

This page was translated from English by the community.
Learn more and join the MDN Web Docs community.

HTML básico

HTML (Linguagem de Marcação de Hipertexto) é o código que você usa para estruturar uma página web e seu conteúdo. Por exemplo, o conteúdo pode ser estruturado em parágrafos, em uma lista com marcadores ou usando imagens e tabelas. Como o título sugere, este artigo fornecerá uma compreensão básica do HTML e suas funções.

Então, o que é HTML?

HTML não é uma linguagem de programação; é uma *linguagem de marcação*, usada para definir a estrutura do seu conteúdo. HTML consiste de uma série de elementos, que você usa para delimitar ou agrupar diferentes partes do conteúdo para que ele apareça ou atue de determinada maneira. As tags anexas podem transformar uma palavra ou imagem num hiperlink, pode colocar palavras em itálico, pode aumentar ou diminuir a fonte e assim por diante. Por exemplo, veja a seguinte linha de conteúdo:

Meu gatinho é muito mal humorado

Se quiséssemos que a linha permanecesse por si só, poderíamos especificar que é um parágrafo colocando-a em uma tag de parágrafo:

<**pp**

Anatomia de um elemento HTML

Vamos explorar esse parágrafo mais profundamente.

 Imagem mostrando como funciona a tag P

As principais partes de um elemento são:

1. **A tag de abertura:** Consiste no nome do elemento (no caso, p), envolvido em **parênteses angulares** de abertura e fechamento. Isso demonstra onde o elemento começa, ou onde seu efeito se inicia — nesse caso, onde é o começo do parágrafo.
2. **A tag de fechamento:** Isso é a mesma coisa que a tag de abertura, exceto que inclui uma barra antes do nome do elemento. Isso demonstra onde o elemento acaba — nesse caso, onde é o fim do parágrafo. Esquecer de incluir uma tag de fechamento é um dos erros mais comuns de iniciantes e pode levar a resultados estranhos.
3. **O conteúdo:** Esse é o conteúdo do elemento, que nesse caso é apenas texto.
4. **O elemento:** A tag de abertura, a de fechamento, e o

conteúdo formam o elemento.

Elementos também podem ter atributos, que parecem assim:

The diagram shows a dark grey rectangular box containing an HTML snippet. At the top center, the word "Attribute" is written in white. A thin black bracket is positioned under the word "Attribute", extending downwards to point to the "class" attribute in the code below. The code is written in white and consists of a single line: <p class="editor-note">My cat is very grumpy</p>. The "class" keyword and its value "editor-note" are highlighted in yellow.

```
<p class="editor-note">My cat is very grumpy</p>
```

Atributos contém informação extra sobre o elemento que você não quer que apareça no conteúdo real. Aqui, `class` é o nome do atributo e `editor-note` é o valor do atributo. O atributo `class` permite que você forneça ao elemento um identificador que possa ser usado posteriormente para aplicar ao elemento informações de estilo e outras coisas.

Um atributo sempre deve ter:

1. Um espaço entre ele e o nome do elemento (ou o atributo anterior, se o elemento já tiver um).
2. O nome do atributo, seguido por um sinal de igual.
3. Aspas de abertura e fechamento, envolvendo todo o valor do atributo.

Nota: Valores de atributos simples que não contém espaço em branco ASCII (ou qualquer um dos caracteres " ' ` = < >) podem permanecer sem aspas, mas é recomendável colocar em todos os valores de atributos, pois isso torna o código mais consistente e compreensível.

Aninhando elementos

Você pode colocar elementos dentro de outros elementos também — isso é chamado de **aninhamento**. Se quiséssemos afirmar que nosso gato é **muito** mal-humorado, poderíamos envolver a palavra "muito" em um elemento , o que significa que a palavra deve ser fortemente enfatizada:

```
| <p>Meu gatinho é <strong>muito</strong> m... h
```

Você precisa, no entanto, certificar-se de que seus elementos estejam adequadamente aninhados. No exemplo acima, abrimos primeiro o elemento <p>, depois o elemento ; portanto, temos que fechar primeiro o elemento , depois o elemento <p>. O código abaixo está incorreto:

```
| <p>Meu gatinho é <strong>muito mal hum... .c
```

Os elementos precisam ser abertos e fechados corretamente para que eles estejam claramente visíveis dentro ou fora um do outro. Se eles se sobreponerem conforme mostrado acima, seu navegador tentará adivinhar o que você estava tentando dizer, o que pode levar a resultados inesperados. Então não faça isso!

Elementos vazios

Alguns elementos não possuem conteúdo e são chamados

de **elementos vazios**. Considere o elemento `` que temos na nossa página HTML:

```
|
```

Ele contém dois atributos, mas não há tag `` de fechamento, e não há conteúdo interno. Isso acontece porque um elemento de imagem não envolve conteúdo para ter efeito em si mesmo. Sua proposta é incorporar uma imagem na página HTML no lugar que o código aparece.

Anatomia de um documento HTML

Isso resume o básico dos elementos HTML individuais, mas eles não são úteis por si só. Agora vamos ver como elementos individuais são combinados para formar uma página HTML inteira. Vamos visitar novamente os códigos que colocamos no exemplo de `index.html` (que vimos no artigo [Lidando com arquivos](#)):

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Minha página de teste</title>
  </head>
  <body>
    
  </body>
</html>
```

Aqui nós temos:

- `<!DOCTYPE html>` — o doctype. É a parte inicial obrigatória do documento. Nas névoas do tempo, quando o HTML era novo (por volta de 1991/2), doctypes eram criados para agir como links para um conjunto de regras que a página HTML tinha que seguir para ser considerada um bom HTML, o que poderia significar checagem automática de erros e outras coisas úteis. No entanto, atualmente, eles não fazem muito sentido e são basicamente necessários apenas para garantir que o documento se comporte corretamente. Isso é tudo que você precisa saber agora.
- `<html></html>` — o elemento [`<html>`](#). Esse elemento envolve todo o conteúdo da página e às vezes é conhecido como o elemento raiz.
- `<head></head>` — o elemento [`<head>`](#). Esse elemento age como um recipiente de tudo o que você deseja incluir em uma página HTML que *não* é o conteúdo que você quer mostrar para quem vê sua página. Isso inclui coisas como [palavras-chave](#) e uma descrição que você quer que apareça nos resultados de busca, CSS para dar estilo ao conteúdo, declarações de conjuntos de caracteres e etc.
- `<meta charset="utf-8">` — esse elemento define o conjunto de caracteres que seu documento deve usar para o UTF-8, que inclui praticamente todos os caracteres da grande maioria dos idiomas escritos. Essencialmente agora ele pode manipular qualquer

~~Escrevendo, agora só posso manipular quaisquer~~

conteúdo textual que você possa colocar. Não há razão para não definir isso e assim pode ajudar a evitar alguns problemas no futuro.

- <title></title> — o elemento [<title>](#). Ele define o título da sua página, que é o título que aparece na guia do navegador onde sua página é carregada.

Ele também é usado para descrever a página quando você a adiciona aos favoritos.

- <body></body> — o elemento [<body>](#). Contém *todo* o conteúdo que você quer mostrar ao público que visita sua página, seja texto, imagens, vídeos, jogos, faixas de áudio reproduzíveis ou qualquer outra coisa.

Imagens

Vamos voltar nossa atenção para o elemento [](#) novamente:

```
|  \(en-US\)](#) - [<h6> \(en-US\)](#), embora você normalmente só use de 3 a 4:

```
<h1>Meu título principal</h1>
<h2>Meu título de alto nível</h2>
<h3>Meu subtítulo</h3>
<h4>Meu segundo subtítulo</h4>
```



Agora, tente adicionar um título adequado à sua página HTML logo acima do elemento [<img>](#).

**Nota:** Você verá que seu nível de cabeçalho 1 tem um estilo implícito. Não use elementos de cabeçalho para aumentar ou negritar o texto, pois eles são usados para acessibilidade e outros motivos, como SEO. Tente criar uma sequência significativa de títulos em suas páginas, sem pular os níveis.

## Parágrafo

Como explicado acima, os elementos [<p>](#) são para conter parágrafos de texto; você os usará com frequência ao marcar

um conteúdo de texto regular:

| <p>Este é um parágrafo simples</p>



Adicione seu texto de exemplo (você o obteve em [Como será o seu site?](#)) Em um ou alguns parágrafos, colocados diretamente abaixo do seu elemento [<img>](#).

## Listas

Muito do conteúdo da web é de listas e o HTML tem elementos especiais para elas. Listas de marcação sempre consistem em pelo menos 2 elementos. Os tipos mais comuns de lista são ordenadas e não ordenadas:

1. **Listas não ordenadas** são para listas onde a ordem dos itens não importa, como uma lista de compras, por exemplo. Essas são envolvidas em um elemento [<ul>](#).
2. **Listas Ordenadas** são para listas onde a ordem dos itens importa, como uma receita. Essas são envolvidas em um elemento [<ol>](#).

Cada item dentro das listas é posto dentro de um elemento [<li>](#) (item de lista).

Por exemplo, se nós quisermos tornar uma parte de um parágrafo numa lista:

| <p>Na Mozilla, somos uma comunidade global e



Nós podemos fazer assim:

```
<p>Na Mozilla, somos uma comunidade global e

 tecnólogos
 pensadores
 construtores

<p>trabalhando juntos ... </p>
```

Tente adicionar uma lista ordenada ou não ordenada à sua página de exemplo.

## Links

Links são muito importantes — eles são o que faz da web ser de fato uma REDE! Para adicionar um link, precisamos usar um elemento simples —  — "a" é a forma abreviada de "âncora". Para transformar o texto do seu parágrafo em um link, siga estas etapas:

1. Escolha algum texto. Nós escolhemos o texto "Mozilla Manifesto".

2. Envolva o texto em um elemento  , assim:

```
<a>Mozilla Manifesto
```

3. Dê ao elemento  um atributo href, assim:

```
Mozilla Manifesto
```

4. Preencha o valor desse atributo com o endereço da Web que você deseja vincular o link:

```
<a href="https://www.mozilla.org/pt-F
```

Você pode obter resultados inesperados se omitir a parte `https://` ou o `http://`, o chamado *protocolo*, no começo do endereço web. Então depois de criar um link, clique nele para ter certeza de que ele está indo para onde você deseja.

`href` pode parecer, numa primeira impressão, uma escolha obscura para um nome de atributo. Se você está tendo problemas para lembrar do nome, lembre que significa *hypertext reference*. (referência em hipertexto)

Adicione um link em sua página agora, se ainda não tiver feito isso.

## Conclusão

Se você seguiu todas as instruções neste artigo, você deve terminar com uma página que pareça algo do tipo (você também pode [vê-la aqui](#)):

Se você ficar emperrado, pode sempre comparar seu trabalho com nosso [código de exemplo finalizado](#) no Github.

Aqui, nós só arranhamos na superfície do HTML. Para descobrir mais, vá a nossa [Estruturando a web com HTML](#).

## Neste módulo

- [Instalando os programas básicos](#)
- [Como será o seu site?](#)
- [Lidando com arquivos](#)
- [HTML básico](#)
- [CSS básico](#)
- [Javascrip básico](#)
- [Publicando seu website](#)
- [Como a web funciona](#)

**Last modified:** 1 de fev. de 2022, [by MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

# Iniciando com HTML

Neste artigo nós abordamos os princípios básicos do HTML, para você começar. Definimos os elementos, atributos e todos os outros termos importantes que você possa ter ouvido e onde eles se encaixam na linguagem. Também mostramos como um elemento HTML é estruturado, como uma página HTML típica é estruturada e explicamos outras importantes características básicas da linguagem. Ao longo do caminho, nós brincaremos com um pouco de HTML, para despertar seu interesse!

<b>Pré-requisitos:</b>	Básico de informática, <a href="#">software básico instalado</a> e conhecimento básico de como <a href="#">trabalhar com arquivos</a> .
<b>Objetivos:</b>	Obter uma familiaridade básica com a linguagem HTML e adquirir um pouco de prática escrevendo alguns elementos HTML.

# O que é HTML?

HTML (HyperText Markup Language) não é uma linguagem de programação, é uma *linguagem de marcação* utilizada para dizer ao seu navegador como estruturar a página web que você visita. A página pode ser tanto complicada como

simples quanto o desenvolvedor web desejar que seja. HTML consiste em uma série de elementos que você usa para anexar, envolver ou *marcar* diferentes partes do conteúdo para que apareça ou aja de uma certa maneira. O fechamento das tags pode transformar uma parte do conteúdo dentro do elemento em um link para direcionar à uma outra página web, colocar as palavras em itálico, e assim por diante. Por exemplo, observe o conteúdo a seguir:

Meu gato é muito mal-humorado.

Se nós quisermos que a linha permaneça, nós podemos especificar que é um parágrafo, fechando-a com a elemento de parágrafo(<p>):

<p>Meu gato é muito mal-humorado.</p>

## Anatomia de um elemento HTML

Vamos explorar nosso elemento parágrafo um pouco mais:

Opening tag

Closing tag



As partes principais do elemento são:

1. **Tag de abertura:** Consiste no nome do elemento ( neste caso: p ), envolvido entre **parênteses angulares** de abertura e fechamento. Isso indica onde o elemento começa, ou inicia a produzir efeito — neste caso, onde o parágrafo se inicia.
2. **Tag de fechamento:** É o mesmo que a tag de abertura, exceto que este inclui uma barra diagonal antes do nome do elemento. Indica onde o elemento termina — neste caso, onde fica o fim do parágrafo. Falhar em incluir o fechamento de uma tag é um erro comum para iniciantes e pode levar a resultados estranhos.
3. **O conteúdo:** Este é o conteúdo do elemento, que neste caso é somente texto.
4. **O elemento:** A tag de abertura, mais a tag de fechamento, mais o conteúdo, é igual ao elemento.

Aprendizado ativo: criando seu primeiro elemento HTML

Edite a linha abaixo na área *Entrada* colocando-a entre as tags `<em>` e `</em>` (adicone o `<em>` antes para abrir o

*elemento*, e `</em>` depois, para fechar o *elemento*). Isto dará à linha uma ênfase em itálico! Você poderá ver as mudanças efetuadas no momento na área *Saída*.

Caso você cometa um erro, você pode usar o botão *Resetar* para desfazer a ação incorreta. Se você realmente não souber o que fazer, pressione o botão *Mostrar solução* para visualizar a resposta.

## **Saída ao vivo**

Este é meu texto.

## **Código editável**

Pressione Esc para afastar o foco da área de código (Tab insere um caractere de tabulação).

Este é meu texto.

**Resetar**

**Mostrar solução**

## Aninhando elementos

Elementos podem ser inseridos dentro de outros elementos — isto é chamado de **aninhamento**. Se nós quisermos dizer

que nosso gato é **muito** mal-humorado, nós poderemos envolver a palavra "muito" com o elemento <strong>, que significa enfatizar fortemente a palavra:

```
| <p>Meu gato é muito mal-
```

No entanto, você precisa garantir que seus elementos estejam adequadamente aninhados: no exemplo acima nós abrimos o elemento `p` primeiro, e então o elemento `strong`, portanto temos que fechar o elemento `strong` primeiro, depois o `p`. O código a seguir está errado:

```
| <p>Meu gato é muito mal-hum...
```

Os elementos devem abrir e fechar corretamente para que eles fiquem claramente dentro ou fora do outro. Caso eles se sobreponham, como no exemplo acima, então o seu navegador tentará adivinhar o que você quis dizer, e talvez você obtenha resultados inesperados. Então não faça isso!

## Elementos em bloco versus elementos inline

Há duas categorias importantes no HTML, que você precisa conhecer. Eles são elementos em bloco e elementos inline.

- Elementos em bloco formam um bloco visível na página — eles aparecerão em uma nova linha logo após

qualquer elemento que venha antes dele, e qualquer conteúdo depois de um elemento em bloco também aparecerá em uma nova linha. Elementos em bloco geralmente são elementos estruturais na página que representam, por exemplo: parágrafos, listas, menus de navegação, rodapés etc. Um elemento em bloco não seria aninhado dentro de um elemento inline, mas pode ser aninhado dentro de outro elemento em bloco.

- Elementos inline (na linha) são aqueles que estão contidos dentro de elementos em bloco envolvem apenas pequenas partes do conteúdo do documento e não parágrafos inteiros ou agrupamentos de conteúdo. Um elemento inline não fará com que uma nova linha apareça no documento: os elementos inline geralmente aparecem dentro de um parágrafo de texto, por exemplo: um elemento `<a>` (hyperlink) ou elementos de ênfase como `<em>` ou `<strong>`.

Veja o seguinte exemplo:

```
primeirosegundoterceiro
<p>quarto</p><p>quinto</p><p>sexta</p>
```

O elemento `<em>` é inline, então como você pode ver abaixo, os três primeiros elementos ficam na mesma linha uns dos outros sem espaço entre eles. O `<p>`, por outro lado, é um elemento em bloco, então cada elemento aparece em uma nova linha, com espaço acima e abaixo de cada um (o espaçamento é devido à estilização CSS padrão que o

browser aplica aos parágrafos).

*primeiro*  
*segundo*  
*terceiro*

*quarto*

*quinto*

*sexto*

**Nota:** o HTML5 redefiniu as categorias de elemento em HTML5: veja [Categorias de conteúdo de elementos](#). Enquanto essas definições são mais precisas e menos ambíguas que as anteriores, elas são muito mais complicadas de entender do que "em bloco" e "inline", então usaremos estas ao longo deste tópico.

**Nota:** Os termos "bloco" e "inline", conforme usados neste tópico, não devem ser confundidos com os [tipos de caixas CSS \(en-US\)](#) com os mesmos nomes. Embora

eles se correlacionem por padrão, alterar o tipo de exibição CSS não altera a categoria do elemento e não afeta em quais elementos ele pode conter e em quais elementos ele pode estar contido. Um dos motivos pelos quais o HTML5 abandonou esses termos foi evitar essa confusão bastante comum.

**Nota:** Você pode encontrar páginas de referência úteis que incluem uma lista de elementos inline e em bloco — veja [elementos em bloco](#) e [elementos inline](#).

## Elementos vazios

Nem todos os elementos seguem o padrão acima de: tag de abertura, conteúdo, tag de fechamento. Alguns elementos consistem apenas em uma única tag, que é geralmente usada para inserir/incorporar algo no documento no lugar em que ele é incluído. Por exemplo, o elemento `<img>` insere uma imagem em uma página na posição em que ele é incluído:

```
| My cat is very grumpy</p>
```

Attribute

Atributos contém informação extra sobre o elemento, mas que você não deseja que apareça no conteúdo. Neste caso, o atributo `class` permite que você dê ao elemento um nome

de identificação, que pode ser usada mais tarde para direcionar informação de estilo ao elemento e outras coisas.

Um atributo deve conter:

1. Um espaço entre ele e o nome do elemento (ou o atributo anterior, caso o elemento já contenha um ou mais atributos.)
2. O nome do atributo, seguido por um sinal de igual.
3. Um valor de atributo, com aspas de abertura e fechamento em volta dele.

## Aprendizado ativo: Adicionando atributos a um elemento

Outro exemplo de um elemento é `<a>` — isso significa "âncora" e fará com que a parte do texto que ele envolve vire um link. Isso pode ter vários atributos, mas os mais comuns são os seguintes:

- O valor desse atributo especifica o endereço da web para o qual você deseja que o link aponte; onde o

navegador irá quando o link for clicado. Por exemplo `href="https://www.mozilla.org/"`.

- `title`: O atributo `title` especifica uma informação extra sobre o link, assim como o assunto da página que está sendo linkada. Por exemplo `title="Homepage da Mozilla"`. Isto será exibido como uma *tooltip* (*dica de contexto*) quando passarmos o mouse sobre o link.

Edite a linha abaixo na área de Entrada para transformá-la em um link para o seu site favorito.

1. Primeiro, adicione o elemento `<a>`.
2. Segundo, adicione o atributo `href` e o atributo `title`.
3. Por último, especifique o atributo `target` para abrir o link em uma nova aba.

Você poderá ver as atualizações das alterações ao vivo na área Saída. Você deve ver um link que, quando passa o mouse sobre ele, exibe o valor do atributo `title` e, quando clicado, navega para o endereço da web no atributo `href`. Lembre-se de que você precisa incluir um espaço entre o nome do elemento e cada atributo.

Caso você cometa um erro, você poderá desfazê-lo usando o botão *Resetar*. Caso você realmente não saiba como fazer, pressione o botão *Mostrar solução* para ver a resposta.

## Saída ao vivo

Um link para o meu site favorito.

## Código editável

Pressione Esc para afastar o foco da área de código (Tab insere um caractere de tabulação).

```
<p>Um link para o meu site
favorito.</p>
```

[Resetar](#)

[Mostrar solução](#)

## Atributos booleanos

Às vezes você verá atributos escritos sem valores — isso é permitido nos chamados atributos booleanos, e eles podem

ter somente um valor, que é geralmente o mesmo nome do atributo. Por exemplo, o atributo `disabled` você pode atribuir para os elementos de entrada de formulários, se desejar que estes estejam desativados (acinzentados), para que o usuário não possa inserir nenhum dado neles.

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

De forma abreviada, é perfeitamente permitido escrever isso da seguinte maneira (também incluímos um elemento de entrada de formulário não desativado para referência, para dar uma idéia do que está acontecendo):

```
<!-- o uso do atributo disabled impede qu
<input type="text" disabled>

<!-- O usuário pode inserir texto na caixa de
<input type="text">
```

Ambos resultarão em uma *Saída* da seguinte forma:

A screenshot showing two empty input fields side-by-side. The top field has a thin gray border, while the bottom field has a thicker black border, indicating they are different elements or states.

Omitindo as aspas dos valores do atributo  
Olhando a World Wide Web (internet), você encontrará todos

os tipos de estilos de marcação HTML, incluindo valores de atributos sem as aspas. Isso é permitido em algumas circunstâncias, mas irá quebrar sua marcação em outras. Por exemplo, se voltarmos ao exemplo anterior de link , nós podemos escrever a versão básica deste somente com o atributo href , da seguinte forma:

```
site favorit
```

No entanto, assim que adicionamos o atributo title neste elemento, a marcação resultará em erro:

```
<a href=https://www.mozilla.org/ title=
```

Neste ponto, o navegador irá interpretar errado sua marcação, de modo a pensar que o atributo title trata-se, na verdade, de três atributos: o atributo title com o valor "The", e dois atributos booleanos, Mozilla e homepage . Esta obviamente não era a intenção e irá causar erros ou um comportamento inesperado no código, assim como visto no exemplo abaixo. Tente colocar o mouse em cima do link para visualizar qual é o título que aparece!

```
site favorito
```

Nossa recomendação é *sempre incluir as aspas nos valores dos atributos* — isto evita inúmeros problemas, além de resultar em um código mais legível.

## Aspas simples ou duplas?

Você pode perceber que os valores dos atributos exemplificados neste artigo estão todos com aspas duplas, mas você poderá ver aspas simples no HTML de algumas pessoas. Esta é puramente uma questão de estilo e você pode se sentir livre para escolher qual prefere. As duas linhas de código a seguir são equivalentes:

```
Um link a
Um link para
```

Entretanto, você deve se certificar de não misturar os dois tipos de aspas juntos. O exemplo a seguir está errado!

```
Um link a
```

Se utilizar um tipo de aspas no seu HTML, você pode inserir o outro tipo de aspas no texto, por exemplo, que não ocorrerá erro, desta forma:

```
<a href="http://www.example.com" title="I t
```

No entanto, se você quiser incluir aspas, dentro de aspas onde ambas as aspas são do mesmo tipo (aspas simples ou aspas duplas), será necessário usar entidades HTML para as aspas. Por exemplo, isso irá quebrar:

```
<a href='http://www.example.com' title='In
```

Então você precisa fazer isso:

```
<a href='http://www.example.com' title='I #
```

## Anatomia de um documento HTML

Já vimos os conceitos básicos dos elementos individuais do HTML, mas eles não são muito úteis sozinhos. Vamos aprender como estes elementos individuais são combinados entre si para criar uma página HTML inteira:

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8">
 <title>My test page</title>
 </head>
 <body>
 <p>This is my page</p>
 </body>
</html>
```

Neste código nós temos:

1. `<!DOCTYPE html>`: O doctype. Nas névoas do tempo, quando o HTML era recente (por volta de 1991/2), doctypes funcionavam como links para uma série de regras as quais uma página HTML tinha que seguir para ser considerada uma página com um bom HTML, o que poderia significar a verificação automática de erros e outras coisas úteis. Ele costumava ser assim:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-t.
```

No entanto, atualmente, ninguém se importa com eles, e eles são realmente apenas um artefato histórico que precisa ser incluído para que tudo funcione corretamente. `<!DOCTYPE html>` é a menor cadeia de caracteres que conta como um doctype válido; é tudo o que você realmente precisa saber.

2. `<html></html>`: O elemento `<html>` envolve o conteúdo da página inteira e é conhecido como o "elemento raiz" da página HTML.
3. `<head></head>`: O elemento `<head>` atua como um container para todo o conteúdo da página HTML que não é visível para os visitantes do site. Isso inclui palavras-chave e a descrição da página que você quer que apareça nos resultados de busca, o CSS para estilizar o conteúdo da página (apesar de ser recomendado faze-lo num arquivo separado), declaração de conjunto de caracteres, e etc.

Você aprenderá mais sobre isso no próximo artigo da série.

4. `<meta charset="utf-8">` : Este elemento define o tipo da codificação dos caracteres que o seu documento deve usar, neste caso, para o tipo UTF-8, que inclui a maioria dos caracteres das línguas humanas escritas. Essencialmente, ele consegue lidar com qualquer tipo de conteúdo textual que você puder colocar no documento. Não há motivos para não indicar essa informação e esse elemento ajuda a evitar vários problemas na sua página.
5. `<title></title>` : O elemento `<title>` . Isto define o título de sua página, que aparecerá na guia do navegador na qual a página está carregada e é usado para descrevê-la quando for salva nos favoritos.
6. `<body></body>` : O elemento `<body>` contém *todo* o conteúdo que você quer mostrar aos usuários quando eles visitarem sua página, como texto, imagens, vídeos, jogos, faixas de áudio reproduzíveis, ou qualquer outra coisa.

## Aprendizado ativo: Adicionando alguns recursos ao documento HTML

Se você quiser experimentar como funciona um documento HTML no seu computador, você pode:

1. Copiar o exemplo de página HTML mostrada acima.
2. Criar um novo documento em seu editor de texto.

3. Colar o código no novo arquivo de texto.
4. Salvar o arquivo com o nome `index.html`.

**Nota:** Você também pode encontrar o template básico de HTML no [MDN Learning Area Github repo](#) .

Você pode abrir este arquivo no navegador para ver como o código renderizado se apresenta, e então, editar o código e atualizar a página no navegador para ver o resultado com as mudanças. Inicialmente será exibido assim:

Neste exercício, você pode editar o código localmente em seu computador, com descrito acima, ou você pode editá-lo na janela de exemplo editável abaixo (esta janela editável representa apenas o conteúdo do elemento `<body>` , neste caso). Nós gostaríamos que você seguisse as seguintes etapas:

- Logo abaixo da tag de abertura `<body>` , adicione um título principal para o documento. Isso deve estar dentro da tag de abertura `<h1>` e da tag de fechamento `</h1>` .
- Edite o conteúdo do parágrafo para incluir algum texto sobre algo de seu interesse.
- Faça com que todas as palavras importantes sejam destacadas em negrito, colocando-as dentro da tag de abertura `<strong>` e da tag de fechamento

</strong> .

- Adicione um link ao seu parágrafo, conforme [explicado anteriormente neste artigo](#).
- conforme explicado anteriormente no artigo. Você receberá pontos de bônus se conseguir vincular a uma imagem diferente (localmente no seu computador ou em outro lugar da web).

Caso você cometa um erro, você poderá desfazê-lo usando o botão *Resetar*. Caso você realmente não saiba como fazer, pressione o botão *Mostrar solução* para ver a resposta.

## Espaços em branco no HTML

Nos exemplos anteriores, você pode ter percebido a presença de espaços em branco nos códigos — isto não é necessário; os dois trechos de códigos a seguir são equivalentes:

```
<p>Dogs are silly.</p>
```

```
<p>Dogs are
 silly.</p>
```



Não importa quantos espaços em branco você use (que pode incluir caracteres de espaço, mas também quebras de linha), o analisador de HTML reduz cada um deles em um único espaço ao renderizar o código. Então, por que espaço em branco? A resposta é legibilidade — é muito mais fácil entender o que está acontecendo no seu código, se você o tiver bem formatado, e não apenas agrupado em uma grande confusão. Em nosso HTML, cada elemento aninhado é indentado em dois espaços a mais do que aquele em que está localizado. Depende de você que estilo de formatação

você usa (quantos espaços para cada nível de recuo, por exemplo), mas considere formatá-lo.

## Referências de entidades: incluindo caracteres especiais no HTML

No HTML, os caracteres <, >, " , ' e o & são caracteres especiais. Eles fazem parte da própria sintaxe HTML; portanto, como você inclui um desses caracteres no seu texto? Por exemplo, se você realmente deseja usar um e comercial ou sinal de menor que, e não tenha ele interpretado como código.

Temos que usar referências de caracteres — códigos especiais que representam caracteres e podem ser usados nessas circunstâncias. Cada referência de caractere é iniciada com um e comercial (&) e finalizada por um ponto e vírgula (;).

Caractere literal	Referência de caractere equivalente
<	&lt;
>	&gt;
"	&quot;
'	&apos;
&	&amp;

Caractere literal	Referência de caractere equivalente
-------------------	-------------------------------------

No exemplo abaixo, você pode ver dois parágrafos, que estão falando sobre tecnologias da web:

```
<p>Em HTML, você define um parágrafo usando o
<p>Em HTML, você define um parágrafo usando o
```

Na saída ao vivo abaixo, você pode ver que o primeiro parágrafo deu errado, porque o navegador pensa que a segunda instância de `<p>` está iniciando um novo parágrafo. O segundo parágrafo parece bom, porque substituímos os parênteses angulares por referências de caracteres.

**Nota:** A tabela com todas as referências de caracteres disponíveis em HTML pode ser encontrada na Wikipédia: [List of XML and HTML character entity references](#).

Observe que você não precisa usar referências de

entidade para outros símbolos, pois os navegadores modernos manipularão os símbolos reais muito bem, desde que a codificação de caracteres do HTML esteja definida como UTF-8.

## Comentários no HTML

Em HTML, assim como na maioria das linguagens de programação, há um mecanismo para escrevermos comentários no código — comentários são ignorados pelo navegador e são invisíveis para o usuário, seu propósito é permitir a inclusão de comentários para descrever como o código funciona, especificar o que cada parte dele faz, e por ai vai. Isso pode ser muito útil se você retornar a uma base de código em que não trabalhou há muito tempo e não se lembrar do que fez — ou se você entregar seu código para outra pessoa trabalhar.

Para transformar uma seção do conteúdo HTML em um comentário, você precisa agrupá-lo nos marcadores especiais `<!--` e `-->`, por exemplo:

```
<p>Eu não estou dentro de um comentário</p>
```

```
<!-- <p>Eu estou!</p> -->
```

Como você pode ver abaixo, o primeiro parágrafo fica visível na saída ao vivo, mas o segundo (que é um comentário) não.

## Sumário

Você chegou ao final do artigo — esperamos que tenha gostado do seu tour pelos princípios básicos do HTML! Nesse ponto, você deve entender como é a linguagem, como ela funciona em um nível básico e ser capaz de escrever alguns elementos e atributos. Este é o lugar perfeito para se estar agora, já que os artigos subsequentes deste módulo abordarão algumas das coisas que você já examinou com mais detalhes e introduzirão alguns novos conceitos da linguagem. Fique ligado!

**Nota:** Nesse ponto, à medida que você começa a aprender mais sobre HTML, também pode querer explorar os conceitos básicos de Cascading Style Sheets, ou [CSS](#). CSS é a linguagem usada para estilizar suas páginas da web (por exemplo, alterando a fonte ou as cores ou alterando o layout da página). HTML e CSS vão muito bem juntos, como você descobrirá em breve.

## Veja também

- [Aplicando cores a elementos HTML usando CSS](#)

# Neste módulo

- [Iniciando com HTML](#)
- [O que está no cabeçalho? Metadados em HTML](#)
- [Fundamentos do texto em HTML](#)
- [Criando links](#)
- [Formatação avançada de texto](#)
- [Estrutura do documento e site](#)
- [Depurando HTML](#)
- [Marcando uma carta](#)
- [Estruturando o conteúdo de uma página](#)

**Last modified:** 31 de jan. de 2022, [by MDN contributors](#)



[This page was translated from English by the community.](#) [Learn more and join the MDN Web Docs community.](#)

## O que está no cabeçalho? Metadados em HTML

O [head \(en-US\)](#) de um documento HTML é a parte que não é exibida no navegador da Web quando a página é carregada. Ele contém informações como [title](#), links para [CSS](#) (se você deseja modelar seu conteúdo HTML com CSS), links para favicons personalizados e outros metadados (dados sobre o HTML, como quem o escreveu, e palavras-chave importantes que descrevem o documento.) Neste artigo, abordaremos todas as coisas acima e mais. Dando-lhe uma boa base para lidar com marcação.

<b>Pré-requisitos:</b>	Familiaridade básica em HTML, tal como <a href="#">Iniciando com HTML</a> .
<b>Objetivo:</b>	Aprender sobre o cabeçalho HTML, seu propósito, os itens mais importantes que ele pode conter e que efeito isso pode ter no documento HTML.

