
  </tr>
  <tr>
    <td colspan="2" rowspan="2">Linhas 1 e 2 Colunas 1 e 2</td>
    <td>Linha 1 Coluna 3</td>
  </tr>
  <tr>
    <td>Linha 2 Coluna 3</td>
  </tr>
</table>
```

Column 1	Column 2	Column 3
Rows 1-2 Columns 1-2		Row 1 Column 3
		Row 2 Column 3

## Cabeçalho das linhas

Anteriormente, eu expliquei como você pode criar os cabeçalhos para as colunas, usando a tag `th` dentro da primeira tag `tr` da tabela.

Você pode adicionar uma tag `th` como o primeiro elemento dentro de um `tr` que não é o primeiro `tr` da tabela, e assim criar cabeçalhos de linha:

```
<table>
  <tr>
    <th></th>
    <th>Coluna 2</th>
    <th>Coluna 3</th>
  </tr>
  <tr>
    <th>Linha 1</th>
    <td>Coluna 2</td>
    <td>Coluna 3</td>
  </tr>
  <tr>
    <th>Linha 2</th>
    <td>Coluna 2</td>
    <td>Coluna 3</td>
  </tr>
</table>
```

	Column 2	Column 3
Row 1	Col 2	Col 3
Row 1	Col 2	Col 3

## Mais tags para organizar a tabela

Você pode adicionar mais três tags em uma tabela, para deixá-la mais organizada.

Isso funciona melhor em tabelas grandes e para definir corretamente cabeçalho e rodapé.

Essas tags são:

- `thead`
- `tbody`
- `tfoot`

Elas envolvem as tags `tr` para definir claramente as diferentes seções da tabela. Aqui está um exemplo:

```
<table>
  <thead>
```



```

<tr>
  <th></th>
  <th>Coluna 2</th>
  <th>Coluna 3</th>
</tr>
</thead>
<tbody>
  <tr>
    <th>Linha 1</th>
    <td>Coluna 2</td>
    <td>Coluna 3</td>
  </tr>
  <tr>
    <th>Linha 2</th>
    <td>Coluna 2</td>
    <td>Coluna 3</td>
  </tr>
</tbody>
<tfoot>
  <tr>
    <td></td>
    <td>Rodapé da coluna 1</td>
    <td>Rodapé da coluna 2</td>
  </tr>
</tfoot>
</table>

```

	Column 2	Column 3
Row 1	Col 2	Col 3
Row 2	Col 2	Col 3
	Footer of Col 1	Footer of Col 2

## Legenda de tabelas

As tabelas devem ter uma tag de legenda, `<caption>`, que descreva seu conteúdo. Essa tag deve ser colocada imediatamente após a tag de abertura da tabela:

```

<table>
  <caption>Idade dos cães</caption>
  <tr>
    <th>Cão</th>
    <th>Idade</th>
  </tr>
  <tr>
    <td>Roger</td>
    <td>7</td>
  </tr>
</table>

```

# TAGS DE MULTIMÍDIA: AUDIO E VIDEO

Nesta seção, quero mostrar as tags `audio` e `video`.

## Tag `audio`

Essa tag permite que você incorpore conteúdo de áudio em suas páginas de HTML.

Este elemento possibilita transmissão de áudio, tanto através de microfone via `getUserMedia()`, ou por reprodução de uma fonte de áudio que você referencia usando o atributo `src`:

```
<audio src="arquivo.mp3">
```

Por padrão, o navegador não mostra nenhum controle para esse elemento. O que significa que o áudio será reproduzido apenas se configurado para reprodução automática (mais sobre isso posteriormente) e o usuário não pode ver como pará-lo, controlar o volume ou mover-se pela faixa.

Para mostrar os controles internos, você pode adicionar o atributo `controls`:

```
<audio src="arquivo.mp3" controls>
```

Os controles podem ter uma capa personalizada.

Você pode especificar o tipo MIME do arquivo de áudio usando o atributo `type`. Se não estiver definido, o navegador tentará determiná-lo automaticamente:

```
<audio src="arquivo.mp3" controls type="audio/mpeg">
```

Um arquivo de áudio, por padrão, não é reproduzido automaticamente. Adicione o atributo `autoplay` para reproduzir o áudio automaticamente:

```
<audio src="arquivo.mp3" controls autoplay>
```

Observação: navegadores de dispositivos móveis não permitem reprodução automática.

O atributo `loop` reinicia a reprodução de áudio para 0:00, se definido; caso contrário, se não estiver presente, o áudio para no final do arquivo:

```
<audio src="arquivo.mp3" controls autoplay loop>
```

Você também pode reproduzir um arquivo de áudio silenciado usando o atributo `muted` (não tenho certeza de qual é a utilidade disso):

```
<audio src="arquivo.mp3" controls autoplay loop muted>
```

Usando JavaScript, você pode "ouvir" vários eventos acontecendo em um elemento `audio`, sendo os mais básicos:

- `play` quando o arquivo começar a tocar
- `pause` quando a reprodução de áudio for pausada
- `playing` quando o áudio for retomado depois de uma pausa
- `ended` quando o final do arquivo de áudio for atingido

## Tag `video`

Essa tag permite que você incorpore conteúdo de vídeo em suas páginas de HTML.

Este elemento pode transmitir vídeo, usando uma webcam via `getUserMedia()` ou **WebRTC**, ou pode reproduzir uma fonte de vídeo que você referencia usando o atributo `src`:

```
<video src="arquivo.mp4">
```

Por padrão, o navegador não mostra nenhum controle para esse elemento, apenas o vídeo.

Isso significa que o vídeo será reproduzido apenas se configurado para reprodução automática (mais sobre isso mais tarde) e o usuário não pode ver como pará-lo, pausá-lo, controlar o volume ou pular para uma posição específica no vídeo.

Para mostrar os controles internos, você pode adicionar o atributo `controls`:

```
<video src="arquivo.mp4" controls>
```

Os controles podem ter uma capa personalizada.

Você pode especificar o tipo MIME do arquivo de vídeo usando o atributo `type`. Se não estiver definido, o navegador tentará determiná-lo automaticamente:

```
<video src="arquivo.mp4" controls type="video/mp4">
```

Um arquivo de vídeo por padrão não é reproduzido automaticamente. Adicione o atributo `autoplay` para reproduzir o vídeo automaticamente:

```
<video src="file.mp4" controls autoplay>
```

Alguns navegadores também exigem o atributo `muted` para reprodução automática. O vídeo é reproduzido automaticamente apenas se silenciado:

```
<audio src="arquivo.mp3" controls autoplay muted>
```

O atributo `loop` reinicia a reprodução do vídeo em 0:00, se definido; caso contrário, se não estiver presente, o vídeo para no final do arquivo:

```
<video src="arquivo.mp4" controls autoplay loop>
```

Você pode definir uma imagem para ser a imagem de capa:

```
<video src="arquivo.mp4" poster="imagem.png">
```

Se não estiver presente, o navegador exibirá o primeiro quadro do vídeo assim que estiver disponível.

Você pode definir os atributos `width` e `height` para definir o espaço que o elemento ocupará para que o navegador possa considerá-lo e não alterar o layout quando finalmente for carregado. Recebe um valor numérico, expresso em pixels.

Usando JavaScript, você pode "ouvir" vários eventos acontecendo em um elemento `video`, sendo os mais básicos:

- `play` quando o arquivo começar a tocar
- `pause` quando o vídeo for pausado
- `playing` quando o vídeo for retomado depois de uma pausa
- `ended` quando o final do arquivo de vídeo for atingido

## IFRAMES

A tag `iframe` nos permite incorporar conteúdo de outras origens (outros sites) em nossa página da web.

Tecnicamente, um *iframe* cria um contexto de navegação aninhado. Isso significa que qualquer coisa no *iframe* não interfere na página mãe e vice-versa. O JavaScript e o CSS não "vazam" de/para *iframes*.

Muitos sites usam *iframes* para realizar várias coisas. Você pode estar familiarizado com o Codepen, o Glitch ou outros sites que permitem programar em uma parte da página e ver o resultado em uma caixa. Isso é um *iframe*.

Você cria um desta forma:

```
<iframe src="pagina.html"></iframe>
```

Você também pode carregar um URL absoluto:

```
<iframe src="https://site.com/pagina.html"></iframe>
```

Você pode definir um conjunto de parâmetros de largura e altura (ou defini-los usando CSS). Caso contrário, o *iframe* usará os padrões, uma caixa de 300x150 pixels:

```
<iframe src="pagina.html" width="800" height="400"></iframe>
```

## Srcdoc

O atributo `srcdoc` permite especificar algum HTML embutido para mostrar. É uma alternativa ao `src`, mas recente e não suportada no Internet Explorer nem no Edge, versão 18 e anteriores:

```
<iframe srcdoc="<p>Eu tenho um cão bonzinho</p>"></iframe>
```

## Sandbox

O atributo `sandbox` permite limitar as operações permitidas nos *iframes*.

Se a omitirmos, tudo é permitido:

```
<iframe src="pagina.html"></iframe>
```

Se definirmos como "", nada é permitido:

```
<iframe src="page.html" sandbox=""></iframe>
```

Podemos selecionar o que permitir adicionando opções no atributo `sandbox`. Você pode permitir várias ações adicionando um espaço entre elas. Aqui está uma lista incompleta das opções que você pode usar:

- `allow-forms`: permitir o envio de formulários
- `allow-modals`: permitir a abertura de janelas modais, incluindo chamar `alert()` em JavaScript
- `allow-orientation-lock`: permitir bloquear a orientação da tela
- `allow-popups`: permitir pop-ups, usando os links `window.open()` e `target="_blank"`
- `allow-same-origin`: tratar o recurso que está sendo carregado como sendo da mesma origem
- `allow-scripts`: permitir que o *iframe* carregado execute scripts (mas não crie pop-ups).
- `allow-top-navigation`: dar acesso para o *iframe* ao contexto de navegação de nível superior.

## Allow

Atualmente experimental e suportado apenas por navegadores baseados em Chromium, este é o futuro do compartilhamento de recursos entre a janela pai e o *iframe*.

É semelhante ao atributo `sandbox`, mas nos auxilia a permitir recursos específicos, incluindo:

- `accelerometer`: dá acesso à interface Sensors API Accelerometer
- `ambient-light-sensor`: dá acesso à interface Sensors API AmbientLightSensor
- `autoplay`: permite reproduzir automaticamente arquivos de vídeo e áudio
- `camera`: permite acessar a câmera da API getUserMedia
- `display-capture`: permite acessar o conteúdo da tela usando a API getDisplayMedia
- `fullscreen`: permite acessar o modo de tela cheia
- `geolocation`: permite acessar a API de geolocalização
- `gyroscope`: dá acesso à interface Sensors API Gyroscope
- `magnetometer`: dá acesso à interface Sensors API Magnetometer
- `microphone`: dá acesso ao microfone do dispositivo usando a API getUserMedia
- `midi`: permite acesso à API Web MIDI
- `payment`: dá acesso à API de solicitação de pagamento
- `speaker`: permite o acesso à reprodução de áudio através dos alto-falantes do dispositivo
- `usb`: dá acesso à API WebUSB.
- `vibrate`: dá acesso à API de vibração
- `vr`: dá acesso à API WebVR

## Referrer

Ao carregar um *iframe*, o navegador envia informações importantes sobre quem o está carregando no cabeçalho `Referer` (atenção para o `r` único, um erro de digitação na língua inglesa com o qual devemos conviver).

O erro de ortografia do "referrer" originou-se na proposta original do cientista da computação Phillip Hallam-Baker para incorporar o campo na especificação do HTTP. O erro de ortografia foi definido no momento de sua incorporação no documento de padrões de solicitação de comentários RFC 1945.

O atributo `referrerpolicy` nos permite definir o referenciador para enviar ao *iframe* ao carregá-lo. O referenciador (em inglês, *referrer*) é um cabeçalho do HTTP que permite que a página saiba quem a está carregando. Estes são os valores permitidos:

- `no-referrer-when-downgrade`: é o padrão e não envia o referenciador quando a página atual é carregada por HTTPS e o *iframe* é carregado no protocolo HTTP
- `no-referrer`: não envia o cabeçalho do referenciador
- `origin`: o referenciador é enviado e contém apenas a origem (porta, protocolo, domínio), não a origem + caminho – que é o padrão
- `origin-when-cross-origin`: ao carregar da mesma origem (porta, protocolo, domínio) no *iframe*, o referenciador é enviado em sua forma completa (origem + caminho). Caso contrário, apenas a origem é enviada
- `same-origin`: o referenciador é enviado apenas ao carregar da mesma origem (porta, protocolo, domínio) no *iframe*
- `strict-origin`: envia a origem como o referenciador se a página atual for carregada por HTTPS e o *iframe* também carregar no protocolo HTTPS. Não envia nada se o *iframe* for carregado por HTTP
- `strict-origin-when-cross-origin`: envia a origem + caminho como referenciador ao trabalhar na mesma origem. Envia a origem como o referenciador se a página atual for carregada por HTTPS e o *iframe* também carregar no protocolo HTTPS. Não envia nada se o *iframe* for carregado por HTTP
- `unsafe-url`: envia a origem + caminho como referenciador mesmo ao carregar recursos de HTTP e a página atual é carregada por HTTPS

## IMAGENS

As imagens podem ser exibidas usando a tag `img`.

Essa tag aceita um atributo `src`, que usamos para definir a fonte da imagem:

```

```

Podemos usar um amplo conjunto de imagens. Os tipos mais comuns são PNG, JPEG, GIF, SVG e, mais recentemente, WebP.

O padrão HTML requer que um atributo `alt` esteja presente para descrever a imagem. Ele é usado por leitores de tela e por bots de mecanismos de pesquisa:

```

```

Você pode configurar os atributos de largura, `width`, e altura, `height`, para definir o espaço que o elemento ocupará, de modo que o navegador possa considerá-lo sem alterar o layout quando a imagem estiver totalmente carregada. Eles recebem um valor numérico, expresso em pixels.

```

```

## Tag `figure`

A tag `figure` é frequentemente usada juntamente com a tag `img`.

`figure` é uma tag semântica, frequentemente usada quando você deseja exibir uma imagem com uma legenda. Você a usa assim:

```
<figure>
  
  <figcaption>Um cãozinho dócil</figcaption>
</figure>
```

A tag `figcaption` envolve o texto da legenda.

## Imagens responsivas usando `srcset`

O atributo `srcset` permite definir imagens responsivas que o navegador pode usar dependendo da densidade de pixels ou da largura da janela, de acordo com suas preferências. Desse modo, o navegador pode baixar apenas os recursos necessários para renderizar a página, sem baixar uma imagem maior se estiver em um dispositivo móvel, por exemplo.

Aqui está um exemplo. Nele, fornecemos 4 imagens adicionais para 4 tamanhos de tela diferentes:

```

```

No `srcset` usamos a medida `w` para indicar a largura da janela.

Como fazemos isso, também precisamos usar o atributo de tamanhos, `sizes`:



```

```

Neste exemplo, a string `(max-width: 500px) 100vw, (max-width: 900px) 50vw, 800px` no atributo `sizes` descreve o tamanho da imagem em relação à janela de visualização, com várias condições separadas por ponto e vírgula.

A condição de mídia `max-width: 500px` define o tamanho da imagem em correlação com a largura da janela de visualização. Resumindo, se o tamanho da janela for menor do que 500px, a imagem é renderizada em 100% do tamanho da janela.

Se o tamanho da janela for maior que 500px, mas menor que 900px, a imagem é renderizada em 50% do tamanho da janela.

Se for ainda maior, a imagem é renderizada em 800px.

A unidade de medida `vw` pode ser nova para você. Resumindo, podemos dizer que 1 `vw` é 1% da largura da janela, então `100vw` é 100% da largura da janela.

Um site útil para gerar o `srcset` e imagens progressivamente menores é <https://responsivebreakpoints.com/>.

## Tag `picture`

O HTML também nos dá a tag `picture`, que faz um trabalho muito semelhante ao `srcset`. As diferenças são muito sutis.

Você usa `picture` quando, em vez de apenas servir uma versão menor de um arquivo, deseja alterá-lo completamente ou veicular um formato de imagem diferente.

O melhor uso que encontrei foi com uma imagem WebP, que é um formato ainda não amplamente suportado. Na tag `picture`, você especifica uma lista de imagens e elas serão usadas em ordem. Portanto, no próximo exemplo, os navegadores que suportam WebP usarão a primeira imagem ou retornarão para JPG, do contrário:

```
<picture>
  <source type="image/webp" srcset="imagem.webp" >
  
</picture>
```

A tag `source` define um (ou mais) formatos para as imagens. A tag `img` é o substituto caso o navegador seja muito antigo e não dê suporte à tag `picture`.

Na tag `source` dentro da imagem, você pode adicionar um atributo `media` para definir consultas de mídia.

O exemplo a seguir funciona como o exemplo acima com `srcset`:

```
<picture>
  <source media="(min-width: 500w)" srcset="cao-500.png" sizes="100vw">
  <source media="(min-width: 800w)" srcset="cao-800.png" sizes="100vw">
  <source media="(min-width: 1000w)" srcset="cao-1000.png" sizes="800px">
  <source media="(min-width: 1400w)" srcset="cao-1400.png" sizes="800px">
  
</picture>
```

Esse, porém, não é o seu caso de uso, porque, como você pode ver, é muito mais detalhado.

A tag `picture` é recente, mas agora é suportada por todos os principais navegadores, exceto Opera Mini e IE (todas as versões).

## ACESSIBILIDADE

É importante que projetemos nosso HTML com acessibilidade em mente.

Fazer um HTML acessível significa que pessoas com deficiência podem usar a web. Existem usuários totalmente cegos ou com deficiência visual, pessoas com problemas de perda auditiva e uma infinidade de outras deficiências diferentes.

Infelizmente, este tópico não tem a atenção de que precisa, talvez por não ser considerado tão "legal" quanto outros tópicos.

O que acontece se uma pessoa não puder ver sua página, mas ainda quiser consumir seu conteúdo? Primeiro, como eles fazem isso? Eles não podem usar o mouse. Eles usam algo chamado **leitor de tela**. Você não precisa imaginar isso. Você pode experimentar um agora: o Google fornece a extensão ChromeVox Chrome gratuita. A acessibilidade também deve cuidar de permitir que as ferramentas selecionem facilmente os elementos ou naveguem pelas páginas.

Páginas da web e aplicações da web nem sempre são construídas com acessibilidade como um de seus primeiros objetivos. Talvez a versão 1 seja lançada sem acessibilidade, mas é possível tornar uma página da web acessível depois de feita. *É melhor fazer isso antes*, mas nunca é **tarde demais**.

Esse é um assunto importante e, no meu país, os sites criados pelo governo ou por outras organizações públicas devem ter acessibilidade.

O que significa tornar um HTML acessível? Deixe-me ilustrar as principais coisas que você precisa pensar.

Observação: existem várias outras coisas nas quais devemos prestar atenção e que podem estar no tópico do CSS, como cores, contraste e fontes. Também é importante saber como tornar as imagens SVG acessíveis (texto em inglês). Eu, no entanto, não falo sobre esses assuntos aqui.

# Use semântica em seu HTML

O HTML semântico é muito importante e é uma das principais coisas das quais você precisa cuidar. Deixe-me ilustrar alguns cenários comuns.

É importante usar a estrutura correta para as tags de títulos. O mais importante é o `h1`. Você usa números mais altos para os menos importantes, mas todos os títulos de mesmo nível devem ter o mesmo significado (pense nisso como uma estrutura de árvore)

```
h1
  h2
    h3
  h2
  h2
    h3
      h4
```

Use `strong` e `em` em vez de `b` e `i`. Visualmente, eles parecem iguais, mas os 2 primeiros têm mais significado associado a eles. `b` e `i` são elementos mais visuais.

Listas são importantes. Um leitor de tela pode detectar uma lista e fornecer uma visão geral, permitindo que o usuário escolha entrar na lista ou não.

Tabelas devem ter uma tag de legenda, `caption`, que descreva seu conteúdo:

```
<table>
  <caption>Idade dos cães</caption>
  <tr>
    <th>Cão</th>
    <th>Idade</th>
  </tr>
  <tr>
    <td>Roger</td>
    <td>7</td>
  </tr>
</table>
```

## Use atributos `alt` para imagens

Todas as imagens devem ter uma tag `alt` descrevendo o conteúdo da imagem. Não é apenas uma boa prática. É exigido pelo padrão HTML e seu HTML sem ele não é validado.