## O que há no cabeçalho HTML?

Vamos rever o simples [Documento HTML que abordamos no artigo anterior](#):

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8">
 <title>Minha página de teste</title>
 </head>
 <body>
 <p>Essa é minha página</p>
 </body>
</html>
```

O cabeçalho HTML é o conteúdo do elemento `<head>` — ao contrário do conteúdo do elemento `<body>` (que são exibidos na página quando carregados no navegador), o conteúdo do cabeçalho não é exibido na página, em vez disso, o trabalho do cabeçalho é conter [metadados](#) sobre o documento. No exemplo seguinte, o cabeçalho é bem simples:

```
<head>
 <meta charset="utf-8">
 <title>Minha página de teste</title>
</head>
```

Em páginas maiores, o cabeçalho pode ter mais conteúdo. Tente acessar um dos seus sites favoritos e use as [ferramentas de desenvolvimento](#) para verificar o conteúdo do cabeçalho. Nossa objetivo aqui não é mostrar a você como usar tudo o que é possível pôr no cabeçalho, mas te ensinar a usar as coisas mais obvias que você vai querer incluir no cabeçalho, e lhe dar alguma familiaridade. Vamos começar.

## Adicionando um título

Nós já vimos o elemento `<title>` em ação — ele pode ser usado para adicionar um título ao documento, mas pode ser confundido com o elemento `<h1>` ([en-US](#)), que é usado para adicionar um título de nível superior ao conteúdo do body — as vezes também é associado como o título da página. Mas são coisas diferentes!

- O elemento `<h1>` ([en-US](#)) aparece na página quando é carregado no navegador — geralmente isso deve ser usado uma vez por página, para marcar o título do conteúdo da sua página, (o título da história, ou da notícia, ou o que quer que seja apropriado para o uso).
- O elemento `<title>` é um metadado que representa o título de todo o document HTML (não o conteúdo do documento).

## Aprendizado ativo: observando um exemplo simples

1. Para começar esta aprendizagem ativa, gostaríamos que você fosse ao nosso depósito GitHub e baixasse uma cópia do nossa página [title-example.html](#). Para fazer isso:
  1. Copie e cole o código em um novo arquivo de texto no seu editor e salve-o com o nome de index.html em um lugar de fácil acesso.
  2. Pressione o botão "Raw" na página do GitHub, que faz com que o código bruto apareça (possivelmente em uma nova guia do navegador). Em seguida, escolha o menu *Arquivo do navegador* > *Salvar página como ...* e escolha um local adequado para salvar o arquivo.
2. Agora abra o arquivo no seu navegador. Você deve ver algo assim:  
Agora deve ser completamente óbvio onde o conteúdo `<h1>` aparece e onde o conteúdo `<title>` aparece!

3. Você também deve tentar abrir o código no seu editor, editar o conteúdo desses elementos e atualizar a página no seu navegador. divirta-se.

O conteúdo do elemento `<title>` também é usado de outras maneiras. Por exemplo, se você tentar favoritar a página, (*Favoritos > Adicionar página aos favoritos* ou o ícone da estrela na barra de URL no Firefox), você verá o conteúdo `<title>` preenchido como o nome sugerido do favorito.

Os conteúdos `<title>` também são usados nos resultados de pesquisa, conforme você verá abaixo.

## Metadados: o elemento `<meta>`

Metadados é dado descreve dados, e HTML possui uma maneira "oficial" de adicionar metadados a um documento — o elemento `<meta>`. Claro, as outras coisas em que estamos falando neste artigo também podem ser pensadas como metadados. Existem muitos tipos diferentes de elementos `<meta>` que podem ser incluídos no `<head>` da sua página, mas não tentaremos explicar todos eles nesta fase, pois seria muito confuso. Em vez disso, explicaremos algumas coisas que você pode ver comumente, apenas para lhe dar uma idéia.

### Especificando a codificação de caracteres do seu documento

No exemplo que vimos acima, esta linha foi incluída:

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

Este elemento simplesmente especifica a codificação de caracteres do documento — o conjunto de caracteres que o documento está autorizado a usar. `utf-8` é um conjunto de caracteres universal que inclui praticamente qualquer caractere de qualquer linguagem humana. Isso significa que sua página web poderá lidar com a exibição de qualquer idioma; portanto, é uma boa idéia configurar isso em todas as páginas web que você cria! Por exemplo, sua página poderia lidar com o Inglês e Japonês muito bem:

Se você definir sua codificação de caracteres para `ISO-8859-1`, por exemplo (o conjunto de caracteres para o alfabeto latino), a renderização de sua página ficaria toda bagunçada:

**Nota:** Alguns navegadores (como o Chrome) corrigem automaticamente as codificações incorretas, então, dependendo do navegador que você usar, você pode não ver esse problema. Ainda assim, você deve definir uma codificação do `utf-8` em sua página, para evitar problemas em outros navegadores.

## Aprendizagem ativa: Experimento com a codificação de caracteres

Para experimentar isso, revise o modelo HTML simples que você obteve na seção anterior em <title> (a página [title-example.html](#)), tente alterar o valor do meta charset para ISO-8859-1 e adicione o Japonês à sua página . Este é o código que usamos:

```
| <p>Exemplo Japonês:ご飯が熱い</p>
```

### Adicionando um autor e descrição

Muitos elementos <meta> incluem atributos de name e content :

- O name especifica o tipo de elemento meta que é; que tipo de informação contém.
- O content especifica o conteúdo real do meta.

Dois desses meta-elementos que são úteis para incluir na sua página definem o autor da página e fornecem uma descrição concisa da página. Vejamos um exemplo:

```
| <meta name="author" content="Chris Mills">
| <meta name="description" content="A Área de Aprendizagem do MDN t
| proporcionar iniciantes em Web com tudo o que eles precisam saber
| para começar a desenvolver sites e aplicativos.">
```

Especificar um autor é útil de muitas maneiras: é útil para poder descobrir quem escreveu a página, se quiser enviar perguntas sobre o conteúdo que você gostaria de contactá-la. Alguns sistemas de gerenciamento de conteúdo possuem ferramentas para extrair automaticamente as informações do autor da página e disponibilizá-las para seus propósitos.

Especificar uma descrição que inclua palavras-chave relacionadas ao conteúdo da sua página é útil porque tem potencial para tornar sua página mais alta nas pesquisas relevantes realizadas nos mecanismos de busca (tais atividades são denominadas [Search Engine Optimization](#) ou [SEO](#)).

## Aprendizagem ativa: Uso da descrição nos motores de busca

A descrição também é usada nas páginas de resultados do mecanismo de pesquisa. Vamos passar por um exercício para explorar isso

1. Vá para a [página inicial da Mozilla Developer Network](#).
2. Veja a fonte da página (botão direito do mouse/ **Ctrl** + clique na página, escolha Ver código-fonte da página no menu de contexto).
3. Encontre a meta tag de descrição. Isso parecerá assim:

```
<meta name="description" content="A Mozilla Developer Network fornece informações sobre tecnologias Open Web, incluindo HTML, CSS e APIs para aplicativos HTML5. Ele também documenta produtos Mozilla, como o sistema operacional Firefox.">
```

4. Agora, procure por "Mozilla Developer Network" no seu motor de busca favorito (Utilizamos o Yahoo.) Você notará a descrição `<meta>` e `<title>` elemento usado no resultado da pesquisa — definitivamente vale a pena ter!

**Nota:** No Google, você verá algumas subpáginas relevantes do MDN listadas abaixo do principal link da página inicial do MDN — estes são chamados de sitelinks e são configuráveis nas [Ferramentas para webmasters do Google](#) — uma maneira de melhorar os resultados de pesquisa do seu site no mecanismo de pesquisa do Google.

**Nota:** Muitos recursos `<meta>` simplesmente não são mais usados. Por exemplo, a palavra-chave `<meta>` elemento (`<meta name="keywords" content="preencha, suas, palavras-chave, aqui">`) — que é suposto fornecer palavras-chave para os motores de busca para determinar a relevância dessa página para diferentes termos de pesquisa — são ignorados pelos motores de busca, porque os spammers estavam apenas preenchendo a lista de palavras-chave com centenas de palavras-chave, influenciando os resultados.

## Outros tipos de metadados

Ao navegar pela web, você também encontrará outros tipos de metadados. Muitos dos recursos que você verá em sites são criações proprietárias, projetados para fornecer a determinados sites (como sites de redes sociais) informações específicas que eles podem usar.

Por exemplo, [Open Graph Data](#) é um protocolo de metadados que o Facebook inventou para fornecer metadados mais ricos para sites. No código-fonte MDN, você encontrará isso:

```
<meta property="og:image" content="https://developer.mozilla.org/...>
<meta property="og:description" content="A Mozilla Developer Network fornece informações sobre tecnologias Open Web, incluindo HTML, CSS e APIs para aplicativos HTML5. Ele também documenta produtos Mozilla, como o sistema operacional Firefox.">
<meta property="og:title" content="Mozilla Developer Network">
```

Um efeito disso é que, quando você liga a MDN no facebook, o link aparece junto com uma imagem e descrição: uma experiência mais rica para usuários.

O Twitter também possui seus próprios metadados proprietários, o que tem um efeito semelhante quando o URL do site é exibido no twitter.com. Por exemplo:

```
| <meta name="twitter:title" content="Mozilla Developer Network">
```



## Adicionando ícones personalizados ao seu site

Para enriquecer ainda mais o design do seu site, você pode adicionar referências a ícones personalizados em seus metadados, e estes serão exibidos em determinados contextos. O mais usado é o **favicon** (abreviação de "favorites icon", referindo-se ao seu uso nas listas "favoritos" nos navegadores).

O humilde favicon existe há muitos anos. É o primeiro ícone desse tipo: um ícone 16 pixels quadrados usado em vários lugares. Você pode ver (dependendo do navegador) ícones favoritos exibidos na guia do navegador que contém cada página aberta e ao lado de páginas marcadas no painel de favoritos.

Um favicon pode ser adicionado à sua página:

1. Salvando-o no mesmo diretório que a página de índice do site, salvo no formato `.ico` (a maioria dos navegadores suportará favicons em formatos mais comuns como `.gif` ou `.png`, mas usar o formato ICO irá garantir que ele funcione tão bem como o Internet Explorer 6.)
2. Adicionando a seguinte linha ao HTML `<head>` para fazer referência a ele:

```
| <link rel="shortcut_icon" href="favicon.ico" type="image/x-ic
```



Aqui está um exemplo de um favicon em um painel de favoritos:

Há muitos outros tipos de ícones para considerar nestes dias também. Por exemplo, você encontrará isso no código-fonte da página inicial do MDN Web Docs:

```
| <!-- iPad de terceira geração com tela retina de alta resolução: -->
| <link rel="apple-touch-icon-precomposed" sizes="144x144" href="https://developer.mozilla.org/favicon-144x144.png" />;
| <!-- iPhone com tela retina de alta resolução: -->
| <link rel="apple-touch-icon-precomposed" sizes="114x114" href="https://developer.mozilla.org/favicon-114x114.png" />;
| <!-- iPad de primeira e segunda geração: -->
| <link rel="apple-touch-icon-precomposed" sizes="72x72" href="https://developer.mozilla.org/favicon-72x72.png" />;
| <!-- iPhone não-Retina, iPod Touch e dispositivos Android 2.1+: -->
| <link rel="apple-touch-icon-precomposed" href="https://developer.mozilla.org/favicon-36x36.png" />;
| <!-- favicon básico -->
| <link rel="shortcut_icon" href="https://developer.mozilla.org/static/favicon.ico" />
```

Os comentários explicam onde cada ícone é usado - esses elementos cobrem coisas como fornecer um ícone de alta resolução agradável para usar quando o site é salvo na tela inicial do iPad.

Não se preocupe muito com a implementação de todos esses tipos de ícone agora — este é um recurso bastante avançado, e você não precisará ter conhecimento disso para avançar no curso. O objetivo principal aqui é permitir que você saiba o que são essas coisas, no caso de você encontrá-las enquanto navega no código-fonte dos outros sites.

**Nota:** Se o seu site usa uma Política de Segurança de Conteúdo (CSP) para aumentar sua segurança, a política se aplica ao favicon. Se você encontrar problemas com o favicon não carregando, verifique se a diretiva `img-src(en-US)` do cabeçalho `Content-Security-Policy` não está impedindo o acesso a ele.

## Aplicando CSS e JavaScript ao HTML

Todos os sites que você usar nos dias atuais empregarão o [CSS](#) para torná-los legais e o [JavaScript](#) para ativar a funcionalidade interativa, como players de vídeo, mapas, jogos e muito mais. Estes são comumente aplicados a uma página web usando o elemento [`<link>`](#) e o elemento [`<script>`](#), respectivamente.

- O elemento [`<link>`](#) sempre vai no cabeçalho do seu documento. Isso requer dois atributos, `rel = "stylesheet"`, que indica que é a folha de estilo do documento e `href`, que contém o caminho para o arquivo de folha de estilo:

```
<link rel="stylesheet" href="meu-arquivo-css.css">
```

- O elemento [`<script>`](#) não precisa ir no cabeçalho; na verdade, muitas vezes é melhor colocá-lo na parte inferior do corpo do documento (antes da tag `</body>` de fechamento), para garantir que todo o conteúdo HTML tenha sido lido pelo navegador antes de tentar aplicar o JavaScript nele (se o JavaScript tentar acessar um elemento que ainda não existe, o navegador gerará um erro.)

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

**Nota:** O elemento `<script>` pode parecer um elemento vazio, mas não é, e, portanto, precisa de uma tag de fechamento. Em vez de apontar para um arquivo de script externo, você também pode escolher colocar seu script dentro do elemento `<script>`.

## Aprendizagem ativa: aplicar CSS e JavaScript a uma página

1. Para iniciar esta aprendizagem ativa, pegue uma cópia dos nossos arquivos [`meta-example.html`](#), [`script.js`](#) e [`style.css`](#) e salve-os em seu computador local no mesmo diretório. Verifique se eles são salvos com os nomes e extensões de arquivo corretos.

2. Abra o arquivo HTML em seu navegador e seu editor de texto.
3. Ao seguir as informações fornecidas acima, adicione os elementos `<link>` e `<script>` ao seu HTML, para que seu CSS e JavaScript sejam aplicados ao seu HTML.

Se for feito corretamente, quando você salvar seu HTML e atualizar seu navegador, verá que as coisas mudaram:

- O JavaScript adicionou uma lista vazia à página. Agora, quando você clica em qualquer lugar da lista, uma caixa de diálogo aparecerá pedindo para que você, insira algum texto para um novo item de lista. Quando você pressiona o botão OK, um novo item será adicionado à lista contendo o texto. Quando você clica em um item de lista existente, uma caixa de diálogo será exibida permitindo que você altere o texto do item.
- O CSS fez com que o plano de fundo ficasse verde e o texto se tornasse maior. Ele também estilizou parte do conteúdo que o JavaScript adicionou à página (a barra vermelha com a borda preta é o estilo que o CSS adicionou à lista gerada por JS).

**Nota:** Se você ficar preso neste exercício e não conseguir aplicar o CSS/JS, tente verificar nossa página de exemplo [css-and-js.html](#).

## Definir o idioma principal do documento

Finalmente, vale a pena mencionar que você pode (e realmente deveria) definir o idioma da sua página. Isso pode ser feito adicionando o [atributo lang](#) à tag HTML de abertura (como visto no [meta-example.html](#) e mostrado abaixo).

```
| <html lang="pt-BR">
```

Isso é útil de várias maneiras. O seu documento HTML será indexado de forma mais eficaz pelos motores de busca se o seu idioma for definido (permitindo que ele apareça corretamente em resultados específicos do idioma, por exemplo), e é útil para pessoas com deficiências visuais usando leitores de tela (por exemplo, a palavra "seis" existe em Francês e Inglês, mas é pronunciado de forma diferente.)

Você também pode definir seções secundárias do seu documento para serem reconhecidas em diferentes idiomas. Por exemplo, podemos configurar nossa seção do idioma Japonês para ser reconhecida como japonesa, assim:

```
| <p>Exemplo japonês: ご飯が熱い。.</p>
```

Esses códigos são definidos pelo padrão [ISO 639](#). Você pode encontrar mais sobre eles em [Tags de idioma em HTML e XML](#).

## Resumo

Isso marca o fim de nossa rápida turnê pelo HTML — há muito mais que você pode fazer aqui, mas uma excursão exaustiva seria chata e confusa nesta fase, e nós só queríamos dar uma idéia das coisas mais comuns você encontrará lá, por enquanto! No próximo artigo, veremos o básico do texto HTML.

**Last modified:** 31 de jan. de 2022, [by MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Fundamentos do texto em HTML

Um dos principais objetivos do HTML é dar estrutura de texto e significado, também conhecido como [semântica](#)), para que um navegador possa exibi-lo corretamente. Este artigo explica a forma como [HTML](#) pode ser usado para estruturar uma página de texto, adicionar títulos e parágrafos, enfatizar palavras, criar listas e muito mais.

<b>Pré-requisitos:</b>	Familiaridade básica em HTML, tal como coberto <a href="#">Iniciando com o HTML</a> .
<b>Objetivo:</b>	Aprenda como marcar uma página básica de texto para dar-lhe estrutura e significado — incluindo parágrafos, títulos, listas, ênfase e citações.

## O básico: Cabeçalhos e Parágrafos

O texto mais estruturado é composto por títulos e

parágrafos, esteja você lendo uma história, um jornal, um livro da faculdade, uma revista, etc.



O conteúdo estruturado torna a experiência de leitura mais fácil e mais agradável.

Em HTML, cada parágrafo deve ser envolvido em um elemento <p>, assim:

<p>Eu sou um parágrafo, oh sim, eu sou.</p>

Cada título deve ser envolvido em um elemento de título:

| <h1>Eu sou o título da história.</h1>

Existem seis elementos de título — [<h1> \(en-US\)](#), [<h2> \(en-US\)](#), [<h3> \(en-US\)](#), [<h4> \(en-US\)](#), [<h5> \(en-US\)](#) e [<h6> \(en-US\)](#). Cada elemento representa um nível diferente de conteúdo no documento; 

# representa o título

principal, 

## representa subtítulos, representa sub-subtítulos, e assim por diante.

## Implementando hierarquia estrutural

Como exemplo, em uma história, 

# representaria o título da história, representaria o título de cada capítulo e representaria sub-seções de cada capítulo, e assim por diante.

<h1>O furo esmagador</h1>

<p>Por Chris Mills</p>

<h2>Capítulo 1: A noite escura</h2>

<p>Era uma noite escura. Em algum lugar, uma

<h2>Capítulo 2: O eterno silêncio</h2>

<p>Nosso protagonista não podia ver mais que

<h3>O espectro fala</h3>

<p>Várias horas se passaram, quando, de repen

Depende realmente de você, o quê, exatamente, representam os elementos envolvidos, desde que a hierarquia faça sentido. Você só precisa ter em mente algumas das melhores práticas ao criar tais estruturas:

- Preferencialmente, você deve usar apenas um `<h1>` por página — esse é o título do nível superior e todos os outros ficam abaixo dele, na hierarquia.
- Certifique-se de usar os títulos na ordem correta na hierarquia. Não use `<h3>` para representar subtítulos, seguido de `<h2>` para representar sub-subtítulos - isso não faz sentido e levará a resultados estranhos.
- Dos seis níveis de cabeçalho disponíveis, você deve usar no máximo três por página, a menos que considere necessário usar mais. Documentos com vários níveis (ou seja, uma hierarquia profunda de cabeçalho) tornam-se difíceis de navegar. Nessas ocasiões, é aconselhável espalhar o conteúdo por várias páginas, se possível.

## Por que precisamos de estrutura?

Para responder a esta pergunta, dê uma olhada em [text-start.html](#) — o ponto de partida do nosso exemplo em execução para este artigo (uma boa receita de hummus.) Você deve salvar uma cópia desse arquivo em sua máquina local, pois será necessário para os exercícios posteriores. No

momento, o corpo deste documento contém várias partes do conteúdo — elas não são marcadas de forma alguma, mas são separadas por quebras de linha (Enter/Return pressionado para ir para a próxima linha).

No entanto, quando você abre o documento em seu navegador, você verá que o texto aparece como uma só parte!

Isso ocorre porque não há elementos para dar a estrutura de conteúdo, então o navegador não sabe o que é um título e o que é um parágrafo. Além disso:

- Os usuários que olham para uma página web tendem a procurar rapidamente conteúdo relevante, muitas vezes apenas lendo os títulos para começar (normalmente passamos um tempo muito curto em uma página web ). Se eles não conseguem ver nada útil dentro de alguns segundos, eles provavelmente ficarão frustrados e irão para outro lugar.
- Os mecanismos de pesquisa que indexam sua página consideram o conteúdo dos títulos como palavras-chave importantes para influenciar os rankings de pesquisa da página. Sem cabeçalhos, sua página irá funcionar mal em termos de SEO (Search Engine Optimization).

- As pessoas com deficiência visual, muitas vezes, não leem páginas da web; elas escutam-na em vez disso. Isso é feito com um software chamado [leitor de tela](#). Este software fornece formas de obter acesso rápido a determinado conteúdo de texto. Entre as várias técnicas utilizadas, elas fornecem um esboço do documento lendo os títulos, permitindo que seus usuários encontrem as informações que precisam rapidamente.

Se os títulos não estiverem disponíveis, eles serão obrigados a ouvir todo o documento lido em voz alta.

- Para criar um estilo de conteúdo com [CSS](#), ou fazer coisas interessantes com [JavaScript](#), você precisa ter elementos que envolvam o conteúdo relevante, assim CSS/JavaScript pode efetivamente segmentá-lo.

Nós, portanto, precisamos dar marcações estruturais ao nosso conteúdo.

## Aprendizagem ativa: Fornecendo nossa estrutura de conteúdo

Vamos pular diretamente com um exemplo ao vivo. No exemplo abaixo, adicione elementos ao texto bruto no campo *Entrada* para que ele apareça como um título e dois parágrafos no campo *Saída*.

Se você cometer um erro, você sempre pode reiniciar usando o botão *Resetar*. Se você ficar preso, pressione o botão *Mostrar solução* para ver a resposta.

Por que precisamos de semântica?

A semântica está em todos os lugares — contamos com experiência anterior para nos dizer qual é a função dos

objetos do dia a dia. Quando vemos algo, sabemos qual será a sua função. Então, por exemplo, esperamos que um semáforo vermelho signifique "pare" e um semáforo verde signifique "ir". As coisas podem ficar complicadas muito rapidamente se a semântica errada for aplicada (os países usam vermelho para significar "ir"? Espero que não).

Na mesma linha, precisamos ter certeza de que estamos usando os elementos corretos, dando ao nosso conteúdo o significado, função ou aparência corretos. Nesse contexto, o elemento `<h1>` (en-US) também é um elemento semântico, que dá o texto que envolve a representação (ou significado) de "um título de nível superior em sua página".

```
| <h1>Este é um título de nível superior</h1>
```

Por padrão, o navegador fornecerá um tamanho de fonte grande para torná-lo um cabeçalho (embora você possa estilizá-lo como qualquer coisa que você quiser usando CSS). Mais importante, seu valor semântico será usado de várias maneiras, por exemplo, por mecanismos de pesquisa e leitores de tela (como mencionado acima).

Por outro lado, você pode fazer com que qualquer elemento se pareça um título de nível superior. Considere o seguinte:

```
| <span style="font-size: 32px; margin: 21px; ;
```

Este é um elemento `<span>`. Não tem semântica. Você usa-

Este é um elemento semântico. Não tem comandos. Vou usar

o para agrupar conteúdo quando deseja aplicar o CSS (ou fazer algo com JavaScript) sem dar-lhe nenhum significado extra (você saberá mais sobre isto mais tarde, no curso). Nós aplicamos alguns CSS para fazê-lo parecer um título de nível superior, mas como não tem valor semântico, ele não receberá nenhum dos benefícios extras descritos acima. É

uma boa idéia usar o elemento HTML relevante para o trabalho.

## Listas

Agora voltemos nossa atenção para as listas. As listas estão em toda parte na vida — de sua lista de compras à lista de instruções que você seguiu inconscientemente para chegar à sua casa todos os dias, para as listas das instruções que está seguindo nesses tutoriais! As listas também estão na Web e temos três tipos diferentes para prestarmos atenção.

### Não ordenada

As listas não ordenadas são usadas para marcar listas de itens para os quais a ordem dos itens não importa — vamos pegar uma lista de compras como um exemplo.

```
leite
ovos
pão
homus
```

Toda lista desordenada começa com um `<ul>` — isso envolve todos os itens da lista:

```

leite
ovos
pão
homus

```



O último passo é envolver cada item da lista em um elemento `<li>` (elemento da lista):

```

 leite
 ovos
 pão
 homus

```



Aprendizagem ativa: Marcando uma lista desordenada

Tente editar a amostra, ao vivo, abaixo para criar sua própria lista não ordenada HTML.

## Ordenada

As listas ordenadas são listas em que a ordem dos itens é importante — vamos seguir um conjunto de instruções como

exemplo:

```
Dirija até o final da estrada
Vire à direita
Vá em frente nas duas primeiras rotatórias
Vire à esquerda na terceira rotatória
A escola fica à sua direita, a 300 metros da es
```

A estrutura de marcação é a mesma das listas não ordenadas, exceto que você deve conter os itens da lista em um elemento <ol>, em vez de <ul>:

```

 Dirija até o final da estrada
 Vire à direita
 Vá em frente nas duas primeiras rotatórias
 Vire à esquerda na terceira rotatória
 A escola fica à sua direita, a 300 metros da es

```

Aprendizagem ativa: Marcando uma lista ordenada

Tente editar a amostra ao vivo abaixo, para criar sua própria lista ordenada por HTML.



## Aprendizagem ativa: marcando nossa página de receita

Então, neste ponto do artigo, você tem todas as informações necessárias para marcar nosso exemplo de página de receita. Você pode optar por salvar uma cópia local do nosso arquivo inicial de [text-start.html](#) e fazer o trabalho lá, ou

fazê-lo no exemplo editável abaixo. Fazê-lo, localmente, provavelmente será melhor, pois você conseguirá salvar o trabalho que está fazendo. Enquanto que, se o preencher no exemplo editável, ele será perdido na próxima vez que você abrir a página. Ambos têm prós e contras.

Se você ficar preso, você sempre pode pressionar o botão *Mostrar solução*, ou confira nosso exemplo de [text-complete.html](#) em nosso repositório Github.

## Aninhando listas

Não há problema em aninhar uma lista dentro de outra. Você pode ter algumas sub-listas abaixo de uma lista de nível superior. Vamos pegar a segunda lista do nosso exemplo de receita:

```

 Retire a pele do alho e pique
 Retire todas as sementes e caule da pim
 Adicione todos os ingredientes em um pr
 Processar todos os ingredientes em uma
 Se você quiser um hummus grosso "pesado"
 Se você deseja um hummus suave, process

```

Uma vez que as duas últimas listas estão intimamente relacionadas com a anterior (elas leem como sub-instruções

ou escorinhas que se encaixam abaixo dessa lista), pode fazer sentido aninhá-las dentro de sua própria lista desordenada e colocar essa lista dentro da quarta lista. Isso pareceria assim:

Como os dois últimos itens da lista estão intimamente relacionadas (eles são lidos como sub-instruções ou opções que se encaixam abaixo dessa lista), pode fazer sentido

aninha-los em sua própria lista não ordenada e colocá-los no quarto item da lista atual. Isso seria assim:

```

 Retire a pele do alho e pique
 Retire todas as sementes e caule da pi
 Adicione todos os ingredientes em um p
 Processar todos os ingredientes em uma

 Se você quiser um hummus grosso "p
 Se você deseja um hummus suave, pr


```

Tente voltar ao exemplo de aprendizagem ativo anterior e atualizar a segunda lista como está aqui.

## Ênfase e importância

Na linguagem humana, muitas vezes enfatizamos certas palavras para alterar o significado de uma frase, e muitas vezes queremos marcar certas palavras como importantes ou

vezes queremos marcar certas palavras como importantes ou diferentes de alguma forma. O HTML fornece vários elementos semânticos que nos permitem marcar o conteúdo textual com esses efeitos, e, nesta seção, vamos ver alguns dos mais comuns.

## Ênfase

Quando queremos acrescentar ênfase na linguagem falada, enfatizamos certas palavras, alterando sutilmente o significado do que estamos dizendo. Da mesma forma, em linguagem escrita tendemos a enfatizar as palavras colocando-as em itálico. Por exemplo, as seguintes duas frases têm significados diferentes.

Estou *feliz* que você não chegou *atrasado*.

Estou **feliz** que você não chegou **atrasado**.

A primeira frase parece genuinamente aliviada de que a pessoa não estava atrasada. Em contraste, a segunda parece ser sarcástica ou passiva-agressiva, expressando aborrecimento que a pessoa chegou um pouco atrasada.

Em HTML usamos o elemento de ênfase <em> para marcar tais instâncias. Além de tornar o documento mais interessante de ler, estes são reconhecidos pelos leitores de tela e falados em um tom de voz diferente. Os navegadores exibem isso como itálico por padrão, mas você não deve usar esta tag apenas para obter estilo itálico. Para fazer isso, você

usaria um elemento `<span>` e alguns CSS, ou talvez um elemento `<i>` (veja abaixo).

```
| <p>Eu estou feliz você não está
```

## Importância

Para enfatizar palavras importantes, tendemos a enfatizá-las na linguagem falada e colocá-la em **negrito** na linguagem escrita. Por exemplo:

Este líquido é **altamente tóxico**.

Eu estou contando com você. **Não** se atrasa!

Em HTML usamos o elemento `<strong>` (importância) para marcar tais instâncias. Além de tornar o documento mais útil, novamente estes são reconhecidos pelos leitores de tela e falados em um tom de voz diferente. Os navegadores exibem este texto como negrito por padrão, mas você não deve usar esta marca apenas para obter um estilo negrito. Para fazer isso, você usaria um elemento `<span>` e algum CSS, ou talvez um `<b>` (veja abaixo).

```
| <p>Este líquido é altamente tóxic
```

```
| <p>Estou contando com você. Não</stro
```

Você pode aninhar importância e ênfase entre si, se desejar:

<p>Este líquido é **altamente tóxico**. Se você beber, **você pode** *morrer* /

## Aprendizagem ativa: Vamos ser importantes!

Nesta seção de aprendizado ativo, fornecemos um exemplo editável. Nele, gostaríamos que você tentasse adicionar ênfase e importância às palavras que acha que precisam delas, apenas para praticar um pouco.

## Itálico, negrito, sublinhado...

Os elementos que discutimos até agora têm clara semântica associada. A situação com `<b>`, `<i>`, e com `<u> (en-US)` é um pouco mais complicada. Eles surgiram para que as pessoas pudessem escrever negrito, itálico ou texto sublinhado em uma época em que o CSS ainda era pouco suportado. Elementos como este, que apenas afetam a apresentação e não a semântica, são conhecidos como **elementos de apresentação** e não devem mais ser usados, porque, como já vimos, a semântica é importante para a acessibilidade, SEO, etc.

O HTML5 redefiniu `<b>`, `<i>` e `<u>` com novas funções semânticas um pouco confusas.

Aqui está a melhor regra geral: provavelmente é apropriado usar `<b>`, `<i>` ou `<u>` para transmitir um significado tradicionalmente transmitido com negrito, itálico ou sublinhado, desde que não exista um elemento mais adequado. No entanto, sempre é essencial manter uma mentalidade de acessibilidade. O conceito de itálico não é muito útil para pessoas que usam leitores de tela ou para pessoas que usam um sistema de escrita diferente do

alfabeto Latino.

- <i> é usado para transmitir um significado tradicionalmente transmitido por itálico: palavras estrangeiras, designação taxonômica, termos técnicos, um pensamento...
- <b> é usado para transmitir um significado tradicionalmente transmitido por negrito: palavras-chave, nomes de produtos, sentença principal...
- <u> (en-US) é usado para transmitir um significado tradicionalmente transmitido pelo sublinhado: nome próprio, erro de ortografia...

Um aviso amável sobre o sublinhado: **as pessoas associam fortemente o sublinhado com hiperlinks**. Portanto, na Web, é melhor sublinhar apenas os links. Use o elemento <u> quando for semanticamente apropriado, mas considere usar o CSS para alterar o sublinhado padrão para algo mais apropriado na Web. O exemplo abaixo ilustra como isso pode ser feito.

```
<!-- nomes científicos -->
<p>
 O Colibri Ruby-throated (<i>Archilochus col
 é o colibri mais comum do Leste da América
</p>
```

```
<!-- palavras estrangeiras -->
<p>
 O menu era um mar de palavras exóticas como
 <i lang="id">nasi goreng</i> e <i lang="fr">
</p>

<!-- um erro de ortografia conhecido -->
<p>
 Algum dia eu vou aprender como <u style="te
</p>

<!-- Destaque as palavras-chave em um conjunt

 Fatie dois pedaços de pão do pão.

 Colocar uma fatia de tomate e uma
 alface entre as fatias de pão.


```

## Teste suas habilidades!

Você chegou ao final deste artigo, mas consegue se lembrar das informações mais importantes? Você pode encontrar alguns testes adicionais para verificar se você absorveu essas informações antes de prosseguir — consulte [Teste suas habilidades: noções básicas de texto HTML](#).

Por enquanto é isso! Este artigo deve fornecer uma boa idéia de como começar a marcar texto em HTML e apresentar alguns dos elementos mais importantes nessa área. Há muito mais elementos semânticos a serem abordados nesta área, e veremos muito mais em nosso artigo de [Formatação avançada de texto](#), mais adiante neste curso. No próximo artigo, veremos detalhadamente como [criar links](#), possivelmente o elemento mais importante na Web.

**Last modified:** 1 de fev. de 2022, [by MDN contributors](#)

This page was translated from English by the community. Learn more and join the MDN Web Docs community.

## Criando hyperlinks

Os hiperlinks são realmente importantes — são o que torna a Web uma *web*. Este artigo mostra a sintaxe necessária para criar um link e discute as suas melhores práticas.

<b>Pre-requisitos:</b>	Familiaridade básica em HTML, conforme <a href="#">Começando com o HTML</a> . Formatação de texto em HTML, conforme <a href="#">Fundamentos do texto em HTML</a> .
<b>Objetivo:</b>	Para aprender a implementar um hiperlink efetivamente e vincular vários arquivos juntos.

## O que é um hiperlink?

Os hiperlinks são uma das inovações mais interessantes que a Web oferece. Bem, eles são uma característica da Web desde o início, mas são o que torna a Web como ela é — eles nos permitem vincular nossos documentos a qualquer outro documento (ou outro recurso) que queremos. Também podemos vincular para partes específicas de documentos e podemos disponibilizar aplicativos em um endereço web simples (em contraste com aplicativos nativos, que devem ser

instalados e tantas outras coisas). Qualquer conteúdo da web pode ser convertido em um link, para que, quando clicado (ou ativado de outra forma) fará com que o navegador vá para outro endereço ([URL](#)).

**Nota:** Um URL pode apontar para arquivos HTML, arquivos de texto, imagens, documentos de texto, arquivos de vídeo e áudio e qualquer outra coisa que possa estar na Web. Se o navegador não souber exibir ou manipular o arquivo, ele perguntará se você deseja abrir o arquivo (nesse caso, o dever de abrir ou manipular o arquivo é passado para um aplicativo nativo adequado no dispositivo) ou fazer o download dele (nesse caso, você pode tentar lidar com isso mais tarde).

A página inicial da BBC, por exemplo, contém um grande número de links que apontam não apenas para várias notícias, mas também diferentes áreas do site (funcionalidade de navegação), páginas de login/registro (ferramentas do usuário) e muito mais.

## Welcome to the BBC

Tuesday, 2 February

Sign in with your BBC ID, or Register to

see Weather, Local News and more



Customise your Homepage



FOOTBALL



US &amp; CANADA



ENTERTAINMENT &amp; ARTS

Latest news &gt;

Ted Cruz wins Iowa  
Republican vote

US &amp; CANADA



Ted Cruz victorious in Iowa

US ELECTION 2016

EU law 'veto powers' to be  
unveiled

Latest sport &gt;

January spending at five-year  
high

FOOTBALL

Mayweather offered 'crazy  
numbers'

BOXING

Which clubs outspent La  
Liga?

## Anatomia de um link

Um link básico é criado envolvendo o texto (ou outro conteúdo, veja [Block level links](#)) que você quer transformar em um link dentro de um elemento `<a>`, e dando-lhe um atributo `href`, (também conhecido como **Hypertext Reference**, ou **target**) que conterá o endereço da Web para o qual você deseja que o link aponte.

```
<p>Estou criando um link para
a página inicial da Mozilla
</p>
```

Isso nos dá o seguinte resultado:

Estou criando um link para [a página inicial da Mozilla](https://www.mozilla.org/pt-BR/).

## Adicionando informações de suporte com o atributo *title*

Outro atributo que você pode querer adicionar aos seus links é o *title*; pretende-se que ele contenha informações úteis adicionais sobre o link, como, que tipo de informação a página contém ou informações importantes. Por exemplo:

```
<p>Estou criando um link para
<a href="https://www.mozilla.org/pt-BR/"
 title="O melhor lugar para encontrar mais informaç</p>
```

Isto nos dá o seguinte resultado (o título aparecerá como uma dica de ferramenta quando o link estiver suspenso):

Estou criando um link para a página inicial da Mozilla.

**Nota:** Um título de link só é revelado ao passar o mouse sobre ele, o que significa que as pessoas que dependem do teclado ou *touchscreen* para navegar em páginas web terão dificuldade em acessar a informação do título. Se a informação de um título é realmente importante para a usabilidade da página, então você deve apresentá-la de uma maneira que será acessível a todos os usuários, por exemplo, colocando-o no texto normal.

Aprendizagem na prática: criando seu próprio link de exemplo

Momento da aprendizagem na prática: gostaríamos que você criasse um documento HTML usando seu editor de código local (nossa modelo inicial seria interessante.)

- Dentro do corpo do HTML, tente adicionar um ou mais parágrafos ou outros tipos de conteúdo que você já conhece.
- Transforme alguns dos conteúdos em links.
- Inclua atributos de título.

## Links de nível de bloco

Como falamos anteriormente, você pode transformar qualquer conteúdo em um link, mesmo elementos de nível de bloco. Se você tiver uma imagem que queira transformar em um link, você pode simplesmente colocar a imagem entre as tags `<a></a>`.

```



```

**Nota:** Você descobrirá muito mais sobre o uso de imagens na Web em artigo posterior.

## Um guia rápido sobre URLs e caminhos

Para entender completamente os destinos de links, você precisa entender URLs e caminhos de arquivos. Esta seção fornece as informações que você precisa para conseguir isso.

Um URL ou *Uniform Resource Locator* é simplesmente uma sequência de texto que define onde algo está localizado na Web. Por exemplo, a página inicial em inglês da Mozilla está localizada em `https://www.mozilla.org/en-US/`.

Os URLs usam caminhos para encontrar arquivos. Os caminhos especificam onde, no explorador de arquivos, o recurso que você está

interessado está localizado. Vejamos um exemplo simples de uma estrutura de diretório (veja o diretório de [criação de hiperlinks](#) ).

A raiz dessa estrutura de diretório é chamada de [criação de hiperlinks](#). Ao trabalhar localmente com um site, você terá um diretório no qual ele todo está dentro. Incluído na raiz, temos um arquivo `index.html` e um arquivo `contacts.html`. Em um site real, `index.html` seria nossa página inicial ou página de entrada (uma página da web que serve como ponto de entrada para um site ou uma seção específica de um site).

Existem também dois diretórios dentro da nossa raiz — `pdfs` e `projects`. Cada um deles contém um único arquivo — um PDF (`projetos-brief.pdf`) e um arquivo `index.html`, respectivamente. Observe como é possível, felizmente, ter dois arquivos `index.html` em um projeto, desde que estejam em locais diferentes no sistema de arquivos. Muitos sites fazem isso. O segundo `index.html` poderia ser a página de destino principal para informações relacionadas ao projeto.

- **Mesmo diretório:** se você deseja incluir um hiperlink dentro de `index.html` (o `index.html` de nível superior) apontando para `contacts.html`, basta especificar o nome do arquivo ao qual deseja vincular, já que está no mesmo diretório que o arquivo atual. Portanto, o URL que você usaria seria `contacts.html`:

```
|<p>Deseja entrar em contato com um membro da 1
|Encontre detalhes sobre nossos serviços em n 2
```

- **Movendo-se para baixo em subdiretórios:** se você quisesse incluir um hiperlink dentro do `index.html` apontando para o projeto/`index.html`, você precisaria descer no diretório de projetos antes de indicar o arquivo que deseja vincular. Isso é

feito especificando o nome do diretório, depois uma barra inclinada e, em seguida, o nome do arquivo. Então o URL que você usaria seria `projeto/index.html`:

```
|<p>Visite minha <a href="projects/index.html"
```

- **Movendo-se de volta para os diretórios pai:** se você quisesse incluir uma hiperlink dentro de `projeto/index.html` apontando para `pdfs/projetos-brief.pdf`, você precisaria subir um nível de diretório e voltar para o diretório `pdf`. "Subir um diretório" é indicado usando dois pontos — .. — então a URL que você usaria seria `../pdfs/project-brief.pdf`

```
|<p>Um link para o meu <a href="..../pdfs/proje
```

**Nota:** Você pode combinar várias instâncias desses recursos em URLs complexas, se necessário, por exemplo `../../../../complex/path/to/my/file.html`.

## Fragments de documento

É possível vincular a uma parte específica de um documento HTML (conhecido como um **fragmento de documento**) e não apenas ao topo do documento. Para fazer isso, primeiro você deve atribuir um atributo "id" ao elemento ao qual deseja vincular. Normalmente faz sentido vincular a um título específico, então ficaria algo do tipo:

```
|<h2 id="Mailing_address">Endereço de correspondê
```

Em seguida, para vincular a esse `id` específico, você o incluirá no final do URL, precedido por um símbolo de hashtag/cerquilha, por exemplo:

<p>Quer escrever uma carta? Use nosso<a href="co:...>

Você pode até usar apenas referência de fragmento do documento por si só para vincular a outra parte do mesmo documento:

<p>O <a href="#Mailing\_address">endereço postal

## URLs absolutos versus relativos

Dois termos que você encontrará na Web são URL **absoluto** e URL **relativo**:

**URL absoluto:** aponta para um local definido por sua localização absoluta na web, incluindo "protocolo" e "nome de domínio". Então, por exemplo, se uma página `index.html` for carregada para um diretório chamado `projeto` que fica dentro da raiz de um servidor web, e o domínio do site é `http://www.exemplo.com`, a página estará disponível em

`http://www.exemplo.com/projeto/index.html` (ou mesmo apenas `http://www.exemplo.com/projeto/`, pois a maioria dos servidores web apenas procura uma página de destino como `index.html` para carregar, se não está especificado no URL.)

Um URL absoluto sempre aponta para o mesmo local, não importa onde seja usado.

**URL relativa:** aponta para um local *relativo* ao arquivo do qual você está vinculando, mais como o que vimos na seção anterior. Por exemplo, se desejássemos vincular nosso arquivo de exemplo em `http://www.exemplo.com/projeto/index.html` para um arquivo PDF no mesmo diretório, o URL seria apenas o nome do arquivo — por exemplo, `project-brief.pdf` — nenhuma informação extra é necessária. Se o PDF estava disponível em um subdiretório dentro de `projects` chamado `pdfs`, o link relativo seria

`pdfs/projeto-brief.pdf` (o URL absoluto equivalente seria `http://www.example.com/projects/pdfs/project-brief.pdf`).

Um URL relativo apontará para lugares diferentes, dependendo da localização real do arquivo ao qual você se refere — por exemplo, se tivermos movido nosso arquivo `index.html` para fora do diretório de projetos e para a raiz do site (no nível superior, não em qualquer diretório), o link relativo à URL referente a

`pdfs/project-brief.pdf` agora apontaria para um arquivo localizado em

`http://www.example.com/pdfs/project-brief.pdf`, não para um arquivo localizado em

`http://www.example.com/projects/pdfs/project-brief.pdf`.

## Práticas recomendadas

Existem algumas práticas recomendadas a seguir, ao escrever links. Vejamos.

### Use palavras-chave claras

É muito fácil jogar links na sua página, porém somente isto não é suficiente. Precisamos tornar nossos links *acessíveis* a todos os leitores, independentemente do contexto atual e de quais ferramentas eles prefiram. Por exemplo:

- Os usuários de leitores de telas gostam pular de link a outro link na página e ler links fora do contexto.
- Os motores de busca usam o texto do link para indexar arquivos de destino, por isso é uma boa idéia incluir palavras-chave no texto do link para descrever efetivamente o que está sendo vinculado.

- Os usuários normalmente deslizam sobre a página em vez de ler cada palavra, e são atraídos para recursos de página que se destacam, como links. Eles acharão os textos descritivos de links úteis.

Vejamos um exemplo específico:

*Texto de link correto:* [Baixe o Firefox](https://firefox.com/)

```
<p>
 Baixe o Firefox
</p>
```

*Texto de link incorreto:* [clique aqui](https://firefox.com/) para baixar o Firefox

```
<p>
 clique aqui

 para baixar o Firefox</p>
```

Outras dicas:

- Não repita o URL como parte do texto do link — Os URLs parecem feios e até são mais feios quando um leitor de tela os lê letra por letra.
- Não diga "link" ou "links para" no texto do link — é apenas ruído. Os leitores de tela já dizem às pessoas que existe um link. Os usuários visuais também sabem que existe um link, porque eles geralmente são de cor diferente e sublinhados (esta convenção geralmente não deve ser quebrada, pois os usuários estão muito acostumados a isso).
- Mantenha seu rótulo de link o mais curto possível — links longos irritam especialmente os usuários de leitores de tela, que têm que ouvir o texto inteiro lido.

- Minimize instâncias em que o mesmo texto esteja ligado a diferentes lugares. Isso pode causar problemas para os usuários do leitor de tela, que muitas vezes mostrará uma lista dos links fora do contexto — vários links todos rotulados como "clique aqui", "clique aqui", "clique aqui"... seria confuso.

## Use links relativos sempre que possível

A partir da descrição acima, você pode pensar que é uma boa idéia usar apenas links absolutos o tempo todo; Afinal, eles não quebram quando uma página é movida como pode ocorrer com *links relativos*. No entanto, você deve usar *links relativos* sempre que possível ao vincular a outros locais dentro do mesmo site (ao vincular a outro site, você precisará usar um link absoluto):

- Para começar, é muito mais fácil verificar seu código — os URLs relativos geralmente são muito mais curtos que os URLs absolutos, o que torna o código de leitura muito mais fácil.
- Em segundo lugar, é mais eficiente usar URLs relativas sempre que possível. Quando você usa um URL absoluto, o navegador começa procurando a localização real do servidor no Servidor de Nomes de Domínio "DNS"; veja [Como a web funciona](#) para obter mais informações), então ele vai para esse servidor e encontra o arquivo que está sendo solicitado. Por outro lado, com um URL relativo, o navegador apenas procura o arquivo que está sendo solicitado, no mesmo servidor. Então, se você usa URLs absolutos para fazer o que os URLs relativos fariam, você está constantemente fazendo o seu navegador realizar trabalho extra, o que significa que ele irá executar de forma menos eficiente.

## Vinculando-se a recursos que não sejam HTML — deixe indicadores claros

Ao vincular a um arquivo que será baixado (como um documento PDF ou Word) ou transmitido (como vídeo ou áudio) ou ainda tiver outro efeito potencialmente inesperado (abrir uma janela pop-up ou carregar um filme Flash), você deve adicionar uma redação clara para reduzir qualquer confusão. Pode ser bastante irritante, por exemplo:

- Se você estiver em uma conexão de baixa banda larga, clicar em um link e, em seguida, um download de muitos megabytes começa inesperadamente.
- Se você não tiver instalado o Flash Player, clicar em um link e, de repente, ser levado para uma página que requer Flash Player.

Vejamos alguns exemplos, para ver que tipo de texto pode ser usado aqui:

```
<p>
 Baixe o relatório de vendas (PDF, 10MB)
</p>

<p>
 Assista ao vídeo (o fluxo abre em separado, qualida
</p>

<p>
 Jogue o jogo de carro (requer Flash Player)
</p>
```

Use o atributo de download ao vincular a um download

Quando você está apontando para um arquivo que deve ser baixado em vez de ser aberto no navegador, poderá usar o atributo de download para fornecer um nome de arquivo salvo padrão. Aqui está

um exemplo, com um link de `baixar` para a versão do Windows 39 do Firefox:

```

 Faça o download do Firefox 39 para Windows

```

## Aprendizagem ativa: criando um menu de navegação

Para este exercício, gostaríamos que você vinculasse algumas páginas a um menu de navegação para criar um *site* com várias páginas. Essa é uma maneira comum de criá-los — a mesma estrutura de página é usada em todas elas, incluindo o mesmo menu de navegação. Portanto, quando os *links* são clicados, dá a impressão de que você permanece no mesmo lugar e que um conteúdo diferente está sendo criado.

Você precisará fazer cópias locais das quatro páginas a seguir, tudo no mesmo diretório (veja também o diretório de [início do menu de navegação](#) para uma lista completa de arquivos):

- [index.html](#)
- [projects.html](#)
- [pictures.html](#)
- [social.html](#)

Você deve:

1. Adicionar uma lista não ordenada no local indicado em uma página, contendo os nomes das páginas a serem vinculadas. Um

menu de navegação geralmente é apenas uma lista de *links*, então está semanticamente correto.

2. Transformar o nome de cada página em um *link* para essa página.
3. Copiar o menu de navegação para cada uma.
4. Em cada página, remova apenas o *link* para a mesma página - é confuso e inútil que uma página inclua um link para si mesma, e a falta de um *link* é um bom lembrete visual de qual página você está atualmente.

O exemplo final acabaria por parecer algo assim:

**Note:** Se você ficar preso, ou não tem certeza se o fez bem, você pode verificar o diretório de navegação-menu-marcado para ver a resposta correta.

## Links de e-mail

É possível criar *links* ou botões que, quando clicados, abrem uma nova mensagem de e-mail de saída em vez de vincular a um recurso ou página. Isso é feito usando o elemento `<a>` e o `mailto:` estrutura de URL.

Na sua forma mais comum, um `mailto:` simplesmente indica o endereço de e-mail do destinatário pretendido. Por exemplo:

```
Enviar email
```

Isso resulta em um *link* que se parece com isto: Enviar e-mail para lugar nenhum.

Na verdade, o endereço de e-mail é opcional. Se você deixar de fora (ou seja, seu [href](#) for simplesmente "mailto:"), uma nova janela de e-mail de saída será aberta pelo aplicativo de e-mail do usuário sem um destinatário. Isso geralmente é útil como "Compartilhar" *links* que os usuários podem clicar para enviar um e-mail para um endereço escolhido.

## Especificando detalhes

Além do endereço de e-mail, você pode fornecer outras informações. Na verdade, qualquer campo de cabeçalho de correio padrão pode ser adicionado ao URL do `mailto:` que você fornece. Os mais utilizados são "assunto", "cc" e "corpo" (que não é um campo de cabeçalho verdadeiro, mas permite que você especifique uma mensagem de conteúdo curta para o novo e-mail). Cada campo e seu valor são especificados como um termo de consulta.

Aqui está um exemplo que inclui um cc, bcc, assunto e corpo de texto:

```
<a href="mailto:nowhere@mozilla.org?cc=name2@rap...&bcc=...&subject=...&body=...
```

**Nota:** Os valores de cada campo devem ser codificados por URL, ou seja, com caracteres não imprimíveis (caracteres invisíveis, como guias, carriage returns e quebras de página) e espaços com percent-escaped. Observe também o uso do ponto de interrogação (?) Para separar o URL principal dos valores do campo e do e comercial (&) para separar cada campo no `mailto:` URL. Essa é a notação de consulta padrão do URL. Leia O método GET para entender para que esta notação de consulta é mais comum.

Aqui estão alguns outros exemplos de URLs de `mailto`:

- <mailto:>
- <mailto:nowhere@mozilla.org>
- <mailto:nowhere@mozilla.org,nobody@mozilla.org>
- <mailto:nowhere@mozilla.org?cc=nobody@mozilla.org>
- [mailto:nowhere@mozilla.org?  
cc=nobody@mozilla.org&subject=This%20is%20the%20subject](mailto:nowhere@mozilla.org?cc=nobody@mozilla.org&subject=This%20is%20the%20subject)

## Resumo

Por enquanto isto é tudo sobre links! Você retornará aos links mais tarde no curso quando começar a analisar o estilo deles. Em seguida, em HTML, retornaremos à semântica de texto e veremos alguns recursos mais avançados/incomuns que você achará úteis — No próximo artigo você verá a formatação avançada de texto.

**Last modified:** 31 de jan. de 2022, by [MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Formatação avançada de texto

Existem muitos outros elementos em HTML para formatação de texto, que não tratamos no artigo de [Fundamentos do texto em HTML](#). Os elementos descritos neste artigo são menos conhecidos, mas ainda são úteis para conhecer (e isso ainda não é uma lista completa de todos os elementos). Aqui, você aprenderá a marcar citações, listas de descrição, código de computador e outros textos relacionados, subscrito e sobrescrito, informações de contato e muito mais.

<b>Pré-requisitos:</b>	Familiaridade básica em HTML, conforme abordado em <a href="#">Introdução ao HTML</a> . Formatação de texto em HTML, conforme abordado em <a href="#">Fundamentais de texto em HTML</a> .
<b>Objetivo:</b>	Aprender a usar elementos HTML menos conhecidos para marcar recursos semânticos avançados.

# Listas de descrição

Nos Fundamentos do texto em HTML, falamos sobre como marcar as listas básicas em HTML, mas não mencionamos o terceiro tipo de lista que ocasionalmente irá encontrar - listas de descrição. O objetivo dessas listas é marcar um conjunto de itens e suas descrições associadas, como termos e definições, ou perguntas e respostas. Vejamos um exemplo de um conjunto de termos e definições:

solilóquio

No drama, onde um personagem fala a si mesmo,  
monólogo

No drama, onde um personagem fala seus pensamentos  
aparte

No drama, onde um personagem compartilha um c

As listas de descrição usam um invólucro diferente dos outros tipos de lista — <dl>; além disso, cada termo está envolvido em um <dt> (termo de descrição) elemento, e cada descrição está envolvida em um <dd> (definição de descrição) elemento. Vamos terminar marcando nosso exemplo:

```
<dl>
 <dt>solilóquio</dt>
 <dd>No drama, onde um personagem fala a si
 <dt>monólogo</dt>
 <dd>No drama, onde um personagem fala seus
```

```
<dt>aparte</dt>
<dd>No drama, onde um personagem compartilha
</dd>
```

Os estilos padrões do navegador exibirão as listas com as descrições indentadas um pouco dos termos. Os estilos da MDN seguem esta convenção de forma bastante parecida, mas também enfatizam os termos, para uma definição extra.

### **solilóquio**

No drama, onde um personagem fala a si mesmo, representando seus pensamentos ou sentimentos internos e no processo, transmitindo-os ao público (mas não a outros personagens).

### **monólogo**

No drama, onde um personagem fala seus pensamentos em voz alta para compartilhá-los com o público e com outros personagens presentes.

### **aparte**

No drama, onde um personagem compartilha um comentário apenas com o público para efeito humorístico ou dramático. Isso geralmente é um sentimento, pensamento ou parte de informação de fundo adicional.

Observe que é permitido ter um único termo com múltiplas descrições, por exemplo:

### **aparte**

No drama, onde um personagem compartilha um comentário apenas com o público para efeito humorístico ou dramático. Isso geralmente é um sentimento, pensamento ou parte de informação de fundo adicional.

Por escrito, uma seção de conteúdo que está relacionada ao tópico atual, mas não se encaixa diretamente no fluxo principal de conteúdo, então é apresentado próximo (muitas vezes em uma caixa ao lado).

## Aprendizagem ativa: marcando um conjunto de definições

É hora de pôr as mãos nas listas de descrição. Adicione elementos ao texto bruto no campo de *Entrada* para que ele se pareça como uma lista de descrição no campo *Saída*. Você pode tentar usar seus próprios termos e descrições, se quiser.

Se você cometer um erro, sempre pode reiniciá-lo usando o botão 'Limpar'. Se ficar realmente preso, pressione o botão '*Mostrar solução*' para ver a resposta.

## Entrada

<p>Hello and welcome to my motivation page.  
As Confucius once said:</p>

<p>It does not matter how slowly you go as long as you do not stop.</p>

<p>I also love the concept of positive thinking, and The Need To Eliminate Negative Self Talk  
(as mentioned in Affirmations for Positive Thinking)</p>

## Saída

Hello and welcome to my motivation page. As Confucius once said:

It does not matter how slowly you go as long as you do not stop.

I also love the concept of positive thinking, and The Need To Eliminate Negative Self Talk (as mentioned in Affirmations for Positive Thinking )

# Cotações

HTML também possui recursos disponíveis para marcação de cotações. Qual elemento você pode usar? Depende se está marcando um bloco ou uma cotação em linha.

## Blockquotes

Se uma seção de conteúdo em nível de bloco (seja um parágrafo, vários parágrafos, uma lista, etc.) for citada em algum outro lugar, você deverá envolvê-la em um elemento `<blockquote>` para indicar isso e incluir um URL apontando para a fonte da citação dentro de um atributo `cite`. Por exemplo, a marcação a seguir é obtida da página do elemento `<blockquote>` do MDN:

```
<p>The HTML <code><blockquote</blockquote> Quotation Element) indicates that th n
```

Para transformar isso em uma cotação em bloco, faríamos assim:

```
<blockquote cite="/en-US/docs/Web/HTML/El < p>The HTML <code><blockquote</blockquote> Quotation Element) indicates that the </blockquote>
```

O estilo padrão do navegador renderiza isso como um parágrafo recuado, como um indicador de que é uma citação. O MDN faz isso, mas também, adiciona um estilo extra:

O Elemento HTML **<blockquote>** (or *HTML Block Quotation Element*) indica que o texto em anexo é uma cotação estendida.

## Cotações em linha

As cotações embutidas funcionam exatamente da mesma maneira, exceto pelo uso do elemento **<q>**. Por exemplo, o bit de marcação abaixo contém uma cotação da página MDN **<q>**:

`<p>The quote element – <code>&lt;q&gt;</code> for short quotations that don't require p </p>`

O estilo padrão do navegador renderiza isso como texto normal entre aspas para indicar uma cotação, assim:

O elemento de cotação — **<q>** — é "destinado a citações curtas que não exigem quebras de parágrafo".

## Citações

O conteúdo do atributo **cite** parece útil, mas, infelizmente, navegadores, leitores de tela etc. não fazem muito uso dele. Não há como fazer com que o navegador exiba o conteúdo de **cite**, sem escrever sua própria solução usando JavaScript ou CSS. Se você deseja disponibilizar a fonte da cotação na página, uma maneira melhor de marcá-la é

colocar o elemento `<cite>` próximo ao elemento quote. Isso realmente tem o objetivo de conter o nome da fonte da citação — ou seja, o nome do livro ou o nome da pessoa que disse a citação — mas não há razão para que você não possa vincular o texto dentro de `<cite>` à citação fonte de alguma forma:

```
<p>According to the MDN blockquote page:</p>

<blockquote cite="/en-US/docs/Web/HTML/Element/Block_Quotes">
 <p>The HTML <code><blockquote></blockquote> Quotation Element) indicates that the</blockquote>

<p>The quote element – <code><q></code> for short quotations that don't require paragraphing</p>
 <code><q></code> MDN q page.</p>
```

As citações são estilizadas em fonte itálica por padrão. Você pode ver esse código funcionando em nosso exemplo [quotations.html](#).

## Aprendizado ativo: quem disse isso?

Hora de outro exemplo de aprendizado ativo! Neste exemplo, gostaríamos que você:

1. Transforme o parágrafo do meio em uma citacão em

... transformando o parágrafo do meio em uma cotação ...

- bloco, que inclui um atributo `cite`.
2. Transforme parte do terceiro parágrafo em uma cotação embutida, que inclui um atributo de `cite`.
  3. Inclua um elemento `<cite>` para cada link.

Pesquise on-line para encontrar fontes de cotação apropriadas.

Se você cometer um erro, sempre poderá redefini-lo usando o botão 'Limpar'. Se você realmente ficar atolado, pressione o botão 'Mostrar solução' para ver a resposta.

## Abreviações

Outro elemento bastante comum que você encontrará ao olhar na Web é o <abbr> — usado para contornar uma abreviação ou sigla e fornecer uma expansão completa do termo (incluído em um atributo title.) Vejamos alguns exemplos

```
<p>Usamos <abbr title="Hypertext Markup Language">HTML</abbr>
<p>Acho que o <abbr title="Reverendo">Rev.</abbr> fez isso na cozinha com a motosserra.
```

Elas aparecerão da seguinte forma (a expansão aparecerá em uma dica de ferramenta quando o termo passar o mouse):

Usamos HTML para estruturar nossos documentos da web.

Acho que o Rev. Green fez isso na cozinha com a motosserra.

**Note:** Há outro elemento, <acronym>, que basicamente faz a mesma coisa que <abbr>, e foi projetado

especificamente para acronymos, em vez de abbreviações. Isso, no entanto, caiu em desuso — não era suportado em navegadores nem o `<abbr>`, e tem uma função semelhante que foi considerado inútil ter os dois. Apenas use `<abbr>`.

## Aprendizado ativo: marcando uma abreviação

Para esta tarefa simples de aprendizado ativo, gostaríamos que você simplesmente marque uma abreviação. Você pode usar nossa amostra abaixo ou substituí-la por uma de sua preferência.

## Marcando detalhes de contato

O HTML possui um elemento para marcar os detalhes do contato — `<address>`. Isso simplesmente envolve seus detalhes de contato, por exemplo:

```
<address>
 <p>Chris Mills, Manchester, The Grim Nc
</address>
```

Porém, uma coisa a se lembrar é que o elemento `<address>` destina-se a marcar os detalhes de contato da pessoa que escreveu o documento HTML e não qualquer endereço. Portanto, o exposto acima só seria bom se Chris tivesse escrito o documento em que a marcação aparece. Observe, que, algo assim também seria bom:

```
<address>
 <p>Page written by <a href="..../authors/ i
</address>
```

## Sobrescrito e subscrito

Ocasionalmente, você precisará usar sobrescrito e subscrito ao marcar itens como datas, fórmulas químicas e equações matemáticas para que eles tenham o significado correto. Os elementos `<sup>` (en-US) e `<sub>` (en-US) manipulam esse trabalho. Por exemplo:

```
<p>My birthday is on the 25th M
<p>Caffeine's chemical formula is C₈H₁₀N₄O₂ u
<p>If $x^{²}$ is 9, x must equal 3 or -
```

A saída desse código é assim:

Meu aniversário é no dia 25 de maio de 2001.

A fórmula química da cafeína é C<sub>8</sub>H<sub>10</sub>N<sub>4</sub>O<sub>2</sub>.

Se  $x^2$  é 9, x deve ser igual a 3 ou -3.

## Representando código de computador

Existem vários elementos disponíveis para marcar código de computador usando HTML:

- `<code>` : Para marcar partes genéricas de código de computador.
- `<pre>` : Para reter espaço em branco (geralmente blocos de código) — se você usar recuo ou espaço em branco em excesso no seu texto, os navegadores o ignorarão e você não o verá na sua página renderizada. Se você envolver o texto nas tags `<pre></pre>` seu espaço em branco será renderizado de forma idêntica à maneira como você o vê no seu editor de texto.
- `<var>` : Para marcar especificamente nomes de variáveis.

- <kbd> (en-US): Para marcar a entrada do teclado (e outros tipos) inserida no computador.
- <samp> (en-US): Para marcar a saída de um programa de computador.

Vejamos alguns exemplos. Você deve tentar brincar com eles (tente pegar uma cópia do nosso arquivo de exemplo

[other-semantics.html](#) ):

```
<pre><code>var para = document.querySelector('div');
para.onclick = function() {
 alert('Owww, stop poking me!');
}</code></pre>

<p>You shouldn't use presentational elements

<p>In the above JavaScript example, <var>para

<p>Select all the text with <kbd>Ctrl</kbd>/<kbd>Shift</kbd></p>

<pre>$ <kbd>ping mozilla.org</kbd>
<samp>PING mozilla.org (63.245.215.20): 56 da
64 bytes from 63.245.215.20: icmp_seq=0 ttl=4</samp></pre>
```

O código acima terá a seguinte aparência:

## Marcando horários e datas

O HTML também fornece o elemento `<time>` para marcar horários e datas em um formato legível por máquina. Por exemplo:

```
| <time datetime="2016-01-20">20 January 20 /
```

Por que isso é útil? Bem, existem muitas maneiras diferentes pelas quais os humanos escrevem datas. A data acima pode ser escrita como:

- 20 January 2016
- 20th January 2016
- Jan 20 2016
- 20/01/16
- 01/20/16
- The 20th of next month
- 20e Janvier 2016
- 2016年1月20日
- And so on

Mas essas formas diferentes não podem ser facilmente reconhecidas pelos computadores — e se você quiser pegar automaticamente as datas de todos os eventos em uma página e inseri-las em um calendário? O elemento `<time>` permite anexar uma data/hora inequívoca e legível por máquina para esse fim.

O exemplo básico acima fornece apenas uma data legível por máquina simples, mas existem muitas outras opções possíveis, por exemplo:

```
| <!-- Data simples padrão -->
```

```
<time datetime="2016-01-20">20 January 20 /
<!-- Apenas ano e mês -->
<time datetime="2016-01">January 2016</time>
<!-- Just month and day -->
<time datetime="01-20">20 January</time>
<!-- Apenas tempo, horas e minutos -->
<time datetime="19:30">19:30</time>
<!-- Você pode fazer segundos e milissegundos