```

```

Também é bom para os mecanismos de pesquisa, se isso for um incentivo para você adicioná-lo.

## Use o atributo `role`

O atributo `role` permite atribuir funções específicas a vários elementos em sua página.

Você pode atribuir muitas funções diferentes: complementar `complementary`, lista `list`, item de lista `listitem`, principal `main`, navegação `navigation`, região `region`, aba `tab`, alerta `alert`, aplicação `application`, artigo `article`, banner, botão `button`, célula `cell`, `checkbox`, informação do conteúdo `contentinfo`, diálogo `dialog`, documento `document`, `feed`, imagem/figura `figure`, formulário `form`, grade `grid`, célula de grade `gridcell`, cabeçalho `heading`, imagem `img`, caixa de lista `listbox`, linha `row`, grupo de linhas `rowgroup`, busca `search`, troca `switch`, tabela `table`, painel de abas `tabpanel`, caixa de texto `textbox` e temporizador `timer`.

São muitas coisas. Para a referência completa de cada um deles, use este [link da MDN](#). Você, no entanto, não precisa atribuir uma função a cada elemento na página. Leitores de tela podem inferir da tag HTML na maioria dos casos. Por exemplo, você não precisa adicionar uma tag de função em tags semânticas, como `nav`, `button` ou `form`.

Vamos pegar o exemplo da tag `nav`. Você pode usá-la para definir a navegação da página assim:

```
<nav>
  <ul>
    <li><a href="/">Início</a></li>
    <li><a href="/blog">Blog</a></li>
  </ul>
</nav>
```

Se você fosse *forçado* a usar uma tag `div` em vez de `nav`, você usaria a função de `navigation`:

```
<div role="navigation">
  <ul>
    <li><a href="/">Início</a></li>
    <li><a href="/blog">Blog</a></li>
  </ul>
</div>
```

Então, aqui, você tem um exemplo prático: `role` é usado para atribuir um valor significativo quando a tag ainda não transmite o significado.

## Use o atributo `tabindex`

O atributo `tabindex` permite alterar a ordem de como pressionar a tecla Tab seleciona os elementos "selecionáveis". Por padrão, apenas links e elementos de formulário são "selecionáveis" por navegação usando a tecla Tab (e você não precisa definir `tabindex` nesses casos).

Adicionar `tabindex="0"` torna um elemento selecionável:

```
<div tabindex="0">
...
</div>
```

Usar `tabindex="-1"` remove um elemento dessa navegação baseada em guias e pode ser bastante útil.

## Use o atributo `aria`

ARIA é um acrônimo em inglês para Accessible Rich Internet Applications (Aplicações Avançadas de Internet Acessíveis) e define a semântica que pode ser aplicada aos elementos.

### `aria-label`

Este atributo é usado para adicionar uma string para descrever um elemento.

Exemplo:

```
<p aria-label="A descrição do produto">...</p>
```

Eu uso esse atributo na barra lateral do meu blog, onde tenho uma caixa de entrada para pesquisa sem um rótulo explícito, pois possuí um atributo `placeholder`.

### `aria-labelledby`

Esse atributo define uma correlação entre o elemento atual e aquele que o rotula.

Se você sabe como um elemento `input` pode ser associado a um elemento `label`, é semelhante.

Passamos o id do item que descreve o elemento atual.

Exemplo:

```
<h3 id="description">A descrição do produto</h3>

<p aria-labelledby="description">
  ...
</p>
```

### `aria-describedby`

Esse atributo permite associar um elemento a outro elemento que serve como descrição.

Exemplo:

```
<button aria-describedby="payNowDescription">Pague agora</button>

<div id="payNowDescription">Clicar no botão enviará você para o formulário do Stripe!</div>
```

## Use `aria-hidden` para esconder

Eu gosto de um design minimalista em meus sites. Meu blog, por exemplo, é basicamente apenas conteúdo, com alguns links na barra lateral. Algumas coisas na barra lateral, porém, são apenas elementos visuais que não contribuem para a experiência de uma pessoa que não pode ver a página, como a imagem do meu logotipo ou o seletor de tema escuro/claro.

Adicionar o atributo `aria-hidden="true"` fará com que os leitores de tela ignorem esse elemento.

## Quer saber mais?

Esta é apenas uma introdução ao tema. Para saber mais, recomendo estes recursos (em inglês):

- <https://www.w3.org/TR/WCAG20/>
- <https://webaim.org>
- <https://developers.google.com/web/fundamentals/accessibility/>

---

Você chegou ao final do Manual de HTML. parabéns!

---