<time datetime="19:30:01.856">19:30:01.856</t
<!-- Data e hora -->
<time datetime="2016-01-20T19:30">7.30pm, 20
<!-- Data e hora com compensação de fuso horá
<time datetime="2016-01-20T19:30+01:00">7.30p
<!-- Chamando um número de semana específico
<time datetime="2016-W04">The fourth week of
```

## Resumo

Isso marca o fim de nosso estudo da semântica de texto HTML. Lembre-se de que o que você viu durante este curso não é uma lista exaustiva de elementos de texto HTML — queríamos tentar cobrir o essencial, e alguns dos mais comuns que você verá na natureza, ou pelo menos podem achar interessantes. Para encontrar muito mais elementos HTML, você pode dar uma olhada no nosso [HTML element reference](#) (a seção [Inline text semantics](#) seria um ótimo ponto de partida.) No próximo artigo, examinaremos os elementos HTML que você usaria para estruturar as diferentes partes de um documento HTML.

# Neste módulo

- [Introdução ao HTML](#)
- [O que tem na cabeça? Metadados em HTML](#)
- [Fundamentos de texto HTML](#)
- [Criando hiperlinks](#)
- [Formatação avançada de texto](#)
- [Estrutura de documentos e sites](#)
- [Depurando HTML](#)
- [Marcando uma carta](#)
- [Estruturando uma página de conteúdo](#)

**Last modified:** 31 de jan. de 2022, [by MDN contributors](#)



[This page was translated from English by the community.](#)  
[Learn more and join the MDN Web Docs community.](#)

## Estrutura de documento e sites

Além de definir as partes individuais de sua página (como "um parágrafo" ou "uma imagem"),

o [HTML](#) também conta com vários elementos de nível de bloco usados para definir as áreas de seu site (como "o cabeçalho", "o menu de navegação", "a coluna de conteúdo principal"). Este artigo explora como planejar uma estrutura básica de website, e escrever o HTML para representar essa estrutura.

<b>Pre-requisitos:</b>	Familiaridade básica com HTML, como mostrado em <a href="#">Iniciando com HTML</a> . Formatação de texto HTML, como mostrado em <a href="#">Fundamentos do texto em HTML</a> . O funcionamento de hiperlinks, como visto em <a href="#">Criando hyperlinks</a> .
<b>Objetivo:</b>	Aprenda a estruturar seu documento usando tags semânticas e como elaborar a estrutura de um site simples

# Seções básicas de um documento

As páginas da web podem e serão muito diferentes umas das outras, mas todas tendem a compartilhar componentes padrão semelhantes, a menos que a página exiba um vídeo ou um jogo em tela cheia, seja parte de algum tipo de projeto de arte ou seja mal estruturada:

## cabeçalho (header)

Normalmente, uma grande faixa na parte superior com um grande título e / ou logotipo. É aí que as principais informações comuns sobre um site geralmente ficam de uma página para outra.

## barra de navegação

Links para as principais seções do site; geralmente representado por botões de menu, links ou guias. Como o cabeçalho, esse conteúdo geralmente permanece consistente de uma página para outra - ter uma navegação inconsistente em seu site levará a usuários confusos e frustrados. Muitos web designers consideram a barra de navegação parte do cabeçalho em vez de um componente individual, mas isso não é um requisito; na verdade, alguns também argumentam que ter os dois separados é melhor para acessibilidade, já que os leitores de tela podem ler

melhor os dois recursos se estiverem separados.

## **conteúdo principal**

Uma grande área no centro que contém a maior parte do conteúdo exclusivo de uma determinada página da Web, por exemplo, o vídeo que você deseja assistir ou a história principal que você está lendo ou o mapa que

deseja visualizar ou as manchetes de notícias, etc.

Esta é a única parte do site que definitivamente irá variar de página para página!

## **barra lateral (sidebar)**

Algumas informações periféricas, links, cotações, anúncios etc. Geralmente, isso é contextual ao conteúdo principal (por exemplo, em uma página de um artigo de notícias, a barra lateral pode conter a biografia do autor ou links para artigos relacionados), mas há também casos em que você encontrará alguns elementos recorrentes como um sistema de navegação secundário.

## **rodapé (footer)**

Uma faixa na parte inferior da página que geralmente contém letras pequenas, avisos de direitos autorais ou informações de contato. É um lugar para colocar informações comuns (como o cabeçalho), mas geralmente essas informações não são críticas ou secundárias ao próprio site. O rodapé também é usado às vezes para fins de SEO, fornecendo links para

acesso rápido a conteúdo popular. Um "site típico" poderia ser colocado assim:

The screenshot shows a website layout with the following structure:

- Header:** Contains the word "Header" in a large, bold, italicized font.
- Navigation Bar:** Includes links for HOME, OUR TEAM, PROJECTS, and CONTACT, along with a search bar and a "Go!" button.
- Main Content Area:** Features an **Article heading** (in bold) followed by a paragraph of placeholder text (Lorem ipsum). Below the heading is a **subsection** (also in bold) with its own text. Further down is another subsection with text.
- Sidebar:** Titled **Related**, it contains a list of five items, each with a blue link:
  - [Oh I do like to be beside the sea side](#)
  - [Oh I do like to be beside the sea](#)
  - [Although in the North of England](#)
  - [It never stops raining](#)
  - [Oh well...](#)
- Footer:** Displays the copyright notice "©Copyright 2050 by nobody. All rights reversed."

## HTML para estruturar conteúdo

O exemplo simples mostrado acima não é bonito, mas é perfeitamente aceitável para ilustrar um exemplo típico de layout de website. Alguns sites têm mais colunas, algumas são bem mais complexas, mas você tem a ideia. Com o CSS certo, você poderia usar praticamente todos os elementos para agrupar as diferentes seções e fazer com que parecesse como você queria, mas como discutido

anteriormente, precisamos respeitar a semântica e **usar o elemento certo para o trabalho certo.**

Isso ocorre porque os visuais não contam toda a história. Usamos cor e tamanho de fonte para chamar a atenção dos usuários para as partes mais úteis do conteúdo, como o menu de navegação e links relacionados, mas sobre pessoas

com deficiência visual, por exemplo, que podem não encontrar conceitos como "rosa" e "grande". fonte "muito útil?

Nota: as pessoas daltônicas representam cerca de 4% da população mundial ou, em outras palavras, aproximadamente 1 em cada 12 homens e 1 em cada 200 mulheres são daltônicas. Cegos e deficientes visuais representam cerca de 4-5% da população mundial (em 2012 havia 285 milhões de pessoas no mundo, enquanto a população total era de cerca de 7 bilhões).

Em seu código HTML, você pode marcar seções de conteúdo com base em sua funcionalidade. Você pode usar elementos que representam as seções de conteúdo descritas acima sem ambigüidade, e tecnologias assistivas, como leitores de tela, podem reconhecer esses elementos e ajudar com tarefas como "localizar a navegação principal". "ou" encontre o conteúdo principal. " Como mencionamos anteriormente no curso, há um número de [consequências de não usar a estrutura de elemento e semântica certas para o trabalho certo.](#)

Para implementar essa marcação semântica, o HTML fornece tags dedicadas que você pode usar para representar essas seções, por exemplo:

- **cabeçalho**: `<header>` .
- **barra de navegação**: `<nav>` .
- **conteúdo principal**: `<main>` , com várias subseções de conteúdo representadas por `<article>` , `<section>` , e elementos `<div>` .
- **rodapé**: `<footer>` .

Aprendizagem ativa: explorando o código para o nosso exemplo

Nosso exemplo visto acima é representado pelo seguinte código (você também pode encontrar o exemplo em nosso repositório GitHub ).

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8">

 <title>My page title</title>
 <link href="https://fonts.googleapis.com/
 <link rel="stylesheet" href="style.css">

 <!-- as três linhas abaixo são uma correç
 <!--[if lt IE 9]>
 <script src="https://cdnjs.cloudflare.c
```

```
  ~~~~~~  
  <![endif]-->  
</head>  
  
<body>  
  <!-- Esse é o cabeçalho (header) principal -->  
  
  <header>  
    <h1>Header</h1>  
  </header>  
  
  <nav>  
    <ul>  
      <li><a href="#">Home</a></li>  
      <li><a href="#">Our team</a></li>  
      <li><a href="#">Projects</a></li>  
      <li><a href="#">Contact</a></li>  
    </ul>  
  
    <!-- Um formulário de pesquisa é uma opção comum -->  
  
    <form>  
      <input type="search" name="q" placeholder="Search..."/>  
      <input type="submit" value="Go!"/>  
    </form>  
  </nav>  
  
  <!-- Esse é o conteúdo principal da nossa página -->  
  <main>  
  
    <!-- Contém um artigo -->  
    <article>  
      <h2>Artigo Titular</h2>  
      <p>Este é o conteúdo do artigo. Pode ser qualquer tipo de texto ou mídia que você desejar incluir. O artigo é a unidade básica de conteúdo em uma página web, fornecendo informações para o leitor. Ele pode conter títulos, parágrafos, listas, imagens e outros elementos de layout. O artigo é tipicamente organizado em seções e subseções, com links para outras páginas ou artigos dentro do próprio artigo. O artigo também pode ter comentários, feedback e outras interações com o leitor. O artigo é uma das principais maneiras de transmitir informações e valor ao usuário final. Ele é fundamental para a estrutura e funcionalidade de muitos tipos de sites, desde blogs até sites de notícias e e-commerce. O artigo é uma das principais maneiras de transmitir informações e valor ao usuário final. Ele é fundamental para a estrutura e funcionalidade de muitos tipos de sites, desde blogs até sites de notícias e e-commerce.  
    </article>  
  </main>
```

```
<h1>Article heading</h1>

<p>Lorem ipsum dolor sit amet, consec

<h2>Subsection</h2>

<p>Donec ut librero sed accu vehicula

<p>Pelientesque auctor nisi id magna

<h3>Another subsection</h3>

<p>Donec viverra mi quis quam pulvina

<p>Vivamus fermentum semper porta. Nu
</article>

<!-- O conteúdo do elemento aside pode
<aside>
    <h2>Related</h2>

    <ul>
        <li><a href="#">Oh I do like to be
        <li><a href="#">Oh I do like to be
        <li><a href="#">Although in the Nor
        <li><a href="#">It never stops rain
        <li><a href="#">Oh well...</a></li>
    </ul>
</aside>

</main>
```

```
<!-- Esse é o rodape principal que vai se  
  
<footer>  
    <p>©Copyright 2050 by nobody. All right  
</footer>  
  
</body>  
</html>
```

Reserve um tempo para examinar o código e entendê-lo - os comentários dentro do código também devem ajudá-lo a entendê-lo. Não estamos pedindo que você faça muito mais neste artigo, porque a chave para entender o layout do documento é escrever uma estrutura HTML sólida e, em seguida, defini-la com CSS. Nós vamos aguardar isso até que você comece a estudar o CSS layout como parte do tópico do estudo de CSS.

## Elementos de layout do HTML em mais detalhes

É bom entender o significado geral de todos os elementos de seção do HTML em detalhes - isso é algo em que você trabalhará gradualmente ao começar a obter mais experiência com o desenvolvimento web. Você pode encontrar muitos detalhes lendo o nosso [Elementos HTML](#). Para agora, estes são as principais definições que você deve tentar entender:

- [<main>](#) é para o conteúdo único dessa página. Use `<main>` apenas uma vez por página e o coloca

mais apenas uma vez por página, e o conteúdo

diretamente dentro do `<body>`. Não é ideal aninhar ele dentro de qualquer outro elemento senão o elemento `<body>`.

- `<article>` inclui um bloco de conteúdo relacionado o qual faz sentido por si só, sem o restante da página (por exemplo, uma postagem singular dum blog).
- `<section>` é similar com `<article>`, mas ele é mais para agrupar uma única parte de página que constitui em um único pedaço de funcionalidade (por exemplo, um minimapa, ou um conjunto de manchetes e resumo). É considerado boa prática começar cada seção com um `título`; observe também que você pode dividir um `<article>`s em diferentes `<section>`s, ou `<section>`s em diferentes `<article>`s, dependendo do contexto.
- `<aside>` é para conteúdo que não está relacionados diretamente com os conteúdos principais, mas que podem providenciar informações complementares a esses (entradas de glossários, biografia do autor, links relacionados, etc.).
- `<header>` representa um grupo de conteúdo introdutório. Se ele for um elemento filho do `<body>`, Ele é um header global da página do site, mas se for um elemento filho de um `<article>` ou `<section>`, é definido como um header específico para essa seção ( tenta não confundir isso com `títulos e cabeçalhos`).
- `<nav>` contém a funcionalidade principal de

navegação da página. Links secundários, etc., não iria na navegação

- <footer> representa um grupo de conteúdo final da página.

## Elementos de layout não-semânticos

Às vezes, você se depara numa situação em que não consegue encontrar um elemento semântico ideal para agrupar alguns itens ou agrupar algum conteúdo. Nesses momentos, convém agrupar um conjunto de elementos para afetá-los todos como uma única entidade com alguns CSS ou JavaScript. Para casos como esses, HTML oferece os elementos <div> e <span>. Você deve usá-los preferencialmente com um atributo class adequado, para fornecer a eles algum tipo de rótulo para que possam ser facilmente referenciados.

<span> é um elemento não-semântico embutido, que você deve usar apenas se não conseguir pensar em um elemento de texto semântico melhor para agrupar seu conteúdo ou se não quiser adicionar um significado específico. Por exemplo:

`<p>O rei voltou bêbado para o quarto às 0 [ ] 0 enquanto ele cambaleando pela porta <span> a`

Nesse caso, a nota do editor deve meramente fornecer orientação extra para o diretor da peça; não é suposto ter um significado semântico extra. Para usuários observador, talvez

o CSS fosse usado para distanciar a nota um pouco do texto principal.

`<div>` é um elemento não semântico no nível de bloco, que você deve usar apenas se não conseguir pensar em um elemento de bloco semântico melhor para usar ou se não quiser adicionar um significado específico. Por exemplo,

imagine um widget de carrinho de compras que você poderia escolher a qualquer momento durante o seu tempo em um site de comércio eletrônico:

```
<div class="carrinho-de-compras">
  <h2>Carrinho de compras</h2>
  <ul>
    <li>
      <p><a href=""><strong>Brincos de prata<
         Pré-requisitos:</b> | Familiaridade com HTML, conforme abordado, por exemplo, em <a href="#">Introdução ao HTML</a> , <a href="#">Fundamentos de texto em HTML</a> e <a href="#">Criação de Hiperlinks</a> . |
| <b>Objetivo:</b>       | Aprender o básico sobre o uso de ferramentas de depuração (debugging) para encontrar problemas em HTML.                                                                                |

## Depurar não é assustador

Ao escrever algum tipo de código, tudo costuma ir bem, até o temido momento quando ocorre um erro — você fez algo errado, então seu código não funciona - talvez não funcione mais nada ou não funcione exatamente como você queria. Por exemplo, a seguir é mostrado um erro relatado ao tentar [compilar](#) um programa simples escrito na linguagem [Rust](#) .

```
To learn more, run the command again with --verbose.  
cm-macbook-pro:first chrismills$ clear  
  
[cm-macbook-pro:first chrismills$ cargo run  
  Compiling hello_world v0.0.1 (file:///Users/chrismills/Dropbox/Work/rust/first)  
[src/main.rs:2:27: 3:2 error: unterminated double quote string  
src/main.rs:2     println!("Hello, world!");  
src/main.rs:3 }  
Could not compile 'hello_world'.  
  
To learn more, run the command again with --verbose.  
cm-macbook-pro:first chrismills$ ]
```

Aqui, a mensagem de erro é relativamente fácil de entender — "string de aspas duplas sem terminação". Se você olhar a listagem, provavelmente verá como `println!("Hello, world!");` pode estar faltando logicamente uma aspa dupla. No entanto, as mensagens de erro podem ficar mais complicadas e menos fáceis de interpretar à medida que os programas se tornam maiores, e até mesmo casos simples podem parecer um pouco intimidadores para alguém que não sabe nada sobre o Rust.

Depurar um código não tem que ser assustador, porém — a chave para se sentir confortável em escrever e depurar qualquer linguagem ou código de programação é a familiaridade com a linguagem e as ferramentas.

## HTML e depuração

HTML não é tão complicado de entender quanto o Rust. O HTML **não é compilado** em um formato diferente antes do navegador analisá-lo e mostrar o resultado (ele é interpretado, não compilado). E a sintaxe do elemento HTML é muito mais fácil de entender do que uma "linguagem de programação real" como Rust, JavaScript, ou Python. A forma como os navegadores analisam o HTML é muito mais **permissiva** do que a forma como as linguagens de programação são executadas, o que é bom e ruim.

### Código permissivo

Então, o que queremos dizer com permissivo? Bem, geralmente quando você faz algo errado no código, existem dois tipos principais de erros que você encontrará:

- **Erros de sintaxe:** São os erros de ortografia no seu código que realmente fazem com que o programa não seja executado, como o erro do Rust mostrado acima. Estes geralmente são fáceis de corrigir, desde que você esteja familiarizado com a sintaxe (forma de escrever) da linguagem e saiba o que significam as mensagens de erro.
- **Erros lógicos:** São erros onde a sintaxe está correta, mas o código não é o que você pretendia, o que significa que o programa é executado incorretamente. Geralmente, eles são mais difíceis de corrigir do que erros de sintaxe, pois não há uma mensagem de erro para direcioná-lo para a origem deste erro.

O próprio HTML não sofre de erros de sintaxe porque os navegadores o analisam permissivamente, o que significa que a página ainda é exibida mesmo se houver erros de sintaxe. Os navegadores têm regras internas para indicar como interpretar a marcação escrita incorretamente, para que você obtenha algo em execução, mesmo que não seja o esperado. Isso, claro, ainda pode ser um problema!

**Nota:** O HTML é analisado permissivamente porque, quando a web foi criada, foi decidido que permitir que as pessoas publicassem seus conteúdos era mais importante do que garantir que a sintaxe estivesse absolutamente correta. A web provavelmente não seria tão popular quanto é hoje, se tivesse sido mais rigorosa desde o início.

## Aprendizado Ativo: Estudando código permissivo

É hora de estudar a natureza permissiva do código HTML.

1. Primeiramente, faça o download do [debug-example demo](#) e o salve localmente. Esse exemplo contém erros propositais para que possamos explorá-los (tal código HTML é dito **badly-formed**, em contraponto ao HTML **well-formed**).
2. Em seguida, abra o arquivo em um navegador. Você verá algo como:
3. Isso claramente não parece bom; vamos dar uma olhada no código fonte para tentar achar os erros (somente o conteúdo de *body* é mostrado):

```
<h1>HTML debugging examples</h1>

<p>What causes errors in HTML?

<ul>
  <li>Unclosed elements: If an element is <strong>not closed properly
      then its effect can spread to areas you didn't intend

  <li>Badly nested elements: Nesting elements properly is also very
      important for code behaving correctly. <strong>strong <em>strong emphasis
      what is this?</em>

  <li>Unclosed attributes: Another common source of HTML problems.
      look at an example: <a href="https://www.mozilla.org/">link to Mozilla</a>
```

```
homepage</a>
</ul>
```

4. Vamos analisar os erros:

- Os elementos parágrafo e item da lista não possuem *tags* de fechamento. Olhando a imagem acima, isso não parece ter afetado muito a renderização do HTML já que é fácil deduzir onde um elemento deveria terminar e outro, começar.
- O primeiro elemento <strong> não possui *tag* de fechamento. Isto é um pouco mais problemático porque não é necessariamente fácil determinar onde um elemento deveria terminar. Assim, todo o resto do texto foi fortemente enfatizado.
- Essa seção foi aninhada incorretamente:  

```
<strong>strong <em>strong emphasised?</strong> what is this?</em>
```

Não é fácil dizer como esse trecho foi interpretado por causa do problema anterior.
- O valor do atributo href não tem as aspas de fechamento. Isso parece ter causado o maior problema — o *link* não foi renderizado.

5. Agora vamos dar uma olhada no HTML que o navegador renderizou, comparando-o com o nosso código fonte. Para fazer isso, usaremos as ferramentas de desenvolvimento oferecidas pelo navegador. Se você não está familiarizado com estas ferramentas, dê uma olhadinha nesse tutorial: [O que são as ferramentas de desenvolvimento do navegador](#).

6. No inspetor DOM, você pode ver como o HTML renderizado fica:

7. Utilizando o inspetor DOM, vamos explorar nosso código detalhadamente para ver como o navegador tentou consertar os erros do código HTML (nós fizemos a análise com o Firefox, mas outros navegadores modernos *devem* apresentar o mesmo resultado):

- As *tags* de fechamento foram colocadas nos parágrafos e itens da lista.
- Não está claro onde o primeiro elemento <strong> deveria terminar, portanto o navegador envolveu cada bloco subsequente em uma *tag strong* própria até o fim do documento!
- O aninhamento incorreto foi corrigido pelo navegador da seguinte maneira:

```
<strong>strong
  <em>strong emphasised?</em>
</strong>
<em> what is this?</em>
```

- O link cujas aspas de fechamento não estavam presentes foi totalmente excluído da renderização. Então o último item ficou assim:

```
<li>
  <strong>Unclosed attributes: Another common source of HT p
    Let's look at an example: </strong>
</li>
```

Então, você pode ver pelo exemplo acima que você realmente quer ter certeza de que o seu HTML foi bem construído! Mas Como? Em um pequeno exemplo como o que foi visto acima, é fácil analisar as linhas e achar os erros, mas se fosse um gigante e complexo documento HTML?

A melhor estratégia é começar rodando a sua página HTML através do [Markup Validation Service](#) — criado e mantido pelo W3C, uma organização que cuida das especificações que define o HTML, CSS, e outras tecnologias WEB. Esta página considera um documento HTML como uma entrada, fazendo a leitura dela e retornando o que há de errado com o seu HTML.

Para especificar o HTML a ser validado, você pode dar um endereço web, fazer o upload de um arquivo HTML, ou diretamente inserir o código HTML.

## Aprendizado Ativo: Validando um documento HTML

Vamos tentar fazer isto com o nosso [sample document](#).

1. Primeiro, carregue o [Markup Validation Service](#) em uma aba no seu navegador, caso já não esteja carregada.
2. Troque para a aba [Validate by Direct Input](#).
3. Copie todo o código do documento de exemplo (não apenas o body) e cole dentro da grande área de texto mostrada no Markup Validation Service.
4. Pressione o botão *Check*.

Você deverá receber uma lista de erros e outras informações.

## Interpretando as mensagens de erros

As mensagens de erros geralmente são úteis, mas algumas vezes elas não ajudam tanto; com um pouco de prática você pode descobrir como interpretar-lás para arrumar o seu código. Vamos dar uma olhada nas mensagens de erros e ver o que elas significam. Você verá que cada mensagem vem com um número para a linha e um para a coluna afim de ajudar você a localizar o erro facilmente.

- "End tag `li` implied, but there were open elements" (2 instances): Estas mensagens indicam que um elemento que está aberto deveria estar fechado. O final da tag está implícito, mas não está realmente lá. A informação de linha/coluna indica para a primeira linha depois de onde a tag de fechamento realmente deveria estar, mas isto é uma pista boa o suficiente para ver o que há de errado.
- "Unclosed element `strong`": Este é muito fácil de entender — um [`<strong>`](#) elemento está aberto, e uma informação de linha/coluna indica diretamente para onde está.

- "End tag `strong` violates nesting rules": Este aponta os elementos incorretamente aninhados, e a informação de linha/coluna aponta onde o erro está.
- "End of file reached when inside an attribute value. Ignoring tag": This one is rather cryptic; it refers to the fact that there is an attribute value not properly formed somewhere, possibly near the end of the file because the end of the file appears inside the attribute value. The fact that the browser doesn't render the link should give us a good clue as to what element is at fault.
- "End of file seen and there were open elements": This is a bit ambiguous, but basically refers to the fact there are open elements that need to be properly closed. The line numbers point to the last few lines of the file, and this error message comes with a line of code that points out an example of an open element:

`|example: <a href="https://www.mozilla.org/>link to Mozilla homepage`

**Note:** An attribute missing a closing quote can result in an open element because the rest of the document is interpreted as the attribute's content.

- "Unclosed element `ul`": This is not very helpful, as the `<ul>` element is closed correctly. This error comes up because the `<a>` element is not closed, due to the missing closing quote mark.

If you can't work out what every error message means, don't worry about it — a good idea is to try fixing a few errors at a time. Then try revalidating your HTML to show what errors are left. Sometimes fixing an earlier error will also get rid of other error messages — several errors can often be caused by a single problem, in a domino effect.

You will know when all your errors are fixed when you see the following banner in your output:

## Summary

So there we have it, an introduction to debugging HTML, which should give you some useful skills to count on when you start to debug CSS, JavaScript, and other types of code later on in your career. This also marks the end of the Introduction to HTML module learning articles — now you can go on to testing yourself with our assessments: the first one is linked below.

## Neste módulo

- [Iniciando com HTML](#)
- [O que está no cabeçalho? Metadados em HTML](#)
- [Fundamentos do texto em HTML](#)
- [Criando hyperlinks](#)
- [Formatação avançada de texto](#)

- [Estrutura de documento e sites](#)
- [Debugging HTML](#)
- [Marking up a letter](#)
- [Structuring a page of content](#)

**Last modified:** 1 de fev. de 2022, [by MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Marcando una Carta

Todos aprendemos a escribir una carta más tarde o más temprano; es también práctico practicar con nuestras habilidades para dar forma a los textos. En esta prueba deberás demostrar tus habilidades para dar forma a textos, incluyendo enlaces, además pondremos a prueba tu familiaridad con algunos contenidos de encabezamiento <head> en HTML.

<b>Prerrequisitos:</b>	Antes de intentar este examen deberías haber trabajado los artículos <a href="#">Getting started with HTML</a> , <a href="#">What's in the head?</a> <a href="#">Metadata in HTML</a> , <a href="#">HTML text fundamentals</a> , <a href="#">Creating hyperlinks</a> , y <a href="#">Advanced text formatting</a> .
<b>Objetivos:</b>	Probar las habilidades básicas y avanzadas de formateo de texto e hyperlinks, y el conocimiento de los encabezamientos en HTML.

# Punto de partida

Para comenzar esta prueba, deberemos copiar [el texto que deberemos trabajar](#), y el [CSS que necesitaremos incluir](#) en nuestro HTML.

Crearemos un archivo nuevo `.html` usando nuestro editor de texto (o alternativamente usaremos otros como [JSBin](#) o [Thimble](#) para hacer nuestra prueba).

## Resumen del proyecto a desarrollar

En este proyecto tu tarea será publicar una carta que debe estar alojada en la intranet de una universidad. La carta es la respuesta de un compañero investigador a un posible estudiante de postgrado en relación a su deseo de trabajar en la universidad.

Semánticas de bloque/estructurales:

- Estructura el documento completo incluyendo los elementos (doctype), [<html>](#), [<head>](#) y [<body>](#).
- Incluye los elementos correspondientes de marcado en la carta tales como párrafos y títulos, a excepción de los siguientes. Hay un título principal (la línea que comienza por "Re:") y tres títulos secundarios.
- Las fechas de comienzo de los semestres, las materias y los bailes exóticos deben ser marcados con los

correspondientes tipos de lista.

- Colocar las dos direcciones dentro de elementos <address> . Cada línea de la dirección debe comenzar en una línea nueva, pero no en un párrafo nuevo.

Semánticas intralínea:

- Los nombres de remitente y destinatario (también "Tel" e "Email") deben ser marcado con importancia (strong).
- Deberás usar los elementos apropiados en las cuatro fechas contenidas en el documento para que puedan ser leidas por los motores de lectura automática.
- La primera dirección y la primera fecha en la carta deben ser asignadas a un atributo de clase llamado "sender-column"; el código CSS lo añadirás posteriormente para que quede bien alineado, como debe ser en un formato de carta clásico.
- Deberás utilizar el elemento apropiado para los cinco acrónimos/abreviaciones contenidos en el texto principal, proporcionándoles las extensiones correspondientes.
- Marca apropiadamente los seis sub/superíndices.
- Los símbolos de los grados, los mayor que y los símbolos de multiplicar deben ser marcados usando las referencias correctas.
- Marca al menos dos palabras en el texto con fuerte importancia/énfasis.

- Hay dos lugares donde deberíamos añadir hyperlinks; añádelos con títulos. Como sitio donde apuntan simplemente usa: <http://example.com>.
- Marca con el elemento apropiado el lema de la universidad y la cita del autor.

El encabezamiento del documento:

- El juego de caracteres del documento deberá ser utf-8 usando una etiqueta meta adecuada.
- El autor de la carta debe estar especificado con la etiqueta meta correspondiente.
- El CSS proporcionado deberá estar incluido dentro de la etiqueta adecuada.

## Pistas y recomendaciones

- Utiliza el [Validador de HTML W3C HTML](#) para validar tu HTML; recibirás puntos de bonificación si se valida.
- No necesitas conocer CSS para hacer este ejercicio; solo debes poner el CSS proporcionado en el elemento HTML adecuado.

## Example

The following screenshot shows an example of what the letter might look like after being marked up.

*Dr. Eleanor Gaye*  
Awesome Science faculty  
University of Awesome  
Bobtown, CA 99999,  
USA  
*Tel:* 123-456-7890  
*Email:* no\_reply@example.com

20 January 2016

*Miss Eileen Dover*  
4321 Cliff Top Edge  
Dover, CT9 XXX  
UK

## Re: Eileen Dover university application

Dear Eileen,

Thank you for your recent application to join us at the University of Awesome's science faculty to study as part of your PhD next year. I will answer your questions one by one, in the following sections.

### Starting dates

We are happy to accomodate you starting your study with us at any time, however it would suit us better if you could start at the beginning of a semester; the start dates for each one are as follows:

- First semester: 9 September 2016
- Second semester: 15 January 2017
- Third semester: 2 May 2017

Please let me know if this is ok, and if so which start date you would prefer.

You can find more information about [important university dates](#) on our website.

### Subjects of study

At the Awesome Science Faculty, we have a pretty open-minded research facility — as long as the subjects fall somewhere in the realm of science and technology. You seem like an intelligent, dedicated researcher, and just the kind of person we'd like to have on our team. Saying that, of the ideas you submitted we were most intrigued by are as follows, in order of priority:

1. Turning H<sub>2</sub>O into wine, and the health benefits of Resveratrol (C<sub>14</sub>H<sub>12</sub>O<sub>3</sub>.)
2. Measuring the effect on performance of funk bassplayers at temperatures exceeding 30°C (86°F), when the audience size exponentially increases (effect of  $3 \times 10^3 > 3 \times 10^4$ .)
3. [HTML](#) and [CSS](#) constructs for representing musical scores.

So please can you provide more information on each of these subjects, including how long you'd expect the

research to take, required staff and other resources, and anything else you think we'd need to know? Thanks.

### Exotic dance moves

Yes, you are right! As part of my post-doctorate work, I *did* study exotic tribal dances. To answer your question, my favourite dances are as follows, with definitions:

#### Polynesian chicken dance

A little known but *very* influential dance dating back as far as 300BC, a whole village would dance around in a circle like chickens, to encourage their livestock or be "fruitful".

#### Icelandic brownian shuffle

Before the Icelanders developed fire as a means of getting warm, they used to practice this dance, which involved huddling close together in a circle on the floor, and shuffling their bodies around in imperceptibly tiny, very rapid movements. One of my fellow students used to say that he thought this dance inspired modern styles such as Twerking.

#### Arctic robot dance

An interesting example of historic misinformation, English explorers in the 1960s believed to have discovered a new dance style characterised by "robotic", stilted movements, being practiced by inhabitants of Northern Alaska and Canada. Later on however it was discovered that they were just moving like this because they were really cold.

For more of my research, see my [exotic dance research page](#).

Yours sincerely,

Dr Eleanor Gaye

University of Awesome motto: "Be excellent to each other." -- *Bill S Preston, Esq*

## Evaluación

Si estás haciendo esta prueba como parte de un curso organizado, deberías entregar tu trabajo al profesor para que lo corrija. Si estás auto-aprendiendo puedes conseguir la guía de corrección fácilmente pidiéndola en el [Hilo del área de aprendizaje](#), o en el canal IRC de [#mdn](#) en [Mozilla IRC](#). Intenta hacerlo primero — no ganarás nada haciendo trampas.

~~LAST MODIFIED: 1 NOV 2022, [MY MDTX COMMUNITIES](#)~~



Portuguese



Portuguese

MDN Web Docs  
moz://a

# Estruturando uma página de conteúdo

Estruturar uma página de conteúdo pronta para o layout usando CSS é uma habilidade muito importante para dominar, portanto, nesta avaliação, você será testado em sua capacidade de pensar em como uma página pode ficar e escolher a semântica estrutural apropriada para criar um layout em cima.

<b>Pré-requisitos:</b>	Antes de tentar esta avaliação, você já deve ter trabalhado no restante do curso, com ênfase particular na <a href="#">estrutura de documentos e sites</a> .
<b>Objetivo:</b>	Para testar o conhecimento de estruturas de páginas da Web e como representar um projeto de layout prospectivo na marcação.



Portuguese → Portuguese



Para começar esta avaliação, você deve ir e pegar o [zip](#) contendo todos os ativos iniciais .

O arquivo zip contém:

- O HTML ao qual você precisa adicionar marcação estrutural.
- CSS para estilizar sua marcação.
- Imagens que são usadas na página.

Crie o exemplo em seu computador local ou, alternativamente, use uma ferramenta online, como [CodePen](#) , [jsFiddle](#) , ou [Falha](#) para trabalhar nas tarefas.

**Observação:** se você ficar preso, peça ajuda - consulte a seção [Avaliação ou ajuda adicional](#) na parte inferior desta página.

## Resumo do projeto

Para este projeto, sua tarefa é pegar o conteúdo da página inicial de um site de observação de pássaros e adicionar elementos estruturais a ele para que possa ter um layout de página aplicado a ele. Ele precisa ter:

- Um cabeçalho abrangendo toda a largura do site contendo o título principal da página, o logo tipo do site



---

navegação aparece abaixo desses dois itens.

- Uma área de conteúdo principal contendo duas colunas — um bloco principal para conter o texto de boas-vindas e uma barra lateral para conter miniaturas de imagens.
- Um rodapé contendo informações de direitos autorais e créditos.

Você precisa adicionar um wrapper adequado para:

- O cabeçalho
- O menu de navegação
- O conteúdo principal
- O texto de boas-vindas
- A barra lateral da imagem
- O rodapé

Você também deveria:

- Aplique o CSS fornecido à página adicionando outro <link> elemento logo abaixo do existente fornecido no início.

## Dicas e sugestões



Portuguese → Portuguese



erros que você poderia ter cometido - para que você possa corrigi-los.

- Você não precisa conhecer nenhum CSS para fazer essa avaliação; você só precisa colocar o CSS fornecido dentro de um elemento HTML.
- O CSS fornecido é projetado para que, quando os elementos estruturais corretos forem adicionados à marcação, eles aparecerão em verde na página renderizada.
- Se você está ficando preso e não consegue imaginar quais elementos colocar onde, desenhe um diagrama de blocos simples do layout da página e escreva nos elementos que você acha que devem envolver cada bloco. Isso é extremamente útil.

## Exemplo

A captura de tela a seguir mostra um exemplo de como a página inicial pode ficar depois de ser marcada.



Portuguese → Portuguese



# BIRDWATCHING



- HOME
- GET STARTED
- PHOTOS
- GEAR
- FORUM

## WELCOME

Welcome to our fake birdwatching site. If this were a real site, it would be the ideal place to come to learn more about birdwatching, whether you are a beginner looking to learn how to get into birding, or an expert wanting to share ideas, tips, and photos with other like-minded people.

So don't waste time! Get what you need, then turn off that computer and get out into the great outdoors!



This fake website example is CC0 – any part of this code may be reused in any way you wish. Original example written by Chris Mills, 2016.

[Dove icon by Lorc.](#)

## Avaliação ou ajuda adicional

Se você gostaria que seu trabalho fosse avaliado, ou está travado e quer pedir ajuda:

1. Coloque seu trabalho em um editor compartilhável online, como [CodePen](#) , [jsFiddle](#) , ou [Falha](#) .



Portuguese → Portuguese



## VIDA

. Sua postagem deve incluir:

- Um título descritivo como "Avaliação desejada para estruturar uma página de conteúdo".
- Detalhes do que você já tentou, e o que você gostaria que fizéssemos, por exemplo, se você está preso e precisa de ajuda, ou quer uma avaliação.
- A link to the example you want assessed or need help with, in an online shareable editor (as mentioned in step 1 above). This is a good practice to get into — it's very hard to help someone with a coding problem if you can't see their code.
- A link to the actual task or assessment page, so we can find the question you want help with.

## In this module

- [Getting started with HTML](#)
- [What's in the head? Metadata in HTML](#)
- [HTML text fundamentals](#)
- [Creating hyperlinks](#)
- [Advanced text formatting](#)
- [Document and website structure](#)



Portuguese → Portuguese



- [Marking up a letter](#)
- **Structuring a page of content**

**Última modificação:** 8 de outubro de 2021, [por contribuidores do MDN](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Multimídia e Incorporação

Nós vimos muito sobre texto até aqui nesse curso, mas a internet seria muito chata se usassemos apenas texto. Vamos começar a ver como fazer a internet criar vida, com conteúdo mais interessante! Esse módulo explora como usar HTML para incluir multimídia em sua pagina web, usando as diferentes formas de inclusão de imagens , e como adicionar video, audio, e até paginas da web inteiras.

## Pré-requisitos

Antes de iniciar esse módulo, você deve ter um conhecimento razoável de HTML, como previamente abrangido em [introdução a HTML](#). Se você não estudou esse módulo (ou algo similar), estude-o primeiro e depois retorne!

**Nota:** Se você está trabalhando em um computador/tablet/outro dispositivo onde você não tem a habilidade de criar seus próprios arquivos, você pode testar (maior parte) dos exemplos do códigos em

pode testar (maior parte) dos exemplos de código em um programa online para codar tais como [JSBin](#) ou [Thimble](#).

## Guias

Esse módulo contém os seguintes artigos, que vão passar por todos os fundamentos para inserir multimídia em páginas da web.

### Imagens em HTML

Existem outros tipos de multimídia para considerar, porém é lógico começar com o modesto elemento [`<img>`](#), usado para inserir uma simples imagem em uma página da web. Nesse artigo vamos aprender a usar esse elemento com mais profundidade, incluindo os básicos, anotando com legendas usando [`<figure>`](#) e como se relaciona com imagens de fundo em CSS.

### Conteúdo em áudio e vídeo

Agora nós iremos aprender como usar os elementos HTML5 [`<video>`](#) e [`<audio>`](#), para inserir video e audio em nossa página; incluindo o básico, fornecendo acesso a diferentes tipos de arquivo para navegadores diferentes, adicionando legenda , e como adicionar alternativas para navegadores mais antigos.

### De `<object>` para `<iframe>` — outras tecnologias incorporadas

A essa altura, nós gostaríamos de avançar alguns passos sobre um conjunto de elementos que permitem você incorporar uma ampla variedade de tipos de conteúdo na suas páginas web: os elementos [<iframe>](#), [<embed>](#) e [<object>](#) ([en-US](#)). [<iframe>](#)s servem para incorporar outras páginas web, enquanto as outras duas permitem você incorporar PDFs, SVG, e até mesmo Flash — uma tecnologia cada vez menos presente, mas que ainda é possível você encontrar de forma quase regular.

## **Adicionando gráficos vetoriais à Web**

Gráficos vetoriais podem ser muito úteis em determinadas situações. Diferente dos formatos comuns, como PNG/JPG, eles não sofrem distorção/pixelização quando o zoom é ampliado — podendo continuar com a mesma qualidade quando alterado em escala. Esse artigo irá introduzir a você o que são os gráficos vetoriais e como incluir o formato [SVG](#) nas páginas web.

## **Imagens responsivas**

Com a atual variedade de tipos de dispositivos capazes de navegar na web - de celulares móveis à computadores pessoais - um conceito essencial para dominar o mundo web moderno é o design responsivo. Este se refere à criação de páginas web que podem automaticamente mudar seus atributos para se adaptar a diferentes resoluções e tamanhos de tela.

Isso será explorado em mais detalhes em um módulo CSS posterior, mas, por enquanto, iremos verificar as ferramentas HTML disponíveis para criar imagens responsivas, incluindo o elemento [`<picture>`](#).

## Testes de Conhecimentos

Os testes de conhecimentos a seguir vão avaliar seu aprendizado nos assuntos abaixo:

### [Página Inicial do Mozilla](#)

Neste teste nós vamos avaliar seus conhecimentos quanto a algumas técnicas discutidas nos artigos desse módulo, devendo você adicionar imagens e vídeos numa divertida página inicial sobre o Mozilla!

## Veja também

### [Adicionando um bitmap no topo de uma imagem](#)

Image maps consiste em um mecanismo que torna diferentes partes de uma imagem em uma forma de acesso para outros lugares ou coisas (pense em um mapa mundi que apresenta informações sobre o país em que você clicou em cima). Essa técnica pode ser útil as vezes.

### [Web literacy basics 2](#)

An excellent Mozilla foundation course that explores and tests some of the skills talked about in the

*Multimedia and embedding* module. Dive deeper into the basics of composing webpages, designing for accessibility, sharing resources, using online media, and working open.

**Last modified:** 1 de fev. de 2022, [by MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

# Imagens no HTML

No início a Web era somente texto, e era tedioso. Felizmente, não demorou muito para que a capacidade de incorporar imagens (e outros tipos de conteúdo mais interessantes) dentro das páginas da web fosse adicionada. Existem outros tipo de mídia para se considerar, mas é lógico começar com o humilde elemento `<img>`, usado para inserir uma simples imagem em uma página web. Neste artigo, analisaremos como dominar seu uso, incluindo o básico, anotando-o com legendas usando o elemento `<figure>`, e detalhando como ele se relaciona com imagens de fundo do CSS.

<b>Pré-requisitos:</b>	Conhecimento básico em informática, <a href="#">Instalando os Programas Básicos</a> , conhecimento básico em <a href="#">lidando com arquivos</a> , familiaridade com fundamentos do HTML (como abordado em <a href="#">Iniciando com HTML</a> .)
	Para aprender a incorporar imagens simples em HTML <a href="#">aperte os com</a>

**Objetivos:**

simples em HTML, anote-as com legendas e como as imagens HTML se relacionam às imagens de plano de fundo CSS.

## Como colocamos uma imagem numa página web?

Para colocar uma única imagem em uma página da web, usamos o elemento `<img>`. Isso é um elemento vazio (quer dizer que não possui conteúdo de texto ou tag de fechamento) que requer no mínimo um atributo para ser útil — `src` (às vezes pronunciado como seu título completo, `source`). O atributo `src` contém um caminho apontando para a imagem que você deseja incorporar na página, que pode ser uma URL relativa ou absoluta, da mesma maneira que os valores de atributo `href` no elemento `<a>`.

**Nota:** Antes de continuar, você deveria ler [Um guia rápido sobre URLs e caminhos](#) para refrescar sua memória sobre URL relativo e absoluto.

Por exemplo, se sua imagem for chamada `dinossauro.jpg`, e está no mesmo diretório de sua página HTML, você poderia inserir a imagem assim:

```
| 
```

Se a imagem estivesse em um subdiretório de `images`, que estivesse dentro do mesmo diretório da página HTML (que o Google recomenda para fins de indexação/[SEO](#)), então você a incorporaria da seguinte maneira:

```
| 
```

E assim por diante.

**Note:** Os mecanismos de pesquisa também leem os nomes dos arquivos de imagem e os contam para o SEO. Portanto, dê à sua imagem um nome de arquivo descritivo; `dinosaur.jpg` é melhor que `img835.png`.

Você pode incorporar a imagem usando seu URL absoluto, por exemplo:

```
| 
```

Mas isso é inútil, pois apenas faz o navegador trabalhar mais, pesquisando o endereço IP do servidor DNS novamente, etc. Você quase sempre manterá as imagens do seu site no mesmo servidor que o HTML.

**Aviso:** A maioria das imagens tem direitos autorais. Não

exiba uma imagem em sua página da web, a menos que:

- 1) você é o dono da imagem
- 2) você recebeu permissão explícita e por escrito do proprietário da imagem, ou
- 3) você tem ampla prova de que a imagem é, de fato, de domínio público.

Violações de direitos autorais são ilegais e antiéticas.

Além disso, **nunca** aponte seu atributo `src` para uma imagem hospedada no site de outra pessoa à qual você não tem permissão para vincular. Isso é chamado de "hotlinking". Mais uma vez, roubar a largura de banda de alguém é ilegal. Ele também torna a página mais lenta, deixando você sem controle sobre se a imagem é removida ou substituída por algo embarçoso.

Nosso código acima nos daria o seguinte resultado:

**Nota:** Elementos como `<img>` e `<video>` às vezes são chamados de elementos substituídos. Isso ocorre porque o conteúdo e o tamanho do elemento são definidos por um recurso externo (como uma imagem ou arquivo de vídeo), não pelo conteúdo do próprio elemento.

**Nota:** Você pode encontrar o exemplo final desta seção [running on Github](#) (Veja o [source code](#) também.)

## Texto alternativo

O próximo atributo que veremos é `alt`. Seu valor deve ser uma descrição textual da imagem, para uso em situações em que a imagem não pode ser vista/exibida ou leva muito tempo para renderizar devido a uma conexão lenta à Internet. Por exemplo, nosso código acima pode ser modificado da seguinte maneira:

```
`, para transformar uma imagem em um link, você ainda deve fornecer links acessíveis. Nesses casos, você também pode escrevê-lo no mesmo elemento `<a>`, ou dentro do atributo alt da imagem. O que funcionar melhor no seu caso.
- **Texto.** Você não deve colocar seu texto em imagens. Se o cabeçalho principal precisar de uma sombra projetada, por exemplo, use CSS para isso, em vez de colocar o texto em uma imagem. No entanto, se você *realmente não puder evitar fazer isso*, deve fornecer o texto dentro do atributo alt .

Essencialmente, a chave é oferecer uma experiência utilizável, mesmo quando as imagens não podem ser vistas. Isso garante que todos os usuários não estejam perdendo nenhum conteúdo. Tente desativar as imagens no seu navegador e veja como as coisas ficam. Você logo perceberá como o texto alternativo é útil se a imagem não puder ser vista

**Nota:** Para mais informações, consulte o nosso guia para [Textos alternativos](#).

## Largura e altura

Você pode usar os atributos `width` e `height`, para especificar a largura e altura da sua imagem. Você pode encontrar a largura e a altura da sua imagem de várias maneiras. Por exemplo, no Mac, você pode usar **Cmd + I** para exibir as informações do arquivo de imagem. Voltando ao nosso exemplo, poderíamos fazer isso:

```

```

Isso não resulta em muita diferença para a tela, em circunstâncias normais. Mas se a imagem não estiver sendo exibida, por exemplo, o usuário acabou de navegar para a página e a imagem ainda não foi carregada, você notará que o navegador está deixando um espaço para a imagem aparecer:

É uma coisa boa a fazer, resultando no carregamento da

página mais rápido e sem problemas.

No entanto, você não deve alterar o tamanho das suas imagens usando atributos HTML. Se você definir o tamanho da imagem muito grande, terá imagens granuladas, confusas ou muito pequenas e desperdiçando largura de banda ao fazer o download de uma imagem que não atenda às necessidades do usuário. A imagem também pode ficar distorcida, se você não mantiver a [proporção de tela](#). Você deve usar um editor de imagens para colocar sua imagem no tamanho correto antes de colocá-la em sua página da web.

**Nota:** Se você precisar alterar o tamanho de uma imagem, use [CSS](#) então.

## Títulos de imagem

Como [nos links](#), você também pode adicionar o atributo title nas images, para fornecer mais informações de suporte, se necessário. No nosso exemplo, poderíamos fazer isso:

```

      width="400"
      height="341"
      title="Um T-Rex em exibição no Museu da Ciência"
```

Isso nos dá uma dica de ferramenta, assim como os títulos dos links:

Os títulos das imagens não são essenciais para incluir. Geralmente, é melhor incluir essas informações de suporte no texto principal do artigo, em vez de anexá-las à imagem. No entanto, eles são úteis em algumas circunstâncias; por exemplo, em uma galeria de imagens quando você não tem espaço para legendas.

## Aprendizado ativo: incorporando uma imagem

Agora é sua vez de jogar! Esta seção de aprendizado ativo o ajudará a executar com um simples exercício de incorporação. Você é fornecido com um básico `<img>` tag; gostaríamos que você incorporasse a imagem localizada no seguinte URL:

[https://raw.githubusercontent.com/mdn/learning-area/master/html/multimedia-and-embedding/images-in-html/dinosaur\\_small.jpg](https://raw.githubusercontent.com/mdn/learning-area/master/html/multimedia-and-embedding/images-in-html/dinosaur_small.jpg)

Anteriormente, dissemos para nunca vincular as imagens a outros servidores, mas isso é apenas para fins de aprendizado; portanto, deixaremos você de fora dessa vez.

Também gostaríamos que você:

- Adicione algum texto alternativo e verifique se ele funciona incorretamente com o URL da imagem.
- Defina a imagem correta `width` e `height` (dica; isto é 200px largo e 171px altura), experimente outros valores para ver qual é o efeito.
- Defina um `title` na imagem.

Se você cometer um erro, sempre poderá redefini-lo usando o botão *Reset*. Se você realmente ficar preso, pressione o botão *Show solution* para ver a resposta:

## Anotar imagens com figuras e legendas de figuras

Por falar em legendas, existem várias maneiras de adicionar uma legenda para acompanhar sua imagem. Por exemplo, não haveria nada para impedi-lo de fazer isso:

```
<div class="figure">
  

  <p>A T-Rex on display in the Manchester Uni
</div>
```

Está tudo bem. Isso contém o conteúdo que você precisa e é bem estiloso usando CSS. Mas há um problema aqui: não há nada que vincule semanticamente a imagem à sua legenda, o que pode causar problemas para os leitores de tela. Por exemplo, quando você tem 50 imagens e legendas, qual legenda combina com qual imagem?

Uma solução melhor, é usar os elementos do HTML5

<figure> e <figcaption>. Eles são criados exatamente para esse propósito: fornecer um contêiner semântico para figuras e vincular claramente a figura à legenda. Nossa exemplo acima, pode ser reescrito assim:

```
<figure>
  

  <figcaption>A T-Rex on display in the Manche
</figure>
```

O elemento <figcaption> informa aos navegadores e à tecnologia de assistência que a legenda descreve o outro conteúdo do elemento <figure>.

**Nota:** Do ponto de vista da acessibilidade, legendas e alt texto têm papéis distintos. As legendas beneficiam até as pessoas que podem ver a imagem, enquanto alt texto fornece a mesma funcionalidade que uma imagem ausente. Portanto, legendas e alt texto não deve apenas dizer a mesma coisa, porque ambos aparecem quando a imagem desaparece. Tente desativar as imagens no seu navegador e veja como fica.

Uma figura não precisa ser uma imagem. É uma unidade de

conteúdo independente que:

- Expressa seu significado de maneira compacta e fácil de entender.
- Pode ir em vários lugares no fluxo linear da página.
- Fornece informações essenciais de suporte ao texto principal.

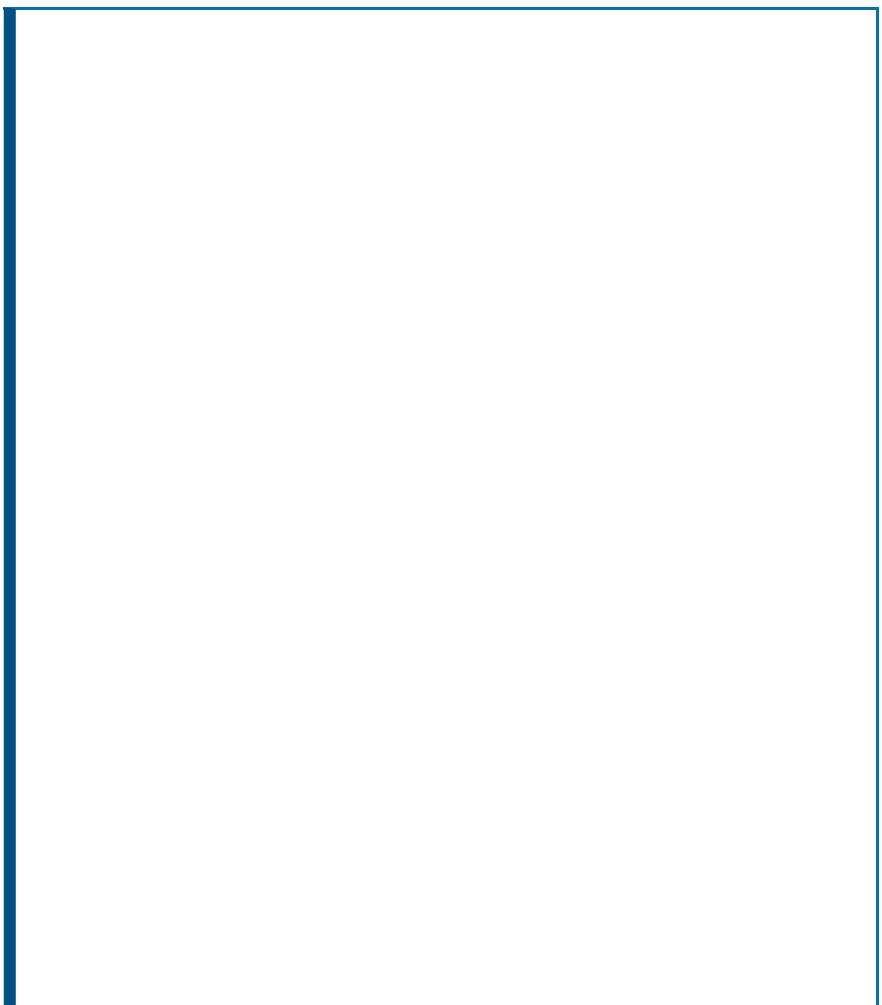
Uma figura pode ser várias imagens, um trecho de código, áudio, vídeo, equações, uma tabela ou outra coisa.

## Aprendizado ativo: criando uma figura

Nesta seção de aprendizado ativo, gostaríamos que você pegasse o código finalizado da seção de aprendizado ativo anterior e o transformasse em uma figura:

- Envolve em um elemento <figure>.
- Copie o texto do atributo `title`, remova o atributo `title`, e coloque o texto dentro de um elemento <figcaption> abaixo da imagem.

Se você cometer um erro, sempre poderá redefini-lo usando o botão *Reset*. Se você realmente ficar preso, pressione o botão *Show solution* para ver a resposta:



# Imagens de fundo CSS

Você também pode usar CSS para incorporar imagens em páginas da web (e JavaScript, mas isso é outra história). A propriedade [background-image \(en-US\)](#) do CSS, e o outras propriedades `background-*`, são usados para controlar o posicionamento da imagem de fundo. Por exemplo, para colocar uma imagem de plano de fundo em cada parágrafo de uma página, você pode fazer o seguinte:

```
p {  
  background-image: url("images/dinosaur.");  
}
```

A imagem incorporada resultante é sem dúvida mais fácil de posicionar e controlar do que as imagens HTML. Então, por que se preocupar com imagens HTML? Como sugerido acima, as imagens de plano de fundo CSS são apenas para decoração. Se você quiser adicionar algo bonito à sua página para melhorar o visual, tudo bem. No entanto, essas imagens não têm significado semântico. Eles não podem ter nenhum equivalente de texto, são invisíveis para os leitores de tela, etc. É hora das imagens HTML brilharem!

Resumindo: se uma imagem tiver significado, em termos de seu conteúdo, você deverá usar uma imagem HTML. Se uma imagem é puramente decorativa, você deve usar imagens de plano de fundo CSS.

**Nota:** Você aprenderá muito mais sobre [CSS background images](#) no nosso tópico de [CSS](#).

É tudo por agora. Cobrimos imagens e legendas em detalhes. No próximo artigo, avançaremos, analisando como

usar HTML para incorporar vídeo e áudio em páginas da web.

**Last modified:** 31 de jan. de 2022, [by MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Conteúdo de vídeo e áudio

Agora que estamos à vontade para adicionar imagens simples a uma página da Web, o próximo passo é começar a adicionar players de vídeo e áudio aos seus documentos HTML! Neste artigo vamos olhar para fazer exatamente isso com os elementos [`<video>`](#) e [`<audio>`](#); Então, vamos terminar de olhar como adicionar legendas/subtítulos aos seus vídeos.

<b>Pré-requisitos:</b>	Alfabetização básica em informática, <a href="#"><u>software básico instalado</u></a> , conhecimento básico de <a href="#"><u>trabalhar com arquivos</u></a> , familiaridade com os fundamentos HTML (Como coberto em <a href="#"><u>Introdução ao HTML</u></a> ) e <a href="#"><u>Images in HTML</u></a> .
<b>Objetivo:</b>	Para aprender a incorporar conteúdo de vídeo e áudio em uma página da Web e adicionar legendas/subtítulos ao vídeo.

# Audio e video na web

Os desenvolvedores da Web quiseram usar vídeo e áudio na Web por um longo tempo, desde o início dos anos 2000, quando começamos a ter largura de banda rápida o suficiente para suportar qualquer tipo de vídeo ( Os arquivos de vídeo são muito maiores que o texto ou mesmo imagens.).

Nos primeiros dias, as tecnologias web nativas, como o HTML, não tinham a capacidade de incorporar vídeo e áudio na Web, de modo que as tecnologias proprietárias (ou baseado em plugin) como o [Flash](#) (e depois, [Silverlight](#)) tornaram-se populares para lidar com esse conteúdo. Esse tipo de tecnologia funcionou bem, mas teve vários problemas, incluindo não funcionar bem com recursos HTML/CSS, problemas de segurança e problemas de acessibilidade.

Uma solução nativa resolveria muito disso, se bem feita. Felizmente, alguns anos depois, o [HTML5 \(en-US\)](#) especificação tinha tais recursos adicionados, com o [`<video>`](#) e [`<audio>`](#) elementos, e alguns novos brilhantes [JavaScript APIs](#) por controlá-los. Não veremos o JavaScript aqui - apenas os fundamentos básicos que podem ser alcançados com o HTML.

Não ensinaremos como produzir arquivos de áudio e vídeo - isso requer um conjunto de habilidades completamente diferente. Nós fornecemos a você [amostras de arquivos de áudio e vídeo e exemplos de códigos](#)

para sua própria experimentação, caso você não consiga se apossar.

**Nota:** Antes de começar aqui, você também deve saber que existem algumas **OVPs** (fornecedores de vídeo online) como [YouTube](#), [Dailymotion](#), e [Vimeo](#), e provedores de áudio on-line como [Soundcloud](#). Essas empresas oferecem uma maneira conveniente e fácil de hospedar e consumir vídeos, para que você não precise se preocupar com o enorme consumo de largura de banda. Os OVPs geralmente oferecem código pronto para incorporar vídeo / áudio em suas páginas da web. Se você seguir esse caminho, poderá evitar algumas das dificuldades que discutimos neste artigo. Discutiremos esse tipo de serviço um pouco mais no próximo artigo.

## O elemento <video>

O elemento [<video>](#) permite incorporar um vídeo com muita facilidade. Um exemplo realmente simples é assim:

```
<video src="rabbit320.webm" controls>
  <p>Your browser doesn't support HTML5 v
</video>
```

Os recursos da nota são:

[src](#)

Da mesma maneira que para o elemento [<img>](#) □

Da mesma maneira que para o elemento `<img>`, o atributo `src` contém um caminho para o vídeo que você deseja incorporar. Funciona exatamente da mesma maneira.

### controls

Os usuários devem poder controlar a reprodução de vídeo e áudio (isso é especialmente crítico para

pessoas que possuem [epilepsy](#).) Você deve usar o atributo `controls` para incluir a própria interface de controle do navegador ou criar sua interface usando o apropriado [JavaScript API](#). No mínimo, a interface deve incluir uma maneira de iniciar e parar a mídia e ajustar o volume.

### O parágrafo dentro do `<video>` tags

Isso é chamado de **conteúdo alternativo** - será exibido se o navegador que acessa a página não suportar o elemento `<video>`, permitindo fornecer um substituto para navegadores mais antigos. Isso pode ser o que você quiser; nesse caso, fornecemos um link direto para o arquivo de vídeo, para que o usuário possa acessá-lo de alguma forma, independentemente do navegador que estiver usando.

O vídeo incorporado será mais ou menos assim:

Você pode tentar o [exemplo ao vivo](#) aqui (veja também o [código fonte](#) .)

## Usando vários formatos de origem para melhorar a compatibilidade

Há um problema no exemplo acima, que você já deve ter notado se você tentou acessar o link do vídeo acima em navegadores mais antigos como Internet Explorer ou até mesmo uma versão antiga do Safari. O vídeo não será reproduzido, porque navegadores diferentes suportam diferentes formatos de vídeo (e áudio). Felizmente, existem coisas que você pode fazer para ajudar a evitar que isso seja um problema.

### Conteúdo de um arquivo de mídia

Primeiro, vamos analisar a terminologia rapidamente.

Formatos como MP3, MP4 e WebM são chamados de arquivos recipiente (formatos de contêiner). Eles definem uma estrutura na qual cada faixa de áudio e / ou vídeo que compõe a mídia é armazenada, juntamente com os metadados que descrevem a mídia, quais codecs são usados para codificar seus canais e assim por diante.

As faixas de áudio e vídeo também estão em diferentes formatos, por exemplo:

- Um contêiner WebM geralmente empacota o áudio do Ogg Vorbis com vídeo VP8 / VP9. Isso é suportado principalmente no Firefox e Chrome.
- Um contêiner MP4 geralmente empacota áudio AAC ou MP3 com vídeo H.264. Isso é suportado principalmente

MP3 COM VÍDEO 11.204. ISSO É SUPORTADO PRINCIPALMENTE NO INTERNET EXPLORER E SAFARI.

- O contêiner Ogg mais antigo tende a usar o áudio Ogg Vorbis e o vídeo Ogg Theora. Isso foi suportado principalmente no Firefox e Chrome, mas foi basicamente substituído pelo formato WebM de melhor qualidade.

Existem alguns casos especiais. Por exemplo, para alguns tipos de áudio, os dados de um codec geralmente são armazenados sem um contêiner ou com um contêiner simplificado. Uma dessas instâncias é o codec FLAC, que é armazenado com mais frequência em arquivos FLAC, que são apenas faixas FLAC brutais.

Outra situação é o sempre popular arquivo MP3. Um "arquivo MP3" é na verdade uma faixa de áudio MPEG-1 Audio Layer III (MP3) armazenada em um contêiner MPEG ou MPEG-2. Isso é especialmente interessante, pois embora a maioria dos navegadores não suporte o uso de mídia MPEG nos elementos [<video>](#) e [<audio>](#), eles ainda podem suportar MP3 devido à sua popularidade.

Um reproduutor de áudio tenderá a reproduzir uma faixa de áudio diretamente, por exemplo um arquivo MP3 ou Ogg. Estes não precisam de contêineres.

**Note:** Não é tão simples, como você pode ver no nosso [tabela de compatibilidade de codec de áudio e vídeo](#).

Alem disso, muitos navegadores de piataforma movei podem reproduzir um formato não suportado, entregando-o ao reproduutor de mídia do sistema subjacente. Mas isso servirá por enquanto.

## Suporte a arquivos de mídia em navegadores

Os codecs descritos na seção anterior existem para compactar vídeo e áudio em arquivos gerenciáveis, pois o áudio e o vídeo bruto são extremamente grandes. Cada navegador da web suporta uma variedade de [Codecs \(en-US\)](#), como Vorbis ou H.264, que são usados para converter o áudio e o vídeo compactados em dados binários e vice-versa. Cada codec oferece suas próprias vantagens e desvantagens, e cada contêiner também pode oferecer seus próprios recursos positivos e negativos, afetando suas decisões sobre qual usar.

As coisas se tornam um pouco mais complicadas porque cada navegador não apenas suporta um conjunto diferente de formatos de arquivo de contêiner, como também suporta uma seleção diferente de codecs. Para maximizar a probabilidade de seu site ou aplicativo funcionar no navegador de um usuário, você pode precisar fornecer cada arquivo de mídia usado em vários formatos. Se o seu site e o navegador do usuário não compartilharem um formato de mídia em comum, sua mídia simplesmente não será reproduzida.

Devido à complexidade de garantir que a mídia do aplicativo seja visível em todas as combinações de navegadores, plataformas e dispositivos que você deseja acessar, a escolha da melhor combinação de codecs e contêiner pode ser uma tarefa complicada. Veja [Choosing the right container](#) in [Media container formats \(file types\)](#) para obter ajuda na seleção do formato de arquivo do contêiner mais adequado

às suas necessidades; Da mesma forma, veja [Choosing a video codec](#) in [Web video codec guide](#) e [Choosing an audio codec](#) in [Web audio codec guide](#) para obter ajuda na seleção dos primeiros codecs de mídia a serem usados no seu conteúdo e no seu público-alvo.

Um aspecto adicional a ter em mente: os navegadores móveis podem suportar formatos adicionais não compatíveis com seus equivalentes de desktop, assim como podem não suportar os mesmos formatos da versão para desktop. Além disso, os navegadores de desktop e móveis *podem* ser projetados para descarregar o manuseio da reprodução de mídia (para todas as mídias ou apenas para tipos específicos que não podem ser tratados internamente). Isso significa que o suporte à mídia depende parcialmente do software que o usuário instalou.

**Note:** Você pode estar se perguntando por que essa situação existe. **MP3** (para áudio) e **MP4 / H.264** (para vídeo) são amplamente suportados e de boa qualidade. No entanto, eles também são patenteados - as patentes

americanas cobrem o MP3 até pelo menos 2017 e o H.264 até 2027, o que significa que os navegadores que não possuem a patente precisam pagar grandes quantias para suportar esses formatos. Além disso, muitas pessoas evitam, por princípio, software restrito, a favor de formatos abertos. É por isso que precisamos fornecer vários formatos para diferentes navegadores.

Então, como fazemos isso? Dê uma olhada no seguinte [exemplo atualizado](#) ([tente ao vivo aqui](#), também):

```
<video controls>
  <source src="rabbit320.mp4" type="video/mp4" />
  <source src="rabbit320.webm" type="video/webm" />
  <p>Your browser doesn't support HTML5 video</p>
</video>
```

Aqui nós tiramos o atributo `src` (`source`) do [`<video>`](#) tag, mas incluímos os elementos [`<source>`](#) que apontam para suas próprias fontes. Nesse caso, o navegador passará pelo elemento [`<source>`](#) e reproduza o primeiro que ele possui o codec para suportar. A inclusão de fontes WebM e MP4 deve ser suficiente para reproduzir seu vídeo na maioria das plataformas e navegadores atualmente.

Cada elemento `<source>` também tem um atributo [`type`](#). Isso é opcional, mas é recomendável que você os inclua - eles contêm o [MIME types \(en-US\)](#) dos arquivos de vídeo, e os navegadores podem lê-los e pular imediatamente os vídeos que não entendem. Se não estiverem incluídos, os

navegadores carregarão e tentarão reproduzir cada arquivo até encontrar um que funcione, consumindo ainda mais tempo e recursos.

**Nota:** Consulte o nosso [guia sobre tipos e formatos de mídias](#) (inglês) para obter ajuda na seleção dos melhores contêineres e codecs para suas necessidades, bem como procurar os tipos MIME certos para especificar cada

## Outros recursos de <video>

Há vários outros recursos que você pode incluir em um vídeo HTML5. Dê uma olhada no nosso terceiro exemplo, abaixo:

```
<video controls width="400" height="400"   
    autoplay loop muted  
    poster="poster.png">  
    <source src="rabbit320.mp4" type="video/mp4"  
    <source src="rabbit320.webm" type="video/we  
    <p>Your browser doesn't support HTML5 video  
</video>
```

Isso nos dará uma saída parecida com esta:

Os novos recursos são:

### width and height

Você pode controlar o tamanho do vídeo com esses

atributos ou com [CSS](#). Nos dois casos, os vídeos mantêm sua proporção largura-altura nativa - conhecida como **proporção de tela**. Se a proporção não for mantida pelos tamanhos definidos, o vídeo aumentará para preencher o espaço horizontalmente, e o espaço não preenchido receberá apenas uma cor sólida de fundo por padrão.

### [autoplay](#)

Faz com que o áudio ou o vídeo comece a ser reproduzido imediatamente, enquanto o restante da página está sendo carregado. É aconselhável não usar vídeo (ou áudio) de reprodução automática em seus sites, porque os usuários podem achar isso realmente irritante.

### [loop](#)

Faz com que o vídeo (ou áudio) comece a ser reproduzido novamente sempre que terminar. Isso também pode ser irritante, portanto, use apenas se for realmente necessário.

### [muted](#)

Faz com que a mídia seja reproduzida com o som desativado por padrão.

### [poster](#)

O URL de uma imagem que será exibida antes da reprodução do vídeo. Destina-se a ser usado para uma tela inicial ou tela de publicidade.

## preload

Usado para armazenar arquivos grandes em buffer;  
pode levar um dos três valores:

- "none" não armazena em buffer o arquivo
- "auto" armazena em buffer o arquivo de mídia
- "metadata" armazena em buffer apenas os metadados do arquivo

Você pode encontrar o exemplo acima disponível para [tocar ao vivo no Github](#) (veja também o [código fonte](#) .) Observe que não incluímos o atributo autoplay na versão ao vivo - se o vídeo começar a ser reproduzido assim que a página for carregada, você não poderá ver o pôster!

## O elemento <audio>

O elemento [<audio>](#) funciona exatamente como o elemento [<video>](#), com algumas pequenas diferenças, conforme descrito abaixo. Um exemplo típico pode parecer assim:

```
<audio controls>
  <source src="viper.mp3" type="audio/mp3"
  <source src="viper.ogg" type="audio/ogg">
  <p>Your browser doesn't support HTML5 audio
</audio>
```

Isso produz algo como o seguinte em um navegador:

**Note:** You can [run the audio demo live](#) on Github (also see the [audio player source code](#) .)

Isso ocupa menos espaço do que um reproduutor de vídeo, pois não há componente visual - você só precisa exibir controles para reproduzir o áudio. Outras diferenças do vídeo HTML5 são as seguintes:

- O elemento `<audio>` não suporta os atributos `width`/`height` — novamente, não há componente visual; portanto, não há nada para atribuir uma largura ou altura.
- Também não suporta o atributo `poster` — novamente, não há componente visual

Mais do que isso, `<audio>` suporta todos os mesmos recursos que `<video>` — revise as seções acima para obter mais informações sobre elas.

## Reiniciando a reprodução de mídia

A qualquer momento, você pode redefinir a mídia para o início, incluindo o processo de seleção da melhor fonte de mídia, se mais de uma for especificada usando o elemento `<source>` — chamando o método `load()` (en-

[US](#)) do elemento:

```
const mediaElem = document.getElementById("my  
mediaElem.load();
```

## Detectando adição e remoção de faixas

Você pode monitorar as listas de faixas em um elemento de mídia para detectar quando as faixas são adicionadas ou removidas da mídia do elemento. Por exemplo, você pode assistir ao evento [addtrack](#) ser disparado no objeto associado [AudioTrackList\\_\(en-US\)](#) (recuperado por meio de [HTMLMediaElement.audioTracks\\_\(en-US\)](#)) para ser informado quando as faixas de áudio forem adicionadas à mídia:

```
const mediaElem = document.querySelector("vid  
mediaElem.audioTracks.onaddtrack = function(e  
    audioTrackAdded(event.track);  
}
```

Você encontrará mais informações sobre isso na nossa documentação [TrackEvent\\_\(en-US\)](#).

## Exibindo trilhas de texto em vídeo

Agora discutiremos um conceito um pouco mais avançado que é realmente útil para se conhecer. Muitas pessoas não podem ou não querem ouvir o conteúdo de áudio / vídeo que

encontram na Web, pelo menos em determinados momentos.

Por exemplo:

- Muitas pessoas têm problemas auditivos (mais comumente conhecidos como deficientes auditivos ou surdos), portanto, não conseguem ouvir o áudio.
- Outros podem não conseguir ouvir o áudio porque estão em ambientes barulhentos (como um bar lotado quando um jogo de esportes está sendo exibido) ou podem não querer incomodar os outros se estiverem em um local silencioso (como uma biblioteca).
- As pessoas que não falam o idioma do vídeo podem querer uma transcrição de texto ou mesmo tradução para ajudá-las a entender o conteúdo da mídia.
- Da mesma forma, em ambientes em que a reprodução do áudio seria uma distração ou perturbação (como em uma biblioteca ou quando um parceiro está tentando dormir), ter legendas pode ser muito útil.

Não seria bom poder fornecer a essas pessoas uma transcrição das palavras que estão sendo ditas no áudio / vídeo? Bem, graças ao vídeo HTML5, você pode, com o formato WebVTT e o elemento <track>.

**Nota:** "Transcrever" significa "escrever as palavras faladas como texto". O texto resultante é uma "transcrição".

O WebVTT é um formato para gravar arquivos de texto contendo várias seqüências de texto, juntamente com metadados, como a que horas do vídeo você deseja que cada sequência de texto seja exibida e até informações limitadas sobre estilo / posicionamento. Essas cadeias de texto são chamadas de **pistas** e existem vários tipos de pistas que são usadas para propósitos diferentes. As dicas mais comuns são:

### **subtitles**

Traduções de material estrangeiro, para pessoas que não entendem as palavras ditas no áudio.

### **captions**

Transcrições sincronizadas de diálogos ou descrições de sons significativos, para permitir que as pessoas que não conseguem ouvir o áudio entendam o que está acontecendo.

### **timed descriptions**

Texto que deve ser falado pelo media player para descrever elementos visuais importantes para usuários cegos ou deficientes visuais.

Um arquivo WebVTT típico terá a seguinte aparência:

WEBVTT

```
1  
00:00:22.230 --> 00:00:24.606  
This is the first subtitle.  
  
2  
00:00:30.739 --> 00:00:34.074  
This is the second.  
  
...
```

Para que isso seja exibido juntamente com a reprodução de mídia HTML, você precisa:

1. Salve-o como um arquivo .vtt em um local adequado.
2. Vincule ao arquivo .vtt com o elemento <track>.  
<track> deve ser colocado dentro de <audio> ou <video>, mas depois de todos os elementos <source>. Use o atributo kind para especificar se as pistas são subtitles, captions, ou descriptions. Além disso, use srlang para informar ao navegador em que idioma você escreveu as legendas.

Aqui está um exemplo:

```
<video controls>  
  <source src="example.mp4" type="video/mp4" />  
  <source src="example.webm" type="video/webm" />  
  <track kind="subtitles" srclang="pt-BR" src="example.vtt" />
```

```
<track kind="subtitles" src="subtitles_en  
</video>
```

Isso resultará em um vídeo com legendas exibidas, mais ou menos assim:

Para mais detalhes, leia [Adicionando legendas e legendas ao vídeo HTML5](#). Você pode [encontrar o exemplo](#) que acompanha este artigo no Github, escrito por Ian Devlin (consulte o [código-fonte](#) também.) Este exemplo usa algum JavaScript para permitir que os usuários escolham entre diferentes legendas. Observe que, para ativar as legendas, você precisa pressionar o botão "CC" e selecionar uma opção - inglês, alemão ou espanhol.

**Nota:** As faixas de texto também ajudam você com o [SEO](#), pois os mecanismos de pesquisa prosperam especialmente no texto. As trilhas de texto permitem até que os mecanismos de pesquisa sejam vinculados diretamente a um ponto no meio do vídeo.

## Aprendizado ativo: incorporando seu próprio áudio e vídeo

Para esse aprendizado ativo, gostaríamos (idealmente) de você sair para o mundo e gravar alguns de seus próprios vídeos e áudio - a maioria dos telefones hoje em dia permite

gravar áudio e vídeo com muita facilidade, e desde que você possa transferi-lo para o seu computador, você pode usá-lo. Talvez você precise fazer algumas conversões para obter um WebM e MP4 no caso de vídeo e um MP3 e Ogg no caso de áudio, mas existem programas suficientes disponíveis para permitir isso sem problemas, como o [Miro Video Converter](#) e o [Audacity](#). Gostaríamos que você experimentasse!

Se você não conseguir obter nenhum vídeo ou áudio, pode usar nossos [exemplos de arquivos de áudio e vídeo](#) para realizar este exercício. Você também pode usar nosso código de exemplo para referência.

Gostaríamos que você:

1. Salve seus arquivos de áudio e vídeo em um novo diretório no seu computador.
2. Crie um novo arquivo HTML no mesmo diretório, chamado `index.html`.
3. Adicione elementos `<audio>` e `<video>` à página; faça com que eles exibam os controles padrão do navegador.
4. Forneça os dois elementos `<source>` para que os navegadores encontrem o formato de áudio mais compatível e o carreguem. Isso deve incluir atributos de `type`.
5. Dê ao elemento `<video>` um pôster que será exibido antes que o vídeo comece a ser reproduzido. Divirta-se criando seu próprio gráfico de pôster.

Para um bônus adicional, você pode tentar pesquisar faixas de texto e descobrir como adicionar legendas ao seu vídeo.

## Resumo

E isso é um tudo; esperamos que você tenha se divertido brincando com vídeo e áudio em páginas da web! No próximo artigo, veremos outras maneiras de incorporar conteúdo na Web, usando tecnologias como [`<iframe>`](#) } e [`<object>`](#) ([en-US](#)).

## Ver também

- Os elementos de mídia HTML: [`<audio>`](#) , [`<video>`](#) , [`<source>`](#) , [`<track>`](#) .
- [Tecnologias de mídia da Web](#).
- [Guia para tipos e formatos de mídia na Web](#).
- [Adicionando legendas e legendas ao vídeo HTML5](#).
- [Entrega de áudio e vídeo](#): muitos detalhes sobre como colocar áudio e vídeo em páginas da Web usando HTML e JavaScript.
- [Manipulação de áudio e vídeo](#): muitos detalhes sobre a manipulação de áudio e vídeo usando JavaScript (por exemplo, adicionando filtros).
- Opções automatizadas para [traduzir multimídia](#).

**Last modified:** 1 de fev. de 2022, [by MDN contributors](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Do objeto ao iframe - outras tecnologias de incorporação

Até agora você já deve ter aprendido a incorporar coisas em suas páginas da web, incluindo imagens, vídeo e áudio. Neste ponto, gostaria de ter um pouco de um passo para o lado, olhando para alguns elementos que permitem incorporar uma ampla variedade de tipos de conteúdo em suas páginas: as [<iframe>](#), [<embed>](#) e [<object>](#) elementos. `<iframe>`s são para incorporar outras páginas da Web, e as outras duas permitem incorporar PDFs, SVG e até Flash - uma tecnologia que está saindo, mas que você ainda verá semi-regularmente.

<b>Pré-requisitos:</b>	Conhecimento básico em informática, <a href="#"><u>software básico instalado</u></a> , conhecimento básico sobre o <a href="#"><u>trabalho com arquivos</u></a> , familiaridade com os fundamentos de HTML (conforme abordado em <a href="#"><u>Introdução ao HTML</u></a> ) e os artigos anteriores deste módulo.
<b>Objetivo:</b>	Para saber como itens incorporar em

páginas da web que usam [<object>](#), [<embed>](#) e [<iframe>](#), como filmes em Flash e outras páginas da web.

## Uma breve história de incorporação

Há muito tempo, na Web, era popular o uso de **quadros** para criar sites - pequenas partes de um site armazenadas em páginas HTML individuais. Elas foram incorporadas em um documento mestre chamado **conjunto de quadros**, que permitiu especificar a área na tela que cada quadro preenchia, como dimensionar as colunas e linhas de uma tabela. Eles foram considerados o auge da frescura entre a metade e o final dos anos 90, e havia evidências de que ter uma página da Web dividida em partes menores como essa era melhor para as velocidades de download - especialmente perceptível pelas conexões de rede que eram tão lentas na época. No entanto, eles tinham muitos problemas, que superavam quaisquer positivos à medida que as velocidades da rede ficavam mais rápidas, para que você não as veja mais sendo usadas.

Um pouco mais tarde (final dos anos 90, início dos anos 2000), as tecnologias de plug-in tornaram-se muito populares, como [Java Applets](#) e [Flash](#) - isso permitiu que os desenvolvedores da Web incorporassem conteúdo rico em páginas da Web, como vídeos e animações, que não estavam

disponíveis apenas no HTML. A incorporação dessas tecnologias foi alcançada através de elementos como `<object>`, e os menos utilizados `<embed>`, e eles eram muito úteis na época. Desde então, ficaram fora de moda devido a muitos problemas, incluindo acessibilidade, segurança, tamanho do arquivo e muito mais; hoje em dia, a maioria dos dispositivos móveis não suporta mais esses plug-ins, e o suporte para desktop está saindo.

Finalmente, o elemento `<iframe>` apareceu (juntamente com outras formas de incorporar conteúdo, como `<canvas>`, `<video>` etc.). Isso fornece uma maneira de incorporar um documento da Web inteiro dentro de outro, como se fosse um `<img>` ou outro elemento, e é usado regularmente hoje .

Com a lição de história fora do caminho, vamos seguir em frente e ver como usar algumas delas.

## Aprendizado ativo: usos clássicos de incorporação

Neste artigo, vamos pular direto para uma seção de aprendizado ativo, para fornecer imediatamente uma idéia real de para que servem as tecnologias de incorporação. O mundo on-line está muito familiarizado com o [Youtube](#) , mas muitas pessoas não conhecem alguns dos recursos de compartilhamento disponíveis. Vejamos como o YouTube nos permite incorporar um vídeo em qualquer página que gostamos de usar `<iframe>` .

1. Primeiro, vá ao Youtube e encontre o vídeo que você gosta.
2. Abaixo do vídeo, você encontrará um botão *Compartilhar* - selecione para exibir as opções de compartilhamento.
3. Selecione o botão *Incorporar* e você receberá um <iframe> código - copie isso.
4. Insira-o na caixa de *entrada* abaixo e veja qual é o resultado na *saída*.

Para pontos de bônus, você também pode tentar incorporar um [mapa do Google](#) no exemplo:

1. Vá para o Google Maps e encontre um mapa que você gosta.
2. Clique no "Menu Hamburger" (três linhas horizontais) no canto superior esquerdo da interface do usuário.
3. Selecione a opção *Compartilhar ou incorporar mapa*.
4. Selecione a opção Incorporar mapa, que fornecerá algum <iframe> código - copie isso.
5. Insira-o na caixa de *entrada* abaixo e veja qual é o resultado na *saída*.

Se você cometer um erro, sempre poderá redefini-lo usando o botão *Redefinir*. Se você realmente ficar atolado, pressione o botão *Mostrar solução* para ver uma resposta.

## Live output

## Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

[Reset](#)

[Show solution](#)

## Iframes em detalhes

Então, isso foi fácil e divertido, certo? Os elementos [<iframe>](#) foram projetados para permitir que você incorpore outros documentos da Web ao documento atual. Isso é

ótimo para incorporar conteúdo de terceiros em seu site sobre o qual você pode não ter controle direto e não querer implementar sua própria versão - como vídeo de fornecedores de vídeo on-line, sistemas de comentários como  [Disqus](#) , mapas on-line fornecedores de mapas, banners publicitários etc. Os exemplos editáveis ao vivo que você está usando neste curso são implementados usando <iframe>s.

Existem algumas sérias  [preocupações de segurança](#)  a serem consideradas com <iframe>s, como discutiremos abaixo, mas isso não significa que você não deve usá-las em seus sites - apenas requer algum conhecimento e pensamento cuidadoso. Vamos explorar o código um pouco mais detalhadamente. Digamos que você queira incluir o glossário MDN em uma de suas páginas da web - você pode tentar algo como isto:

```
<iframe src="https://developer.mozilla.org/en-US
           width="100%" height="500" frameborder="0
           allowfullscreen sandbox>
<p>
  <a href="https://developer.mozilla.org/en-US
      Fallback link for browsers that don't sup
  </a>
</p>
</iframe>
```

Este exemplo inclui os fundamentos básicos necessários para usar um <iframe>:

[\*\*allowfullscreen\*\*](#)

Se definido, ele `<iframe>` poderá ser colocado no modo de tela cheia usando a [API de tela cheia](#) (um pouco além do escopo deste artigo).

### **frameborder**

Se definido como 1, isso indica ao navegador para desenhar uma borda entre esse quadro e outros quadros, que é o comportamento padrão. 0 remove a borda. Usar isso não é mais recomendado, pois o mesmo efeito pode ser melhor alcançado usando em seu [CSS](#) `.border : none;`

### **src**

Este atributo, como [`<video>`](#) / [`<img>`](#), contém um caminho apontando para o URL do documento a ser incorporado.

### **width e height**

Esse atributos especificam a largura e a altura que você deseja que o iframe seja.

## **Conteúdo alternativo**

Da mesma forma que outros elementos semelhantes [`<video>`](#), você pode incluir conteúdo de fallback entre as `<iframe></iframe>` tags de abertura e fechamento que aparecerão se o navegador não suportar `<iframe>`. Nesse caso, incluímos um link para a página. É improvável que você encontre qualquer navegador que não suporte `<iframe>`s atualmente.

### **sandbox**

Esse atributo, que funciona em navegadores um pouco mais modernos que o restante dos `<iframe>` recursos (por exemplo, IE 10 e superior), requer configurações de segurança mais elevadas; falaremos mais sobre isso na próxima seção.

**Nota:** Para melhorar a velocidade, é uma boa ideia definir o `src` atributo do `iframe` com JavaScript após o carregamento do conteúdo principal. Isso torna sua página utilizável mais cedo e diminui o tempo de carregamento da página oficial (uma importante métrica de SEO ).

## Preocupações com segurança

Acima, mencionamos preocupações de segurança - vamos abordar isso com mais detalhes agora. Não esperamos que você entenda todo esse conteúdo perfeitamente da primeira vez; queremos apenas que você fique ciente dessa preocupação e forneça uma referência para retornar à medida que for mais experiente e começar a considerar o uso de `<iframe>`s em seus experimentos e trabalhos. Além disso, não há necessidade de ter medo e não usar `<iframe>`s - você só precisa ter cuidado. Leia...

Fabricantes de navegadores e desenvolvedores da Web descobriram da maneira mais difícil que iframes são um alvo comum (termo oficial: **vítor de ataque** ) para pessoas más na Web (geralmente chamadas de **hackers** ou, mais precisamente, **crackers** ) atacarem se estiverem tentando modificar maliciosamente seu página da web ou induzir as

pessoas a fazer algo que não desejam, como revelar informações confidenciais como nomes de usuário e senhas. Por esse motivo, engenheiros de especificações e desenvolvedores de navegadores desenvolveram vários mecanismos de segurança para torná- `<iframe>` los mais seguros, e também existem práticas recomendadas a serem consideradas - abordaremos alguns deles abaixo.

Clickjacking é um tipo comum de ataque iframe, no qual hackers incorporam um iframe invisível ao documento (ou incorporam o documento ao próprio site malicioso) e o usam para capturar as interações dos usuários. Essa é uma maneira comum de enganar os usuários ou roubar dados confidenciais.

Um primeiro exemplo rápido - tente carregar o exemplo anterior que mostramos acima em seu navegador - você pode encontrá-lo ao vivo no Github (consulte o código-fonte também.) Na verdade, você não verá nada exibido na página e se olhar para o *console* nas ferramentas de desenvolvedor do navegador, você verá uma mensagem informando o motivo. No Firefox, você será informado sobre o *Load negado pelo X-Frame-Options*:

[https://developer.mozilla.org/en-US/docs/Glossary/não permite o enquadramento](https://developer.mozilla.org/en-US/docs/Glossary/não%20permite%20o%20enquadramento). Isso ocorre porque os desenvolvedores que criaram o MDN incluíram uma configuração no servidor que serve as páginas do site para impedir que elas sejam incorporadas dentro de `<iframe>`s (consulte Configurar diretivas CSP, abaixo.) Isso faz sentido - uma página MDN inteira não faz sentido para ser incorporada em outras páginas, a

menos que você queira fazer algo como incorporá-las ao seu site e reivindicá-las como suas - ou tentar roubar dados via clickjacking , que são coisas muito ruins para se fazer. Além disso, se todos começassem a fazer isso, toda a largura de banda adicional começaria a custar muito dinheiro à Mozilla.

### Incorporar somente quando necessário

Às vezes, faz sentido incorporar conteúdo de terceiros - como vídeos e mapas do youtube -, mas você pode economizar muitas dores de cabeça se incorporar apenas conteúdo de terceiros quando completamente necessário. Uma boa regra geral para a segurança da Web é "*Você nunca pode ser muito cauteloso. Se você fez isso, verifique-o de qualquer maneira. Se alguém o fez, assuma que é perigoso até prova em contrário*".

Além da segurança, você também deve estar ciente dos problemas de propriedade intelectual. A maioria dos conteúdos tem direitos autorais, offline e online, mesmo o conteúdo que você não pode esperar (por exemplo, a maioria das imagens no [Wikimedia Commons](#) ). Nunca exiba conteúdo em sua página da Web, a menos que você seja o proprietário ou os proprietários tenham lhe dado uma permissão inequívoca por escrito. As penalidades por violação de direitos autorais são severas. Novamente, você nunca pode ser muito cauteloso.

Se o conteúdo for licenciado, você deverá obedecer aos termos da licença. Por exemplo, o conteúdo no MDN é [licenciado sob CC-BY-SA](#) . Isso significa que você deve [creditar-nos adequadamente](#) quando citar nosso conteúdo, mesmo que faça alterações substanciais.

## Use HTTPS

HTTPS é a versão criptografada do HTTP. Você deve utilizar HTTPS em seus websites sempre que possível:

1. HTTPS reduz a chance de que conteúdo remoto tenha sido adulterado no caminho.
2. HTTPS previne que conteúdo que tenha incorporado ao site acesse conteúdo em seu documento de origem, e vice versa.

Utilizar HTTPS requer um certificado de segurança, que pode ser bem caro (apesar que o Let's Encrypt deixa as coisas mais fáceis) — se você não puder obter um certificado, você deve fornecer seus documentos com HTTP. Contudo, por conta do segundo benefício do HTTPS descrito acima, *não importa o custo, você nunca deve incorporar conteúdo de terceiros em HTTP.* (No caso do melhor cenário, o navegador de seu usuário irá dizer um aviso assustador.) Todas as empresas com boa reputação irão fornecer conteúdo para ser incorporado por meio <iframe> irá fazê-lo disponível através de HTTPS — veja as URLs dentro do <iframe> no atributo src, quando você está incorporando conteúdo do Google Maps ou Youtube, por exemplo.

**Nota:** Páginas do Github permitem que conteúdo seja fornecido via HTTPS por padrão, então é útil para hospedar conteúdo. Se você está utilizando uma hospedagem diferente e não tem certeza do mesmo, pergunte sobre com o seu provedor de hospedagem.

## Sempre utilize o atributo `sandbox`

Você deseja que os atacantes tenham a menor quantidade possível de poder para causar danos ao seu website, portanto, você deve dar ao conteúdo incorporado *apenas a permissão para fazer o seu trabalho*. É claro, isto se aplica ao seu próprio conteúdo, também. Um container para código onde ele possa ser utilizado apropriadamente — ou para testes — mas não pode causar nenhum dano ao resto do código base (tanto acidental quanto maliciosa) é chamado [sandbox](#).

Conteúdo fora de uma sandbox pode fazer muito mais que o esperado (executar JavaScript, submeter forms, criar novas janelas no navegador, etc.) Por padrão, você deve impor todas as restrições disponíveis utilizando o atributo `sandbox` sem parâmetros, como mostrado em nosso exemplo anterior.

Se absolutamente necessário, você pode adicionar permissões uma a uma (dentro do valor do atributo `sandbox=" "`) — veja em [sandbox](#) as referências de entrada para todas as opções disponíveis. Uma nota importante é que você *nunca* deve adicionar ambos `allow-scripts` e `allow-same-origin` no atributo de `sandbox` — neste caso, conteúdo incorporado pode burlar a política de segurança de mesmo destino que impede sites de executarem scripts, e utilizar JavaScript para desativar o sandboxing completamente.

**Nota:** Sandboxing não fornece nenhuma proteção se atacantes puderem enganar os usuários para que visitem conteúdo malicioso diretamente (fora de um `iframe`). Se existir qualquer chance que certo conteúdo possa ser

malicioso (exemplo, conteúdo gerado por usuários), por favor forneça-o em um domain diferente de seu site principal.

## Configure directivas CSP

CSP ([en-US](#)) significa política de segurança de conteúdo e fornece um conjunto de cabeçalhos HTTP (metadados enviados junto com suas páginas da web quando são veiculados de um servidor da web) projetados para melhorar a segurança do seu documento HTML. Quando se trata de proteger `<iframe>`s, você pode configurar seu servidor para enviar um cabeçalho `X-Frame-Options` apropriado.

stands for content security policy and provides a set of HTTP Headers (metadata sent along with your web pages when they are served from a web server) designed to improve the security of your HTML document. When it comes to securing `<iframe>`s, you can configure your server to send an appropriate `X-Frame-Options` header. Isso pode impedir que outros sites incorporem seu conteúdo em suas páginas da Web (o que habilitaria o clickjacking e vários outros ataques), exatamente o que os desenvolvedores do MDN fizeram, como vimos anteriormente.

**Nota:** Você pode ler a publicação de Frederik Braun `X-Frame-Options Security Header` para obter mais informações sobre este tópico. Obviamente, está fora do escopo uma explicação completa neste artigo.

# The `<embed>` and `<object>` elements

The [`<embed>`](#) and [`<object>` \(en-US\)](#) elements serve a different function to [`<iframe>`](#) — these elements are general purpose embedding tools for embedding multiple types of external content, which include plugin technologies like Java Applets and Flash, PDF (which can be shown in a browser with a PDF plugin), and even content like videos, SVG and images!

**Note:** A **plugin**, in this context, refers to software that provides access to content the browser cannot read natively.

However, you are unlikely to use these elements very much — Applets haven't been used for years, Flash is no longer very popular, due to a number of reasons (see [The case against plugins](#), below), PDFs tend to be better linked to than embedded, and other content such as images and video have much better, easier elements to handle those. Plugins and these embedding methods are really a legacy technology, and we are mainly mentioning them in case you come across them in certain circumstances like intranets, or enterprise projects.

If you find yourself needing to embed plugin content, this is the kind of information you'll need, at a minimum:

	<a href="#"><code>&lt;embed&gt;</code></a>	<a href="#"><code>&lt;object&gt;</code> (en-US)</a>
<a href="#"><u>URL</u></a> of the embedded content	<a href="#"><u>src</u></a>	<a href="#"><u>data</u> (en-US)</a>

	<a href="#"><code>&lt;embed&gt;</code></a>	<a href="#"><code>&lt;object&gt;</code> (en-US)</a>
accurate <a href="#">media type</a> (en-US) of the embedded content	<a href="#"><code>type</code></a>	<a href="#"><code>type</code> (en-US)</a>
height and width (in CSS pixels) of the box controlled by the plugin	<a href="#"><code>height</code></a> <a href="#"><code>width</code></a>	<a href="#"><code>height</code> (en-US)</a> <a href="#"><code>width</code> (en-US)</a>
names and values, to feed the plugin as parameters	ad hoc attributes with those names and values	single-tag <a href="#"><code>&lt;param&gt;</code> (en-US)</a> elements, contained within <code>&lt;object&gt;</code>
independent HTML content as fallback for an unavailable resource	not supported ( <code>&lt;noembed&gt;</code> is obsolete)	contained within <code>&lt;object&gt;</code> , after <code>&lt;param&gt;</code> elements

**Note:** `<object>` requires a `data` attribute, a `type` attribute, or both. If you use both, you may also use the [`typemustmatch` \(en-US\)](#) attribute (only implemented in Firefox, as of this writing). `typemustmatch` keeps the embedded file from running unless the `type` attribute provides the correct media type. `typemustmatch` can therefore confer significant security benefits when you're embedding content from a different [origin](#) (it can keep attackers from running arbitrary scripts through the plugin).

Here's an example that uses the `<embed>` element to embed a Flash movie (see this [live on Github](#) , and [check the source code](#) too):

```
<embed src="whoosh.swf" quality="medium"
       bgcolor="#ffffff" width="550" height="300"
       name="whoosh" align="middle" allowScriptAccess="true"
       allowFullScreen="false" type="application/x-shockwave-flash"
       pluginspage="http://www.macromedia.com/go/getflashplayer/">
```

Pretty horrible, isn't it? The HTML generated by the Adobe Flash tool tended to be even worse, using an `<object>` element with an `<embed>` element embedded in it, to cover all bases (check out an example.) Flash was even used successfully as fallback content for HTML5 video, for a time, but this is increasingly being seen as not necessary.

Now let's look at an `<object>` example that embeds a PDF into a page (see the [live example](#) and the [source code](#)):

```
<object data="mypdf.pdf" type="application/pdf"
       width="800" height="1200" typemustmatch="true">
<p>You don't have a PDF plugin, but you can
   <a href="mypdf.pdf">download the PDF file.
</a>
</p>
</object>
```

PDFs were a necessary stepping stone between paper and digital, but they pose many [accessibility challenges](#) and can be

hard to read on small screens. They do still tend to be popular in some circles, but it is much better to link to them so they can be downloaded or read on a separate page, rather than embedding them in a webpage.

## The case against plugins

Once upon a time, plugins were indispensable on the Web. Remember the days when you had to install Adobe Flash Player just to watch a movie online? And then you constantly got annoying alerts about updating Flash Player and your Java Runtime Environment. Web technologies have since grown much more robust, and those days are over. For virtually all applications, it's time to stop delivering content that depends on plugins and start taking advantage of Web technologies instead.

- **Broaden your reach to everyone.** Everyone has a browser, but plugins are increasingly rare, especially among mobile users. Since the Web is easily used without any plugins, people would rather just go to your competitors' websites than install a plugin.
- **Give yourself a break from the extra accessibility headaches that come with Flash and other plugins.**
- **Stay clear of additional security hazards.** Adobe Flash is notoriously insecure, even after countless patches. In 2015, Alex Stamos, then-Chief Security Officer at Facebook, requested that Adobe discontinue Flash.

**Note:** Due to its inherent issues and the lack of support for Flash, Adobe announced that they would stop supporting it at the end of 2020. As of January 2020, most browsers block Flash content by default, and by December 31st of 2020, all browsers will have completely removed all Flash functionality. Any existing Flash content will be inaccessible after that date.

So what should you do? If you need interactivity, HTML and [JavaScript](#) can readily get the job done for you with no need for Java applets or outdated ActiveX/BHO technology. Instead of relying on Adobe Flash, you should use [HTML5 video](#) for your media needs, [SVG](#) for vector graphics, and [Canvas](#) for complex images and animations.

[Peter Elst was already writing some years ago](#) that Adobe Flash is rarely the right tool for the job. As for ActiveX, even Microsoft's [Edge \(en-US\)](#) browser no longer supports it.

## Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see [Test your skills: Multimedia and embedding](#).

## Summary

The topic of embedding other content in web documents can quickly become very complex, so in this article, we've tried to introduce it in a simple, familiar way that will immediately seem

relevant, while still hinting at some of the more advanced features of the involved technologies. To start with, you are unlikely to use embedding for much beyond including third-party content like maps and videos on your pages. As you become more experienced, however, you are likely to start finding more uses for them.

There are many other technologies that involve embedding external content besides the ones we discussed here. We saw some in earlier articles, such as `<video>`, `<audio>`, and `<img>`, but there are others to discover, such as `<canvas>` for JavaScript-generated 2D and 3D graphics, and `<svg>` for embedding vector graphics. We'll look at [SVG](#) in the next article of the module.

## In this module

- [Images in HTML](#)
- [Video and audio content](#)
- [From `<object>` to `<iframe>` — other embedding technologies](#)
- [Adding vector graphics to the Web](#)
- [Responsive images](#)
- [Mozilla splash page](#)





This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

# Adicionando vetor gráfico na web

Vector graphics are very useful in many circumstances — they have small file sizes and are highly scalable, so they don't pixelate when zoomed in or blown up to a large size. In this article we'll show you how to include one in your webpage.

<b>Prerequisites:</b>	You should know the <a href="#">basics of HTML</a> and how to <a href="#">insert an image into your document</a> .
<b>Objective:</b>	Learn how to embed an SVG (vector) image into a webpage.

**Note:** This article doesn't intend to teach you SVG; just what it is, and how to add it to web pages.

# O que são vetores gráficos?

Na web, você pode trabalhar com dois tipos de imagem — **raster images**, and **vector images**:

- **Imagens Raster** são definidos usando uma grade de pixels — a raster image file contains information showing exactly where each pixel is to be placed, and exactly what color it should be. Popular web raster formats include Bitmap ( .bmp ), PNG ( .png ), JPEG ( .jpg ), and GIF ( .gif ).
- **Imagens vetoriais** são definidas usando algoritmos — um arquivo de imagem vetorial contains shape and path definitions that the computer can use to work out what the image should look like when rendered on the screen. The [SVG](#) format allows us to create powerful vector graphics for use on the Web.

To give you an idea of the difference between the two, let's look at an example. You can find this example live on our Github repo as [vector-versus-raster.html](#) — it shows two seemingly identical images side by side, of a red star with a black drop shadow. The difference is that the left one is a PNG, and the right one is an SVG image.

The difference becomes apparent when you zoom in the page — the PNG image becomes pixellated as you zoom in because it contains information on where each pixel should be (and what color). When it is zoomed, each pixel is simply

increased in size to fill multiple pixels on screen, so the image starts to look blocky. The vector image however continues to look nice and crisp, because no matter what size it is, the algorithms are used to work out the shapes in the image, with the values simply being scaled as it gets bigger.



**Note:** The images above are actually all PNGs — with the left-hand star in each case representing a raster image, and the right-hand star representing a vector image. Again, go to the [vector-versus-raster.html](#) demo for a real example!

Moreover, vector image files are much lighter than their raster equivalents, because they only need to hold a handful of algorithms, rather than information on every pixel in the image individually.

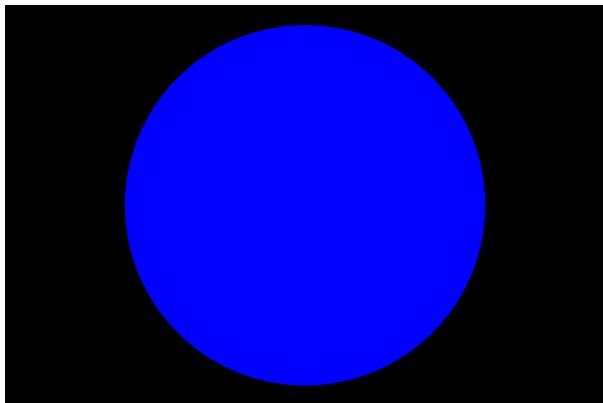
## What is SVG?

SVG is an XML-based language for describing vector images. It's basically markup, like HTML, except that you've got many different elements for defining the shapes you want to appear in your image, and the effects you want to apply to those shapes. SVG is for marking up graphics, not content. At the simplest end of the spectrum, you've got elements for creating simple shapes, like <circle> and <rect>. More advanced SVG features include <feColorMatrix> (en-US) (transform colors using a transformation matrix,) <animate> (animate parts of your vector graphic,) and <mask> (apply a mask over the top of your image.)

As a simple example, the following code creates a circle and a rectangle:

```
<svg version="1.1"
      baseProfile="full"
      width="300" height="200"
      xmlns="http://www.w3.org/2000/svg">
  <rect width="100%" height="100%" fill="black">
  <circle cx="150" cy="100" r="90" fill="blue">
</svg>
```

This creates the following output:



From the example above, you may get the impression that SVG is easy to handcode. Yes, you can handcode simple SVG in a text editor, but for a complex image this quickly starts to get very difficult. For creating SVG images, most people use a vector graphics editor like [Inkscape](#) or [Illustrator](#). These packages allow you to create a variety of illustrations using various graphics tools, and create approximations of photos (for example Inkscape's Trace Bitmap feature.)

SVG has some additional advantages besides those described so far:

- Text in vector images remains accessible (which also benefits your [SEO](#)).
- SVGs lend themselves well to styling/scripting, because each component of the image is an element that can be styled via CSS or scripted via JavaScript.

So why would anyone want to use raster graphics over SVG?

Well, SVG does have some disadvantages:

- SVG can get complicated very quickly, meaning that file sizes can grow; complex SVGs can also take significant processing time in the browser.
- SVG can be harder to create than raster images, depending on what kind of image you are trying to create.
- SVG is not supported in older browsers, so may not be suitable if you need to support older versions of Internet Explorer with your web site (SVG started being supported as of IE9.)

Raster graphics are arguably better for complex precision images such as photos, for the reasons described above.

**Note:** In Inkscape, save your files as Plain SVG to save space. Also, please refer to this [article describing how to prepare SVGs for the Web](#).

# Adding SVG to your pages

In this section we'll go through the different ways in which you can add SVG vector graphics to your web pages.

## The quick way: <img>

To embed an SVG via an `<img>` element, you just need to reference it in the `src` attribute as you'd expect. You will need a `height` or a `width` attribute (or both if your SVG has no inherent aspect ratio). If you have not already done so, please read [Images in HTML](#).

```

```

### Pros

- Quick, familiar image syntax with built-in text equivalent available in the `alt` attribute.
- You can make the image into a hyperlink easily by nesting the `<img>` inside an `<a>` element.
- The SVG file can be cached by the browser, resulting in faster loading times for any page that uses the image loaded in the future.

## Cons

- You cannot manipulate the image with JavaScript.
- If you want to control the SVG content with CSS, you must include inline CSS styles in your SVG code.  
(External stylesheets invoked from the SVG file take no effect.)
- You cannot restyle the image with CSS pseudoclasses (like `:focus`).

## Troubleshooting and cross-browser support

For browsers that don't support SVG (IE 8 and below, Android 2.3 and below), you could reference a PNG or JPG from your `src` attribute and use a `srcset` attribute (which only recent browsers recognize) to reference the SVG. This being the case, only supporting browsers will load the SVG — older browsers will load the PNG instead:

```
| ` method described above, inserting SVGs using CSS background images means that the SVG can't be manipulated with JavaScript, and is also subject to the same CSS limitations.

If your SVGs aren't showing up at all, it might be because your server isn't set up properly. If that's the problem, this [article will point you in the right direction](#).

## How to include SVG code inside your HTML

You can also open up the SVG file in a text editor, copy the SVG code, and paste it into your HTML document — this is sometimes called putting your **SVG inline**, or **inlining SVG**. Make sure your SVG code snippet begins and ends with the `<svg></svg>` tags (don't include anything outside those.) Here's a very simple example of what you might paste into your document:

```
<svg width="300" height="200">
  <rect width="100%" height="100%" fill="red">
</svg>
```

## Pros

- Putting your SVG inline saves an HTTP request, and

therefore can reduce a bit your loading time.

- You can assign `classes` and `ids` to SVG elements and style them with CSS, either within the SVG or wherever you put the CSS style rules for your HTML document. In fact, you can use any [SVG presentation attribute](#) as a CSS property.
- Inlining SVG is the only approach that lets you use CSS interactions (like `:focus`) and CSS animations on your SVG image (even in your regular stylesheet.)
- You can make SVG markup into a hyperlink by wrapping it in an [<a>](#) element.

## Cons

- This method is only suitable if you're using the SVG in only one place. Duplication makes for resource-intensive maintenance.
- Extra SVG code increases the size of your HTML file.
- The browser cannot cache inline SVG as it would cache regular image assets, so pages that include the image will not load faster after the first page containing the image is loaded.
- You may include fallback in a [<foreignObject> \(en-US\)](#) element, but browsers that support SVG still download any fallback images. You need to weigh whether the extra overhead is really worthwhile, just to support obsolescent browsers.

## How to embed an SVG with an <iframe>

You can open SVG images in your browser just like webpages. So embedding an SVG document with an <iframe> is done just like we studied in [From <object> to <iframe> — other embedding technologies](#).

Here's a quick review:

```
<iframe src="triangle.svg" width="500" height="500>
  
</iframe>
```

This is definitely not the best method to choose:

### Cons

- `iframe`s do have a fallback mechanism, as you can see, but browsers only display the fallback if they lack support for `iframe`s altogether.
- Moreover, unless the SVG and your current webpage have the same [origin](#), you cannot use JavaScript on your main webpage to manipulate the SVG.

## Active Learning: Playing with SVG

In this active learning section we'd like you to simply have a go at playing with some SVG for fun. In the *Input* section below you'll see that we've already provided you with some

samples to get you started. You can also go to the [SVG Element Reference](#), find out more details about other toys you can use in SVG, and try those out too. This section is all about practising your research skills, and having some fun.

If you get stuck and can't get your code working, you can always reset it using the *Reset* button.

# Summary

This article has provided you with a quick tour of what vector graphics and SVG are, why they are useful to know about, and how to include SVG inside your webpages. It was never intended to be a full guide to learning SVG, just a pointer so you know what SVG is if you meet it in your travels around

the Web. So don't worry if you don't feel like you are an SVG expert yet. We've included some links below that might help you if you wish to go and find out more about how it works.

In the last article of this module we will explore responsive images in detail, looking at the tools HTML has to allow you to make your images work better across different devices.

## See also

- [SVG tutorial](#) on MDN
- [Quick tips for responsive SVGs](#)
- [Sara Soueidan's tutorial on responsive SVG images](#)
- [Accessibility benefits of SVG](#)
- [How to scale SVGs](#) (it's not as simple as raster graphics!)

## In this module

- [Images in HTML](#)

- [Video and audio content](#)
- [From <object> to <iframe> — other embedding technologies](#)
- [Adding vector graphics to the Web](#)
- [Responsive images](#)
- [Mozilla splash page](#)

**Last modified:** 31 de jan. de 2022, [by MDN contributors](#)

This page was translated from English by the community. Learn more and join the MDN Web Docs community.

## Imagens responsivas

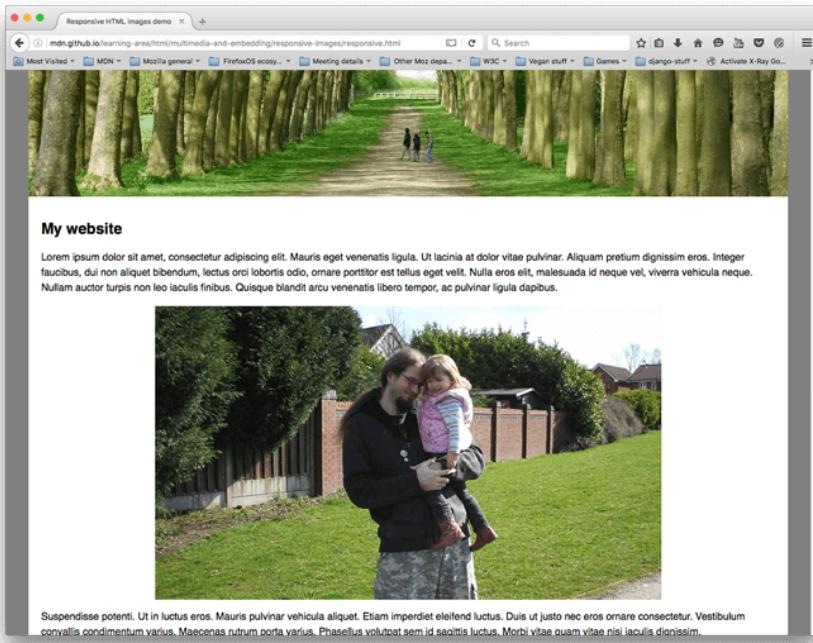
Neste artigo nós iremos aprender sobre o conceito de imagens responsivas —imagens que funcionam em dispositivos com diferentes tamanhos de tela, resoluções e outras funcionalidades— e entrar em contato com quais ferramentas o HTML oferece para ajudar a implementá-las. Imagens responsivas são apenas uma parte do web design responsável, um futuro [tópico de CSS](#) para você aprender.

<b>Pré-requisitos:</b>	Você deve ter visto a <a href="#">introdução ao HTML</a> e como <a href="#">adicionar imagens estáticas numa página web</a> .
<b>Objetivo:</b>	Aprender como usar funcionalidades como <a href="#"><code>srcset</code></a> e o elemento <a href="#"><code>&lt;picture&gt;</code></a> para implementar soluções de imagens responsivas em websites.

## Por que imagens responsivas?

Então qual o problema nós estamos tentando resolver com imagens responsivas? Vamos examinar um cenário típico. Um típico website

provavelmente tem uma imagem de cabeçalho, para ter uma boa aparência para os visitantes, e talvez mais algumas imagens no conteúdo abaixo. Você provavelmente quer fazer uma imagem do cabeçalho em toda a largura do cabeçalho, e o as imagens no conteúdo caiba dentro de alguma coluna. Vamos dar uma olhada em um exemplo simples disso:



Isto funciona bem em um dispositivo de tela grande, como um laptop ou desktop (você pode [ver o exemplo ao vivo](#) e encontrar o [código-fonte](#) no Github.) Nós não vamos discutir muito o CSS, exceto para dizer que:

- O conteúdo do <body> foi colocado para o máximo de 1200 pixels de largura (width) —em viewports acima, o body continua a 1200 pixels e centrado no espaço disponível. Em viewports abaixo, o body vai usar 100% da largura disponível.

- A imagem de cabeçalho foi colocada para estar sempre no centro, não importando a largura do título. Então, se a página está sendo vista em uma tela mais estreita, o detalhe importante no centro da imagem (as pessoas) continuam sendo visto, e o excesso é perdido nos lados. E tem 200 pixels de altura.
- As imagens do conteúdo foram configuradas para caso o elemento body se torne menor que as imagens, então elas começam a diminuir. Assim sempre estarão dentro do body, mesmo que ultrapassando ele.

Isto está bom, mas o problema vem quando você começa a ver a página em uma tela estreita - o cabeçalho parece bom, mas está começando a pegar um tamanho grande para um dispositivo móvel; A primeira imagem do conteúdo por outro lado parece terrível - neste tamanho você mal consegue ver as pessoas nela.

Not responsive demo

mdn.github.io/lea

Most Visited MDN Mozilla general

## My website

Lore ipsum dolor sit amet, consectetur adipiscing elit. Mauris eget venenatis ligula. Ut lacinia at dolor vitae pulvinar. Aliquam pretium dignissim eros. Integer faucibus, dui non aliquet bibendum, lectus orci lobortis odio, ornare porttitor est tellus eget velit. Nulla eros elit, malesuada id neque vel, viverra vehicula neque. Nullam auctor turpis non leo iaculis finibus. Quisque blandit arcu venenatis libero tempor, ac pulvinar ligula dapibus.



Suspendisse potenti. Ut in luctus eros. Mauris pulvinar vehicula aliquet. Etiam imperdiet eleifend luctus. Duis ut justo nec eros ornare consectetur.

Seria muito melhor mostrar uma versão cortada da imagem que contenha os detalhes importantes quando a página é vista em uma tela estreita, e talvez algo entre as duas para uma tela de largura média como um tablet - isto é comumente conhecido como o **problema de direção artística**.

Ainda, não é preciso embutir estas imagens grandes em páginas se será visto em pequenas telas de celulares; isto é chamado de problema de mudança de resolução - uma imagem rasterizada é um número de pixels de largura e um número de pixels de altura; como nós vimos quando olhamos para [vetores gráficos](#), uma imagem rasterizada começa a parecer granulada e horrível se é mostrada maior que seu tamanho original (enquanto que um vetor não). E se isto é mostrado muito menor que seu tamanho original, é um desperdício de largura de banda - usuários mobile especialmente não querem ter que gastar seu pacote de dados baixando uma imagem grande feita para desktop, quando uma imagem pequena poderia ser feita para seu dispositivo. Uma situação ideal seria ter múltiplas resoluções disponíveis e apresentar tamanhos apropriados dependendo dos diferentes dispositivos que acessam a página.

Para tornar as coisas mais complicadas, alguns dispositivos tem uma alta resolução, que demanda imagens maiores do que as que você espera para ficar bom. Isto é essencialmente o mesmo problema, mas em um contexto diferente.

Você pode pensar que imagens vetorizadas resolveria estes problemas, e elas resolvem em certo grau - elas têm um tamanho pequeno e se adaptam bem, e você deveria usá-las sempre que possível. Mas elas não são possíveis para todos os tipos de imagem - enquanto elas são ótimas para gráficos simples, padrões, elementos de interface, etc., começa a ficar complexo criar uma imagem baseada em vetor com o tipo de detalhe que você

encontraria em uma foto, por exemplo. Formatos de imagens rasterizadas, como JPEGs, são melhores para este tipo como nós vemos no exemplo acima.

Este tipo de problema não existe quando a web começou a existir, no começo dos anos 1990 - naquele tempo somente desktops e laptops navegavam a Web, então engenheiros de navegadores e programadores nem pensavam em implementar soluções.

*Tecnologias de imagens responsivas* foram implementadas recentemente para resolver os problemas indicados acima, permitindo a você oferecer ao navegador vários arquivos de imagens, todas mostrando a mesma coisa mas contendo diferente número de pixels (mudança de resolução), ou diferentes imagens para diferente espaços de alocação (direção de arte).

**Note:** As novas funcionalidades discutidas neste artigo — `srcset`/`sizes`/`<picture>` — são todas suportadas nas versões atuais de navegadores mobile e desktop (incluindo Microsoft Edge, embora não suportada pelo Internet Explorer).

## Como você faz para criar imagens responsivas?

Nesta seção, nós iremos ver os dois problemas ilustrados acima e mostrar como resolvê-los usando funcionalidades de imagens responsivas do HTML. Você deve notar que nós iremos focar no elemento HTML `<img>` para esta seção, como visto na área de conteúdo do exemplo acima - a imagem no cabeçalho da página é somente para decoração, e assim implementada usando imagens de background CSS.

[CSS indiscutivelmente tem ferramentas melhores para design responsivo](#)

do que HTML, e nós iremos falar sobre estas ferramentas em um módulo futuro de CSS.

## Mudança de resolução: Diferentes tamanhos

Então, qual é o problema que nós queremos resolver com mudança de resolução? Nós queremos mostrar identico conteúdo da imagem, somente maior ou menor dependendo do dispositivo - esta é a situação que nós temos com a segunda imagem do conteúdo em nosso exemplo. O padrão do elemento `<img>` tradicionalmente somente permite apontar o navegador para uma única fonte:

```

```

Nós podemos, entretanto, usar dois novos atributos — `srcset` e `sizes` — para fornecer várias fontes adicionais juntamente com sugestões para ajudar o navegador a pegar a correta. Você pode ver um exemplo disso no nosso exemplo [responsive.html](#) no Github (ver também [o código fonte](#)):

```

```

Os atributos `srcset` e `sizes` parecem complicados, mas não são difíceis de entender se você formata eles como mostrado acima, com uma parte diferente do valor do atributo para cada linha. Cada valor contém uma lista separada por vírgula, e cada parte da lista é

dividida em três sub-partes. Vamos percorrer o conteúdo de cada agora:

**srcset** define o conjunto de imagens que nós iremos permitir ao navegador escolher, e qual tamanho cada imagem tem. Antes de cada vírgula nós escrevemos:

1. Um **nome do arquivo da imagem** (`elva-fairy-480w.jpg`).
2. Um espaço.
3. A **largura da imagem em pixels** (`480w`) — note que é usado a unidade `w`, e não `px` como você pode esperar. Este é o tamanho real da imagem, que pode ser encontrado no arquivo dela no seu computador (por exemplo no Mac você pode selecionar a imagem no Finder, e pressionar `Cmd + I` para mostrar as informações na tela).

**sizes** define um conjunto de condições de mídia (ex.: largura da tela) e indica qual tamanho da imagem deveria ser a melhor escolha, quando certas condições de tela são verdadeiras - Estas são as sugestões que nós falamos antes. Neste caso, antes de cada vírgula nós escrevemos:

1. Uma **condição de mídia** (`(max-width: 480px)`) — Você vai aprender mais sobre isso no [tema CSS](#), mas para agora vamos somente dizer que a condição de mídia descreve um possível estado em que a tela pode estar. Neste caso, nós estamos dizendo "quando a largura da tela é 480px ou menos".
2. Um espaço.
3. A **largura do slot da imagem** vai preencher quando a condição de mídia for verdadeira (`440px`).

**Note:** Para a largura do slot, você pode fornecer um tamanho absoluto (`px`, `em`) ou um tamanho relativo (como

porcentagem). Você pode ter notado que o último slot de largura não tem condição de mídia - isto é o padrão que será escolhido quando nenhuma condição for verdadeira. O navegador ignora tudo depois da primeira condição satisfeita, então tenha cuidado com a ordem de condições.

Então, com estes atributos no lugar, o navegador irá:

1. Ver a largura do dispositivo.
2. Ver qual condição de mídia na lista `sizes` é a primeira a ser verdadeira.
3. Ver qual é o slot para aquela condição de mídia.
4. Carregar a imagem definida na lista `srcset` que mais chega perto do tamanho do slot.

E é isto! Então neste ponto, se um navegador suportado com uma largura de 480px carregar a página, a condição (`max-width: 480px`) será verdadeira, então o slot `440px` será escolhido, então o `elva-fairy-480w.jpg` será carregada, como a largura inerente (`480w`) é a mais próxima de `440px`. A imagem de 800px é 128KB no disco enquanto que a versão de 480px é somente 63KB - economizando 65KB. Agora imagine se fosse uma página que tivesse várias imagens. Usando esta técnica poderia economizar os dados de usuários de celular.

Navegadores antigos que não suportam estas funcionalidades serão ignorados, seguiremos e carregaremos a imagem definida no atributo `src` como normal.

**Nota:** No `<head>` do documento você encontrará a linha `<meta name="viewport" content="width=device-width">`: isto força navegadores de celular adotar a largura real para

carregar páginas web (alguns navegadores mobile mentem sobre sua largura da janela, e em vez carregam páginas em uma largura grante e então encolhem a página carregada, o que é de muita ajuda para imagens e design responsivos. Nós iremos ensinar mais sobre isso em um módulo futuro).

## Ferramentas de desenvolvimento úteis

Há algumas [ferramenta de desenvolvimento](#) úteis em navegadores para ajudar a exercitar o necessário para slot de largura, etc, que você precisa usar. Quando eu estava trabalhando neles, eu primeiro carreguei a versão não responsiva do meu exemplo (`not-responsive.html`), então fui no [Modo de Design Responsivo](#) (Ferramentas > Desenvolvimento Web > Modo de Design Responsivo), que permite a você ver o layout da sua página como se ele estivesse visto através de uma variedade de diferentes tamanhos de telas.

Eu configurei a largura da janela para 320px e depois 480px; para cada uma eu fui no [DOM Inspector](#), cliquei no elemento `<img>` no qual nós estamos interessados, então olhei o tamanho na aba Box Model view no lado direito da tela. Isto deve dar para você a dica da largura de imagem que você precisa.

Próximo, você pode checar se o `srcset` está funcionando configurando a largura da janela para a qual você quiser (coloque para uma largura estreita, por exemplo), abrindo o Network Inspector (Ferramentas > Web Developer > Network), então recarregue a página. Isto deve dar a você uma lista do que foi carregado na página, e aqui você pode checar qual arquivo da imagem foi escolhida para baixar.

**Nota:** Use o Mozilla Firefox para testar `srcset`. O Chrome carrega a melhor imagem se estiver em cache no navegador, anulando o propósito do teste na ferramenta de desenvolvimento.

## Mudança de Resolução: Mesmo tamanho, diferente resoluções

Se você está dando suporte a múltiplas resoluções de vídeo, mas todas veem sua imagem no tamanho real na tela, você pode permitir ao navegador escolher uma resolução apropriada para a imagem usando `srcset` com x identificadores e sem `sizes` - uma sintaxe um pouco mais fácil! Você pode encontrar um exemplo de como isto parece em [srcset-resolutions.html](#) (ver também [o código fonte](#)):

```
](#) permite a nós implementar justamente este tipo de solução.

Voltando para o nosso exemplo [not-responsive.html](#), nós temos uma imagem que precisa de uma direção de arte:

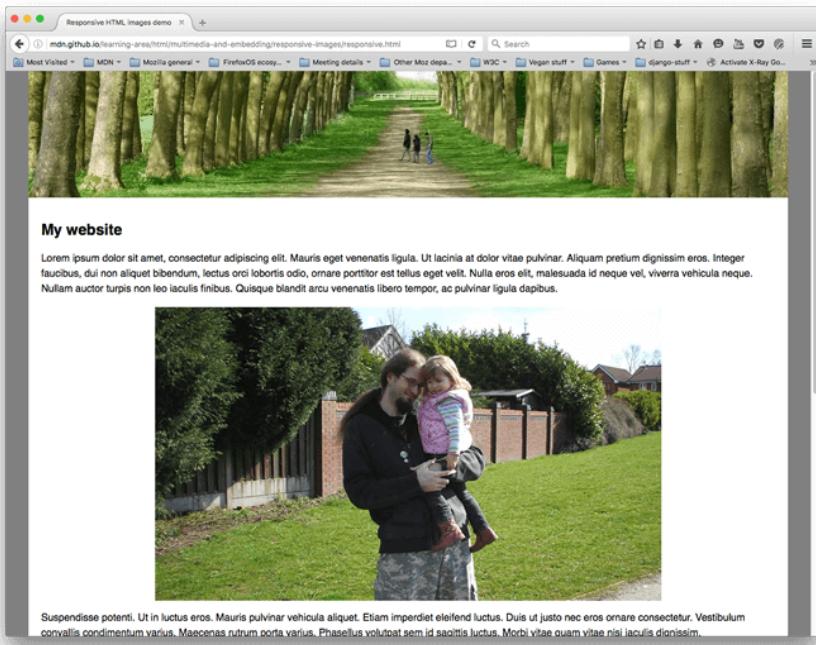
```
|
```

Vamos consertar isso, com [<picture>](#)! Como [<video>](#) e [<audio>](#), O elemento [<picture>](#) é um invólucro contendo muitos elementos [<source>](#) que fornecem muitas fontes diferentes para o navegador escolher, seguido pelo elemento mais importante [<img>](#). O código em [responsive.html](#) ficará assim então:

```
<picture>
  <source media="(max-width: 799px)" srcset="elva-800w.jpg" alt="Chris standing up high, wide shot"/>
  <source media="(min-width: 800px)" srcset="elva-800w.jpg" alt="Chris standing up high, wide shot"/>
  
</picture>
```

- Os elementos `<source>` inclui um atributo `media` que contem uma condição de mídia - como no nosso primeiro exemplo `srcset`, estas condições são testadas para qual imagem será mostrada no dispositivo - a primeira que retornar um valor verdadeiro, será escolhida. Neste caso, se a largura da janela é 799px ou menor, a primeira imagem do elemento `<source>` será mostrada. Se a largura da janela é 800px ou maior, será escolhida a segunda.
- Os atributos `srcset` contem o caminho para a imagem que será apresentada. Note que como acabamos de ver acima com `<img>`, `<source>` pode pegar um atributo `srcset` com múltiplas imagens referenciadas, e um atributo `sizes` também. Então você pode oferecer múltiplas imagens via um elemento `<picture>`, mas também oferecer múltiplas resoluções para cada uma. Na prática, você provavelmente não vai querer fazer isso com frequência.
- Em todos os casos, você deve fornecer um elemento `<img>`, com `src` e `alt`, logo antes do `</picture>`, de outra forma não aparecerá imagens. Assim um padrão será aplicado quando nenhuma condição for atendida (você pode remover o segundo elemento `<source>` neste exemplo), e verificar navegadores que não suportam o elemento `<picture>`.

Este código nos permite mostrar uma imagem adequada em ambas extensas e estreitas telas, como podemos ver abaixo:



Nota: Você deveria usar o atributo `media` somente em cenários de direção de arte; quando você usa `media`, não oferecendo também condições com o atributo `sizes`.

## Por que não podemos só fazer isso usando CSS ou JavaScript?

Quando o navegador começa a carregar a página, inicia o download de todas as imagens antes do analisador principal ter começado a carregar e interpretar o JavaScript e o CSS da página. Isto é uma técnica útil, a qual diminui o tempo de carregamento médio em 20%. Entretanto, isto não é útil para imagens responsivas, então é necessário implementar soluções como `srcset`. Você não pode, por exemplo, carregar o elemento `<img>`, então detectar a largura da janela com JavaScript e mudar dinamicamente o arquivo da imagem

para um menor, caso deseje. Até lá, a imagem original deveria já ter sido carregado, e você iria carregar uma menor, o que é ainda pior em termos de imagens responsivas.

## Use bastante formatos de imagens modernos

Há vários novos e excitantes formatos de imagens (como WebP e JPEG-2000) que podem manter baixo tamanho de arquivo e alta qualidade ao mesmo tempo. Entretanto, o suporte do navegador é menor.

<picture> permite-nos continuar dando suporte para navegadores antigos. Você pode suprir tipos MIME dentro de atributos `type`, então o navegador pode rejeitar imediatamente arquivos não suportados:

```
<picture>
  <source type="image/svg+xml" srcset="pyramid.svg"/>
  <source type="image/webp" srcset="pyramid.webp">
  
```

- *Não use o atributo `media`, a menos que você também precise de direção de arte.*
- No elemento `<source>`, você só pode refenciar imagens de tipos declarados em `type`.
- Como antes, você é encorajado a usar uma lista separada por vírgula com `srcset` e `sizes`, caso precise.

## Aprendizado ativo: Implementando suas próprias imagens responsivas

Para esse exercício, nós estamos esperando que você seja corajoso e vá sozinho.. principalmente. Nós queremos que você implemente sua própria adequada direção de arte em tela estreita/ampla usando `<picture>`, e um exemplo de mudança de resolução que use `srcset`.

1. Escreva um simples HTML contendo seu código (use `not-responsive.html` como um ponto de partida, se quiser)
2. Encontre uma boa imagem ampla de um panorama com algum detalhe contido em alguma parte. Crie uma versão de tamanho web usando um editor de imagem, então coloque para mostrar uma pequena parte que amplia o detalhe, e salve em uma segunda imagem (algo como 480px está bom).
3. Use o elemento `<picture>` para implementar uma mudança de imagem!
4. Crie múltiplos arquivos de imagem de diferentes tamanhos, cada um mostrando a mesma imagem.
5. Use `srcset`/`size` para criar um exemplo de mudança de resolução, que sirva para os mesmos tamanhos de imagens em diferentes resoluções, ou diferentes tamanhos de imagens em cada largura de janela.

**Nota:** Use a ferramenta de desenvolvimento do navegador para ajudar a ver os tamanhos que você precisa, como mencionado acima.

## Sumário

Isto é um geral sobre imagens responsivas - nós esperamos que você tenha aproveitado estas novas técnicas. Recapitulando, há dois problemas que nós discutimos aqui:

- **Direção de Arte:** O problema consiste em apresentar imagens cortadas para diferentes layouts - por exemplo, uma imagem panorâmica mostrada completa em um layout desktop, e uma imagem retrato mostrando o objeto principal ampliado em um layout mobile. Isto pode ser resolvido usando o elemento [`<picture>`](#).
- **Mudança de resolução:** O problema é apresentar arquivos menores de imagens para dispositivos estreitos, porque eles não precisam de imagens gigantes como em computadores - e também, opcionalmente, que você quer apresentar imagens de diferentes resoluções para alta e baixa densidades de tela. Isto pode resolver usando [`vector graphics`](#) (imagens SVG), e os atributos [`srcset`](#) e [`sizes`](#).

Isto também encerra o módulo [Multimedia and embedding!](#) A única coisa para fazer agora é seguir e tentar nosso teste de multimídia, e ver como você está. Se divirta.

## Veja também

- [Jason Grigsby's excelente introdução a imagens responsivas](#)  
[Imagens respondivas: Se você está mudando de resolução,](#)
- [use srcset](#)
  - Inclui mais explicação sobre como o navegador resolve qual imagem usar
- [<img>](#)
- [<picture>](#)
- [<source>](#)

## Neste Módulo

- [Imagens em HTML](#)

- [Video and audio](#)
- [De <object> a <iframe> — outras tecnologias](#)
- [Adicionando gráficos vetoriais na Web](#)
- [Imagens responsivas](#)
- [Mozilla splash page](#)

**Last modified:** 31 de jan. de 2022, [by MDN contributors](#)



# página inicial da Mozilla

Nesta avaliação, testaremos seu conhecimento de algumas das técnicas discutidas nos artigos deste módulo, fazendo com que você adicione algumas imagens e vídeos a uma página inicial divertida sobre a Mozilla!

<b>Pré-requisitos:</b>	Antes de tentar esta avaliação, você já deve ter trabalhado com o restante do módulo <a href="#">Multimídia e incorporação</a> .
<b>Objetivo:</b>	Para testar o conhecimento sobre a incorporação de imagens e vídeos em páginas da Web, quadros e técnicas de imagem responsiva HTML.

## Ponto de partida

Para começar esta avaliação, você precisa pegar o HTML e todas as imagens disponíveis no [mdn-splash-page-start](#)

diretório no GitHub. Salve o conteúdo do index.html em um



diretório (clique com o botão direito na imagem para obter uma opção para salvá-la.)

Acesse as diferentes imagens no [originais](#) diretório e salve-os da mesma maneira; você desejará salvá-los em um diretório diferente por enquanto, pois precisará manipulá-los (alguns deles) usando um editor gráfico antes de estarem prontos para serem usados.

Alternativamente, você pode usar uma ferramenta online, como [Falha](#) to create your example. This would also be useful if you want to get it assessed, or ask for help — see the [Assessment or further help](#) section at the bottom of this page.

**Note:** The example HTML file contains quite a lot of CSS, to style the page. You don't need to touch the CSS, just the HTML inside the `<body>` element — as long as you insert the correct markup, the styling will make it look correct.

## Project brief

In this assessment we are presenting you with a mostly-finished Mozilla splash page, which aims to say something nice and interesting about what Mozilla stands for, and provide some links to further resources. Unfortunately, no



Portuguese → Portuguese



---

make more sense. The following subsections detail what you need to do:

## Preparing images

Using your favorite image editor, create 400px wide and 120px wide versions of:

- `firefox_logo-only_RGB.png`
- `firefox-addons.jpg`
- `mozilla-dinosaur-head.png`

Call them something sensible, e.g. `firefoxlogo400.png` and `firefoxlogo120.png`.

Along with `mdn.svg`, these images will be your icons to link to further resources, inside the `further-info` area. You'll also link to the firefox logo in the site header. Save copies of all these inside the same directory as `index.html`.

Next, create a 1200px wide landscape version of `red-panda.jpg`, and a 600px wide portrait version that shows the panda in more of a close up shot. Again, call them something sensible so you can easily identify them. Save a copy of both of these inside the same directory as `index.html`.



Note: You should continue using ICO and PNG images to



Portuguese



Portuguese



## Adding a logo to the header

Inside the `<header>` element, add an `<img>` element that will embed the small version of the Firefox logo in the header.

## Adding a video to the main article content

Just inside the `<article>` element (right below the opening tag), embed the YouTube video found at

<https://www.youtube.com/watch?v=ojcNcvb1olg>, using the appropriate YouTube tools to generate the code. The video should be 400px wide.

## Adding responsive images to the further info links

Inside the `<div>` with the class of `Further-Info` you will find four `<a>` elements — each one linking to an interesting Mozilla-related page. To complete this section you'll need to insert an `<img>` element inside each one containing appropriate `src`, `alt`, `srcset` and `sizes` attributes.

In each case (except one — which one is inherently responsive?) we want the browser to serve the 120px wide



Make sure you match the correct images with the correct links!

**Note:** To properly test the `srcset / sizes` examples, you'll need to upload your site to a server (using [Github pages](#) is an easy and free solution), then from there you can test whether they are working properly using browser developer tools such as the Firefox [Network Monitor](#).

## An art directed red panda

Inside the [`<div>`](#) with the class of `red-panda`, we want to insert a [`<picture>`](#) element that serves the small portrait panda image if the viewport is 600px wide or less, and the large landscape image otherwise.

## Hints and tips

- You can use the [W3C Nu HTML Checker](#) to catch mistakes in your HTML.
- You don't need to know any CSS to do this assessment; you just need the provided HTML file. The CSS part is already done for you.
- The provided HTML (including the CSS styling) already does most of the work for you, so you can just focus on the media embedding.



Portuguese → Portuguese



The following screenshots show what the splash page should look like after being correctly marked up, on a wide and narrow screen display.



Portuguese → Portuguese





Portuguese → Portuguese



## Assessment or further help

If you would like your work assessed, or are stuck and want to ask for help:

1. Put your work into an online shareable editor such as [CodePen](#), [jsFiddle](#), or [Glitch](#). Glitch is probably better for this example, as it allows you upload assets like images, whereas some of the other tools don't.
2. Write a post asking for assessment and/or help at the [MDN Discourse forum "Learning" category](#). Your post should include:
  - A descriptive title such as "Assessment wanted for Mozilla splash page".
  - Details of what you have already tried, and what you would like us to do. For example, if you are stuck and need help, or want an assessment.
  - A link to the example you want assessed or need help with, in an online shareable editor (as



Portuguese → Portuguese



someone with a coding problem if you can't see their code.

- A link to the actual task or assessment page, so we can find the question you want help with.

## In this module

- [Images in HTML](#)
- [Video and audio content](#)
- [From <object> to <iframe> — other embedding technologies](#)
- [Adding vector graphics to the Web](#)
- [Responsive images](#)
- [Mozilla splash page](#)

**Última modificação:** 31 de janeiro de 2022, [por contribuidores do MDN](#)

This page was translated from English by the community.  
Learn more and join the MDN Web Docs community.

## Tabelas em HTML

Uma tarefa muito comum em HTML é estruturar os dados tabulares, e há vários elementos e atributos próprios para essa finalidade. Em conjunto com a linguagem [CSS](#) para estilização, o HTML torna fácil a exibição de tabelas com informação na Web, tais como o seu plano de lições escolares, o horário na sua piscina local, ou estatísticas sobre os seus dinossauros favoritos ou seu time de futebol favorito. Este módulo te guia por tudo que você precisa saber sobre a estruturação de dados tabulares utilizando o HTML.

## Pré-requisitos

Antes de iniciar este módulo, você deverá ter domínio dos básicos de HTML — consulte [Introdução ao HTML](#).

**Nota:** se está utilizando um computador/tablet/outro dispositivo onde não tem a possibilidade de criar os seus próprios arquivos, pode testar a maioria dos exemplos de código num programa de codificação on-line, tais como

[JSBin](#) ou [Thimble](#)

## Guias

Este módulo contém os seguintes artigos:

### [HTML - o básico sobre tabelas](#)

Este artigo apresenta as tabelas HTML, cobrindo o essencial, tal como linhas e células, cabeçalhos, como extender células por múltiplas colunas e linhas, e como agrupar todas as células numa coluna para efeitos de estilo.

### [HTML - funcionalidades avançadas de tabelas e acessibilidade](#)

No segundo artigo deste módulo, nós vamos ver algumas funcionalidades mais avançadas das tabelas HTML — tais como legendas/resumos e agrupar as suas filas no cabeçalho da tabela (head), seções de corpo (body) e rodapé (footer) — bem como, veremos sobre a acessibilidade das tabelas para os utilizadores deficientes visuais .

## Exercícios

### [Estruturar dados sobre planetas](#)

Na nossa avaliação sobre tabelas em HTML, vamos fornecer alguns dados sobre os planetas do nosso sistema solar, para que possa estruturá-los numa

tabela HTML.

**Last modified:** 1 de fev. de 2022, [by MDN contributors](#)



[This page was translated from English by the community. Learn more and join the MDN Web Docs community.](#)

## HTML table basics

Este artigo é uma introdução às tabelas HTML, cobrindo o básico, como linhas e células, cabeçalhos, fazendo as células ocuparem várias colunas e linhas e como agrupar todas as células em uma coluna para fins de estilo.

Pré-requisitos:	Noções básicas de HTML (consulte <a href="#">Introdução ao HTML</a> ).
Objetivo:	Para obter familiaridade básica com tabelas HTML.

### O que é uma tabela?

Uma tabela é um conjunto estruturado de dados composto de linhas e colunas (**dados tabulares**). Uma tabela permite consultar de forma rápida e fácil valores que indicam algum tipo de conexão entre diferentes tipos de dados, por exemplo, uma pessoa e sua idade, ou um dia da semana, ou os horários de uma piscina local.

Person	Age
Chris	38
Dennis	45
Sarah	29
Karen	47

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
<b>Public Swim</b> 06:30 - 10:30	<b>Public Swim</b> 06:30 - 09:00	<b>Public Swim</b> 06:30 - 09:00	<b>Public Swim</b> 06:30 - 11:15	<b>Public Swim</b> 06:30 - 09:00	<b>Lane Swim</b> 08:00 - 09:00	<b>Lane Swim</b> 08:00 - 09:00
<b>Aquacise</b> 10:30 - 11:15	<b>Aqua Jog</b> 09:15 - 10:00	<b>Education Swimming Lessons</b> 09:00 - 12:00	<b>Aquacise</b> 11:15 - 12:00	<b>Education Swimming Lessons</b> 09:00 - 12:00	<b>Oldham Active Kids Swimming Lessons</b> 09:00 - 13:00	<b>Public Swim</b> 09:00 - 11:00
<b>Lane Swim</b> 11:30 - 13:00	<b>Parent &amp; Baby Class</b> 09:30 - 10:15	<b>Lane Swim</b> 12:00 - 13:00	<b>Lane Swim</b> 12:00 - 13:00	<b>Lane Swim</b> 12:00 - 13:00	<b>Parent and Baby</b> 12:00 - 12:45	<b>Aquacise</b> 11:00 - 11:45
<b>Education Swimming Lessons</b>	<b>Public Swim</b> 15:00 - 16:45	<b>Public Swim</b> 15:00 - 16:45	<b>Education Swimming Lessons</b>	<b>Oldham Active Kids Swimming Lessons</b>	<b>Public Swim</b> 15:00 - 17:00	<b>Public Swim</b> 15:45 - 17:00

As tabelas são muito comumente usadas na sociedade humana, e têm sido por muito tempo, como evidenciado por este documento do Censo dos EUA de 1800:

Portanto, não é de se admirar que os criadores do HTML tenham fornecido um meio de estruturar e apresentar dados tabulares na web.

## Como funciona uma tabela?

Tabelas possuem estrutura. As informações são facilmente interpretadas fazendo associações visuais entre os cabeçalhos de linha e coluna. Veja a tabela abaixo, por exemplo, e encontre um gigante gasoso Júpiter com 62 luas. Você pode encontrar a resposta associando os cabeçalhos de linha e coluna relevantes.

Dados sobre os planetas do nosso sistema solar (fatos planetários relevantes)

		<b>Nome</b>	<b>Massa (<math>10^{24}</math> kg)</b>	<b>Diâmetro (km)</b>	<b>Densidade (kg/m<sup>3</sup>)</b>
<b>Planetas Terrestres</b>		<b>Mercúrio</b>	0.330	4,879	5427
		<b>Venus</b>	4.87	12,104	5243
		<b>Terra</b>	5.97	12,756	5514
		<b>Marte</b>	0.642	6,792	3933
<b>Planetas Jovianos</b>	<b>Gigantes Gasosos</b>	<b>Júpiter</b>	1898	142,984	1326
		<b>Saturno</b>	568	120,536	687
	<b>Gigante de gelo</b>	<b>Urano</b>	86.8	51,118	1271
		<b>Netuno</b>	102	49,528	1638
<b>Planeta Anão</b>		<b>Plutão</b>	0.0146	2,370	2095

Quando criadas corretamente, até pessoas cegas podem interpretar dados tabulares em uma tabela HTML - uma tabela HTML bem sucedida deve melhorar a experiência tanto de usuários com deficiências visuais quanto daqueles capazes de ver

## Estilizando tabelas

Você também pode dar uma [neste exemplo real](#) no GitHub! Uma coisa que você perceberá é que essa tabela parece um pouco mais legível lá — isso ocorre porque a tabela que você vê acima nesta página tem o mínimo de estilização, enquanto a versão do GitHub tem um CSS expressivo aplicado.

Não se iluda; para que tabelas sejam de fato efetivas na web, você precisa fornecer algumas informações de estilo com [CSS](#), bem como uma boa estrutura sólida com HTML. Neste módulo

nos iremos focar na parte do HTML; Para saber mais sobre a parte do CSS você deve visitar nosso [Estilizando Tabelas](#) artigo depois que você finalizar este.

Nós não focaremos em CSS neste módulo, mas nós forneceremos uma mínima folha de estilo CSS para você usar que fará suas tabelas mais legíveis que o normalmente você teria sem nenhuma estilização. Você pode procurar a [folha de estilo aqui](#), e você pode também procurar um [template HTML](#) que aplica a folha de estilo — juntos eles serão um bom ponto de partida para serem implementados com tabelas HTML.

## Quando NÃO utilizar tabelas HTML?

HTML tables should be used for tabular data — this is what they are designed for. Unfortunately, a lot of people used to use HTML tables to lay out web pages, e.g. one row to contain the header, one row to contain the content columns, one row to contain the footer, etc. You can find more details and an example at [Page Layouts](#) in our [Accessibility Learning Module](#). This was commonly used because CSS support across browsers used to be terrible; table layouts are much less common nowadays, but you might still see them in some corners of the web.

In short, using tables for layout rather than [CSS layout techniques](#) is a bad idea. The main reasons are as follows:

1. **Layout tables reduce accessibility for visually impaired users:** [Screenreaders](#), used by blind people, interpret the tags that exist in an HTML page and read out the contents to the user. Because tables are not the right tool for layout, and the markup is more complex than with CSS layout techniques, the screenreaders' output will be confusing to their users.
2. **Tables produce tag soup:** As mentioned above, table layouts generally involve more complex markup structures than proper layout techniques. This can result in the code being harder to write, maintain, and debug.
3. **Tables are not automatically responsive:** When you use proper layout containers (such as `<header>`, `<section>`, `<article>`, or `<div>`), their width defaults to 100% of their parent element. Tables on the other hand are sized according to their content by default, so extra measures are needed to get table layout styling to effectively work across a variety of devices.

## Active learning: Creating your first table

We've talked table theory enough, so, let's dive into a practical example and build up a simple table.

1. First of all, make a local copy of [blank-template.html](#) and [minimal-table.css](#) in a new directory on your local machine.
2. The content of every table is enclosed by these two tags : `<table></table>`. Add these inside the body of your HTML.

3. The smallest container inside a table is a table cell, which is created by a `<td>` element ('td' stands for 'table data'). Add the following inside your table tags:

```
<td>Hi, I'm your first cell.</td>
```

4. If we want a row of four cells, we need to copy these tags three times. Update the contents of your table to look like so:

```
<td>Hi, I'm your first cell.</td>
<td>I'm your second cell.</td>
<td>I'm your third cell.</td>
<td>I'm your fourth cell.</td>
```

As you will see, the cells are not placed underneath each other, rather they are automatically aligned with each other on the same row. Each `<td>` element creates a single cell and together they make up the first row. Every cell we add makes the row grow longer.

To stop this row from growing and start placing subsequent cells on a second row, we need to use the `<tr>` element ('tr' stands for 'table row'). Let's investigate this now.

1. Place the four cells you've already created inside `<tr>` tags, like so:

```
<tr>
  <td>Hi, I'm your first cell.</td>
  <td>I'm your second cell.</td>
  <td>I'm your third cell.</td>
  <td>I'm your fourth cell.</td>
</tr>
```

2. Now you've made one row, have a go at making one or two more — each row needs to be wrapped in an additional `<tr>` element, with each cell contained in a `<td>`.

This should result in a table that looks something like the following:

Hi, I'm your first cell.	I'm your second cell.	I'm your third cell.	I'm your fourth cell.
Second row, first cell.	Cell 2.	Cell 3.	Cell 4.

**Note:** You can also find this on GitHub as [simple-table.html](#) (see it live also ).

## Adding headers with `<th>` elements

Now let's turn our attention to table headers — special cells that go at the start of a row or column and define the type of data that row or column contains (as an example, see the

"Person" and "Age" cells in the first example shown in this article). To illustrate why they are useful, have a look at the following table example. First the source code:

```
<table>
  <tr>
    <td>&nbsp;</td>
    <td>Knocky</td>
    <td>Flor</td>
    <td>Ella</td>
    <td>Juan</td>
  </tr>
  <tr>
    <td>Breed</td>
    <td>Jack Russell</td>
    <td>Poodle</td>
    <td>Streetdog</td>
    <td>Cocker Spaniel</td>
  </tr>
  <tr>
    <td>Age</td>
    <td>16</td>
    <td>9</td>
    <td>10</td>
    <td>5</td>
  </tr>
  <tr>
    <td>Owner</td>
    <td>Mother-in-law</td>
    <td>Me</td>
    <td>Me</td>
    <td>Sister-in-law</td>
  </tr>
  <tr>
    <td>Eating Habits</td>
    <td>Eats everyone's leftovers</td>
    <td>Nibbles at food</td>
    <td>Hearty eater</td>
    <td>Will eat till he explodes</td>
  </tr>
</table>
```

Now the actual rendered table:

	Knocky	Flor	Ella	Juan
Breed	Jack Russell	Poodle	Streetdog	Cocker Spaniel

Age	16	9	10	5
Owner	Mother-in-law	Me	Me	Sister-in-law
Eating Habits	Eats everyone's leftovers	Nibbles at food	Hearty eater	Will eat till he explodes

The problem here is that, while you can kind of make out what's going on, it is not as easy to cross reference data as it could be. If the column and row headings stood out in some way, it would be much better.

## Active learning: table headers

Let's have a go at improving this table.

1. First, make a local copy of our [dogs-table.html](#) and [minimal-table.css](#) files in a new directory on your local machine. The HTML contains the same Dogs example as you saw above.
2. To recognize the table headers as headers, both visually and semantically, you can use the `<th>` element ('th' stands for 'table header'). This works in exactly the same way as a `<td>`, except that it denotes a header, not a normal cell. Go into your HTML, and change all the `<td>` elements surrounding the table headers into `<th>` elements.
3. Save your HTML and load it in a browser, and you should see that the headers now look like headers.

**Note:** You can find our finished example at [dogs-table-fixed.html](#) on GitHub ([see it live also](#)).

## Why are headers useful?

We have already partially answered this question — it is easier to find the data you are looking for when the headers clearly stand out, and the design just generally looks better.

**Note:** Table headings come with some default styling — they are bold and centered even if you don't add your own styling to the table, to help them stand out.

Tables headers also have an added benefit — along with the `scope` attribute (which we'll learn about in the next article), they allow you to make tables more accessible by associating each header with all the data in the same row or column. Screenreaders are then able to read out a whole row or column of data at once, which is pretty useful.

## Allowing cells to span multiple rows and columns

Sometimes we want cells to span multiple rows or columns. Take the following simple example, which shows the names of common animals. In some cases, we want to show the names of the males and females next to the animal name. Sometimes we don't, and in such cases we just want the animal name to span the whole table.

The initial markup looks like this:

```
<table>
  <tr>
    <th>Animals</th>
  </tr>
  <tr>
    <th>Hippopotamus</th>
  </tr>
  <tr>
    <th>Horse</th>
    <td>Mare</td>
  </tr>
  <tr>
    <td>Stallion</td>
  </tr>
  <tr>
    <th>Crocodile</th>
  </tr>
  <tr>
    <th>Chicken</th>
    <td>Hen</td>
  </tr>
  <tr>
    <td>Rooster</td>
  </tr>
</table>
```

But the output doesn't give us quite what we want:

Animals	
Hippopotamus	
Horse	Mare
Stallion	
Crocodile	

Chicken	Hen
Rooster	

We need a way to get "Animals", "Hippopotamus", and "Crocodile" to span across two columns, and "Horse" and "Chicken" to span downwards over two rows. Fortunately, table headers and cells have the `colspan` and `rowspan` attributes, which allow us to do just those things. Both accept a unitless number value, which equals the number of rows or columns you want spanned. For example, `colspan="2"` makes a cell span two columns.

Let's use `colspan` and `rowspan` to improve this table.

1. First, make a local copy of our [animals-table.html](#) and [minimal-table.css](#) files in a new directory on your local machine. The HTML contains the same animals example as you saw above.
2. Next, use `colspan` to make "Animals", "Hippopotamus", and "Crocodile" span across two columns.
3. Finally, use `rowspan` to make "Horse" and "Chicken" span across two rows.
4. Save and open your code in a browser to see the improvement.

**Note:** You can find our finished example at [animals-table-fixed.html](#) on GitHub ([see it live also](#)).

## Providing common styling to columns

There is one last feature we'll tell you about in this article before we move on. HTML has a method of defining styling information for an entire column of data all in one place — the `<col>` and `<colgroup>` elements. These exist because it can be a bit annoying and inefficient having to specify styling on columns — you generally have to specify your styling information on every `<td>` or `<th>` in the column, or use a complex selector such as [:nth-child\(1\) \(en-US\)](#).

**Note:** Styling columns like this is [limited to a few properties](#) : `border`, `background`, `width`, and `visibility`. To set other properties you'll have to either style every `<td>` or `<th>` in the column, or use a complex selector such as [:nth-child\(1\) \(en-US\)](#).

Take the following simple example:

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

```

<th>Data 1</th>
<th style="background-color: yellow">Data 2</th>
</tr>
<tr>
  <td>Calcutta</td>
  <td style="background-color: yellow">Orange</td>
</tr>
<tr>
  <td>Robots</td>
  <td style="background-color: yellow">Jazz</td>
</tr>
</table>

```

Which gives us the following result:

Data 1	Data 2
Calcutta	Orange
Robots	Jazz

This isn't ideal, as we have to repeat the styling information across all three cells in the column (we'd probably have a `class` set on all three in a real project and specify the styling in a separate stylesheet). Instead of doing this, we can specify the information once, on a `<col>` element. `<col>` elements are specified inside a `<colgroup>` container just below the opening `<table>` tag. We could create the same effect as we see above by specifying our table as follows:

```

<table>
  <colgroup>
    <col>
      <col style="background-color: yellow">
  </colgroup>
  <tr>
    <th>Data 1</th>
    <th>Data 2</th>
  </tr>
  <tr>
    <td>Calcutta</td>
    <td>Orange</td>
  </tr>
  <tr>
    <td>Robots</td>
    <td>Jazz</td>
  </tr>
</table>

```

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

Effectively we are defining two "style columns", one specifying styling information for each column. We are not styling the first column, but we still have to include a blank `<col>` element — if we didn't, the styling would just be applied to the first column.

If we wanted to apply the styling information to both columns, we could just include one `<col>` element with a `span` attribute on it, like this:

```
|<colgroup>  
|  <col style="background-color: yellow" span="2">  
|</colgroup>
```



Just like `colspan` and `rowspan`, `span` takes a unitless number value that specifies the number of columns you want the styling to apply to.

## Active learning: colgroup and col

Now it's time to have a go yourself.

Below you can see the timetable of a languages teacher. On Friday she has a new class teaching Dutch all day, but she also teaches German for a few periods on Tuesday and Thursdays. She wants to highlight the columns containing the days she is teaching.

Recreate the table by following the steps below.

1. First, make a local copy of our [timetable.html](#) file in a new directory on your local machine. The HTML contains the same table you saw above, minus the column styling information.

2. Add a `<colgroup>` element at the top of the table, just underneath the `<table>` tag, in which you can add your `<col>` elements (see the remaining steps below).
3. The first two columns need to be left unstyled.
4. Add a background color to the third column. The value for your `style` attribute is `background-color:#97DB9A;`
5. Set a separate width on the fourth column. The value for your `style` attribute is `width: 42px;`
6. Add a background color to the fifth column. The value for your `style` attribute is `background-color: #97DB9A;`
7. Adicione uma cor de fundo diferente para a sexta coluna, para indicar que este é um dia especial e ela está dando uma nova aula. Os valores do seu `style` atributo são `background-color:#DCC48E; border:4px solid #C1437A;`
8. Os últimos dois dias são dias livres, então apenas defina-os para nenhuma cor de fundo, mas uma largura definida; o valor do `style` atributo é `width: 42px;`

Veja como você segue com o exemplo. Se você tiver dúvidas ou quiser verificar seu trabalho, pode encontrar nossa versão no GitHub como [schedule-fixed.html](#) ( [veja ao vivo também](#) ).

## Resumo

Isso envolve o básico das tabelas HTML. No próximo artigo, veremos alguns recursos de mesa um pouco mais avançados e começaremos a pensar como eles são acessíveis para pessoas com deficiência visual.

## Neste módulo

- [Noções básicas de tabela HTML](#)
- [Recursos avançados e acessibilidade da tabela HTML](#)
- [Estruturação de dados do planeta](#)

**Last modified:** 31 de jan. de 2022, by [MDN contributors](#)



# Recursos avançados e acessibilidade da tabela HTML

No segundo artigo deste módulo, veremos alguns recursos mais avançados de tabelas HTML — como legendas/resumos e agrupamento de suas linhas em seções de cabeçalho, corpo e rodapé da tabela — bem como a acessibilidade de tabelas para usuários com deficiência visual .

<b>Pré-requisitos:</b>	Os fundamentos do HTML (consulte <a href="#">Introdução ao HTML</a> ).
<b>Objetivo:</b>	Para saber mais sobre os recursos de tabela HTML mais avançados e a acessibilidade das tabelas.

## Adicionando uma legenda à sua tabela com <caption>

Você pode dar uma legenda à sua tabela colocando-a dentro de um `<caption>` elemento e aninhando-a dentro do `<table>` elemento. `<table>` Você deve colocá-lo logo abaixo da tag de abertura .

```
<table>
  <caption>Dinosaurs in the Jurassic period</caption>
  ...
</table>
```



Como você pode deduzir do breve exemplo acima, a legenda deve conter uma descrição do conteúdo da tabela. Isso é útil para todos os leitores que desejam ter uma ideia rápida se a tabela é útil para eles enquanto examinam a página, mas principalmente para usuários cegos. Em vez de um leitor de tela ler o conteúdo de muitas células apenas para descobrir do que se trata a tabela, o usuário pode confiar em uma legenda e decidir se deseja ou não ler a tabela com mais detalhes.

A caption is placed directly beneath the `<table>` tag.



<caption> element instead, however, as summary is deprecated by the HTML5 spec, and can't be read by sighted users (it doesn't appear on the page.)

## Active learning: Adding a caption

Let's try this out, revisiting an example we first met in the previous article.

1. Open up your language teacher's school timetable from the end of [HTML Table Basics](#), or make a local copy of our [timetable-fixed.html](#) file.
2. Add a suitable caption for the table.
3. Save your code and open it in a browser to see what it looks like.

**Note:** You can find our version on GitHub — see [timetable-caption.html](#) ( [see it live also](#) ).

## Adding structure with <thead>, <tfoot>, and <tbody>

As your tables get a bit more complex in structure, it is useful to give them more structural definition. One clear way to do this is by using [<thead>](#), [<tfoot>](#), and [<tbody>](#), which allow you to mark up a header, footer, and body section for the table.

These elements don't make the table any more accessible to screenreader users, and don't result in any visual enhancement on their own. They are however very useful for styling and layout — acting as useful hooks for adding CSS to your table. To give you some interesting examples, in the case of a long table you could make the table header and footer repeat on every printed page, and you could make the table body display on a single page and have the contents available by scrolling up and down.

To use them:

- The <thead> element must wrap the part of the table that is the header — this is usually the first row containing the column headings, but this is not necessarily always the case. If you are using [<col>](#)/[<colgroup>](#) element, the table header should come just below those.
- The <tfoot> element needs to wrap the part of the table that is the footer — this might be a final row with items in the previous rows summed, for example. You can include the table footer right at the bottom of the table as you'd expect, or just below the table header (the browser will still render it at the bottom of the table).



depending on how you decided to structure it.

**Note:** <tbody> is always included in every table, implicitly if you don't specify it in your code. To check this, open up one of your previous examples that doesn't include <tbody> and look at the HTML code in your [browser developer tools](#) — you will see that the browser has added this tag for you. You might wonder why you ought to bother including it at all — you should, because it gives you more control over your table structure and styling.

## Active learning: Adding table structure

Let's put these new elements into action.

1. First of all, make a local copy of [spending-record.html](#) and [minimal-table.css](#) in a new folder.
2. Try opening it in a browser — You'll see that it looks OK, but it could stand to be improved. The "SUM" row that contains a summation of the spent amounts seems to be in the wrong place, and there are some details missing from the code.
3. Put the obvious headers row inside a <thead> element, the "SUM" row inside a <tfoot> element, and the rest of the content inside a <tbody> element.
4. Save and refresh, and you'll see that adding the <tfoot> element has caused the "SUM" row to go down to the bottom of the table.
5. Next, add a [colspan](#) attribute to make the "SUM" cell span across the first four columns, so the actual number appears at the bottom of the "Cost" column.
6. Let's add some simple extra styling to the table, to give you an idea of how useful these elements are for applying CSS. Inside the head of your HTML document, you'll see an empty <style> element. Inside this element, add the following lines of CSS code:

```
tbody {  
    font-size: 95%;  
    font-style: italic;  
}  
  
tfoot {  
    font-weight: bold;  
}
```

7. Save and refresh, and have a look at the result. If the <tbody> and <tfoot> elements weren't in place, you'd have to write much more complicated selectors/rules to apply the same styling.



we also have an article specifically on [styling tables](#)).

Your finished table should look something like the following:

**Note:** You can also find it on Github as [spending-record-finished.html](#) ([see it live also](#)).

## Nesting Tables

It is possible to nest a table inside another one, as long as you include the complete structure, including the `<table>` element. This is generally not really advised, as it makes the markup more confusing and less accessible to screenreader users, and in many cases you might as well just insert extra cells/rows/columns into the existing table. It is however sometimes necessary, for example if you want to import content easily from other sources.

The following markup shows a simple nested table:

```
<table id="table1">
  <tr>
    <th>title1</th>
    <th>title2</th>
    <th>title3</th>
  </tr>
  <tr>
    <td id="nested">
      <table id="table2">
        <tr>
```



Portuguese → Portuguese



```
</tr>
</table>
</td>
<td>cell2</td>
<td>cell3</td>
</tr>
<tr>
<td>cell4</td>
<td>cell5</td>
<td>cell6</td>
</tr>
</table>
```

The output of which looks something like this:

Items Sold August 2016						
		Clothes			Accessories	
		Trousers	Skirts	Dresses	Bracelets	Rings
Belgium	Antwerp	56	22	43	72	23
	Gent	46	18	50	61	15
	Brussels	51	27	38	69	28

## Tables for visually impaired users

Let's recap briefly on how we use data tables. A table can be a handy tool, for giving us quick access to data and allowing us to look up different values. For example, it takes only a short glance at the table below to find out how many rings were sold in Gent during August 2016. To understand its information we make visual associations between the data in this table and its column and/or row headers.

Items Sold August 2016

		Clothes			Accessories	
		Trousers	Skirts	Dresses	Bracelets	Rings
		56	22	43	72	23
Belgium	Antwerp	46	18	50	61	15
	Brussels	51	27	38	69	28



	Utrecht	80	12	43	36	19
--	---------	----	----	----	----	----

But what if you cannot make those visual associations? How then can you read a table like the above? Visually impaired people often use a screenreader that reads out information on web pages to them. This is no problem when you're reading plain text but interpreting a table can be quite a challenge for a blind person. Nevertheless, with the proper markup we can replace visual associations by programmatic ones.

**Note:** There are around 253 Million people living with Visual Impairment according to [WHO data in 2017](#)

This section of the article provides further techniques for making tables as accessible as possible.

## Using column and row headers

Screenreaders will identify all headers and use them to make programmatic associations between those headers and the cells they relate to. The combination of column and row headers will identify and interpret the data in each cell so that screenreader users can interpret the table similarly to how a sighted user does.

We already covered headers in our previous article — see [Adding headers with <th> elements](#).

## The scope attribute

A new topic for this article is the `scope` attribute, which can be added to the `<th>` element to tell screenreaders exactly what cells the header is a header for — is it a header for the row it is in, or the column, for example? Looking back to our spending record example from earlier on, you could unambiguously define the column headers as column headers like this:

```
<thead>
<tr>
  <th scope="col">Purchase</th>
  <th scope="col">Location</th>
  <th scope="col">Date</th>
  <th scope="col">Evaluation</th>
  <th scope="col">Cost (€)</th>
```



And each row could have a header defined like this (if we added row headers as well as column headers):

```
<tr>
  <th scope="row">Haircut</th>
  <td>Hairdresser</td>
  <td>12/09</td>
  <td>Great idea</td>
  <td>30</td>
</tr>
```

Screenreaders will recognize markup structured like this, and allow their users to read out the entire column or row at once, for example.

`scope` has two more possible values — `colgroup` and `rowgroup`. These are used for headings that sit over the top of multiple columns or rows. If you look back at the "Items Sold August 2016" table at the start of this section of the article, you'll see that the "Clothes" cell sits above the "Trousers", "Skirts", and "Dresses" cells. All of these cells should be marked up as headers (`<th>`), but "Clothes" is a heading that sits over the top and defines the other three subheadings. "Clothes" therefore should get an attribute of `scope="colgroup"`, whereas the others would get an attribute of `scope="col"`.

## The `id` and `headers` attributes

An alternative to using the `scope` attribute is to use `id` and `headers` attributes to create associations between headers and cells. The way they are used is as follows:

1. You add a unique `id` to each `<th>` element.
2. You add a `headers` attribute to each `<td>` element. Each `headers` attribute has to contain a list of the `ids` of all the `<th>` elements that act as a header for that cell, separated by spaces.

This gives your HTML table an explicit definition of the position of each cell in the table, defined by the header(s) for each column and row it is part of, kind of like a spreadsheet. For it to work well, the table really needs both column and row headers.

Returning to our spending costs example, the previous two snippets could be rewritten like this:

```
<thead>
  <tr>
```



Portuguese → Portuguese



```
<tr id="date"><td>2012-09-12</td>
<th id="evaluation">Evaluation</th>
<th id="cost">Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
  <th id="haircut">Haircut</th>
  <td headers="location haircut">Hairdresser</td>
  <td headers="date haircut">12/09</td>
  <td headers="evaluation haircut">Great idea</td>
  <td headers="cost haircut">30</td>
</tr>
...
</tbody>
```

**Note:** This method creates very precise associations between headers and data cells but it uses a **lot** more markup and does not leave any room for errors. The `scope` approach is usually enough for most tables.

## Active learning: playing with scope and headers

1. For this final exercise, we'd like you to first make local copies of [items-sold.html](#) and [minimal-table.css](#), in a new directory.
2. Now try adding in the appropriate `scope` attributes to make this table more appropriate.
3. Finally, try making another copy of the starter files, and this time make the table more accessible using `id` and `headers` attributes.

**Note:** You can check your work against our finished examples — see [items-sold-scope.html](#) ([also see this live](#)) and [items-sold-headers.html](#) ([see this live too](#)).

## Summary

There are a few other things you could learn about table HTML, but we have really given all you need to know at this moment in time. At this point, you might want to go and learn about styling HTML tables — see [Styling Tables](#).

## In this module



Portuguese → Portuguese



- 
- [Structuring planet data](#)

**Last modified:** Jan 21, 2022, [by MDN contributors](#)